aws

Aurora 用户指南

Amazon Aurora



Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon Aurora: Aurora 用户指南

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon 的商标和商业外观不得用于任何非 Amazon 的商品或服务，也不得以任何可能引起客户混淆或者贬低或诋毁 Amazon 的方式使用。所有非 Amazon 拥有的其他商标均为各自所有者的财产，这些所有者可能附属于 Amazon、与 Amazon 有关联或由 Amazon 赞助，也可能不是如此。

Table of Contents

什么是 Aurora？ 1
Amazon RDS 责任共担模式 2
Amazon Aurora 如何与 Amazon RDS 协同工作 2
Aurora 数据库集群 3
Aurora 版本 5
支持的数据库引擎 5
指定 Aurora 版本 5
Aurora 版本控制 6
Aurora 数据库集群升级 11
Aurora 版本支持 13
区域和可用区 15
AWS 地区 16
可用区 23
数据库集群的本地时区 23
不同区域和引擎支持的 Aurora 功能 29
表约定 30
蓝绿部署 30
Aurora 集群配置 30
数据库活动流 31
将集群数据导出到 Amazon S3 37
将快照数据导出到 Amazon S3 39
Aurora 全局数据库 40
IAM 数据库身份验证 47
Kerberos 身份验证 48
Aurora 机器学习 53
Performance Insights 61
零 ETL 集成 69
RDS 代理 72
Secrets Manager 集成 79
Aurora Serverless v2 80
Aurora Serverless v1 85
RDS 数据 API 90
零停机时间修补（ZDP） 96
引擎原生功能 97

端点连接 97
Aurora 端点的类型 98
查看终端节点 99
终端节点和高可用性 100
集群端点 100
读取器终端节点 101
实例端点 102
自定义端点 103
数据库实例类 121
数据库实例类类型 121
支持的数据库引擎 124
确定 AWS 区域 中的数据库实例类支持 131
硬件规格 135
存储 142
Aurora 存储概述 142
集群卷内容 143
Aurora 集群存储配置 143
存储大小的调整方式 144
数据计费 144
可靠性 145
存储自动修复 145
自动恢复页面缓存 145
从计划外重启中恢复 146
Aurora 安全性 147
将 SSL 与 Aurora 数据库集群配合使用 148
高可用性 148
Aurora 数据的高可用性 149
Aurora 数据库实例的高可用性 149
使用 Aurora Global Database 跨 AWS 区域的高可用性 150
容错能力 150
Amazon RDS 代理的高可用性 151
复制 152
Aurora 副本 152
Aurora MySQL 154
Aurora PostgreSQL 154
Aurora 的数据库实例计费 155

按需数据库实例 157
预留数据库实例 158
设置环境 172
注册 AWS 账户 172
创建具有管理访问权限的用户 172
授权以编程方式访问 174
确定要求 175
提供对数据库集群的访问权限 176
开始使用 179
创建 Aurora MySQL 数据库集群并连接到该集群 179
先决条件 181
步骤 1：创建 EC2 实例 181
步骤 2：创建 Aurora MySQL 数据库集群 186
（可选）使用 AWS CloudFormation 创建 VPC、EC2 实例和 Aurora MySQL 集群 190
步骤 3：连接到 Aurora MySQL 数据库集群 192
步骤 4：删除 EC2 实例和数据库集群 195
（可选）删除使用 CloudFormation 创建的 EC2 实例和数据库集群 196
（可选）将您的数据库集群连接到 Lambda 函数 196
创建 Aurora PostgreSQL 数据库集群并连接到该集群 196
先决条件 198
步骤 1：创建 EC2 实例 198
步骤 2：创建 Aurora PostgreSQL 数据库集群 203
（可选）使用 AWS CloudFormation 创建 VPC、EC2 实例和 Aurora PostgreSQL 集群 208
步骤 3：连接到 Aurora PostgreSQL 数据库集群 210
步骤 4：删除 EC2 实例和数据库集群 212
（可选）删除使用 CloudFormation 创建的 EC2 实例和数据库集群 213
（可选）将您的数据库集群连接到 Lambda 函数 213
教程：创建 Web 服务器和 Amazon Aurora 数据库集群 214
启动 EC2 实例以连接数据库集群 215
创建数据库集群 221
安装 Web 服务器 232
教程和示例代码 244
本指南中的教程 244
其他 AWS 指南中的教程 245
Amazon Aurora PostgreSQL 的 AWS 研讨会和实验室内容门户 246
Amazon Aurora MySQL 的 AWS 研讨会和实验室内容门户 247

GitHub 中的教程和示例代码 248
使用 AWS SDK 249
以编程方式访问 Amazon Aurora 251
Console-to-Code 251
设置 Aurora 数据库集群 253
创建数据库集群 254
先决条件 255
创建数据库集群 260
可用的设置 270
不适用于 Aurora 的据库集群设置 287
不适用于 Aurora 数据库实例的设置 288
使用 AWS CloudFormation 创建资源 291
Aurora 和 AWS CloudFormation 模板 291
了解有关 AWS CloudFormation 的更多信息 291
连接到数据库集群 292
使用 AWS 驱动程序连接到 Aurora 数据库集群 293
连接到 Aurora MySQL 294
连接到 Aurora PostgreSQL 301
排除连接故障 302
参数组 304
参数组概述 304
数据库集群参数组 307
数据库参数组 324
比较数据库参数组 340
指定数据库参数 340
将数据迁移到数据库集群 345
Aurora MySQL 345
Aurora PostgreSQL 345
从 Amazon RDS 创建 ElastiCache 缓存 346
使用 Aurora 数据库集群设置创建 ElastiCache 缓存的概述 346
使用 Aurora 数据库集群中的设置创建 ElastiCache 缓存 347
自动迁移 EC2 数据库 350
概览 350
先决条件 351
限制 352
创建 IAM 资源 352

设置数据迁移 360
管理迁移 361
监控 363
教程：使用自定义参数组创建 MySQL 数据库集群 365
先决条件 365
创建自定义参数组 366
修改参数值 366
创建数据库集群 367
管理 Aurora 数据库集群 368
停止和启动集群 369
停止和启动集群概述 369
限制 370
停止数据库集群 370
停止数据库集群后 371
启动数据库集群 372
连接 EC2 实例 374
概述 375
连接 EC2 实例 378
查看连接的计算资源 381
连接到运行特定数据库引擎的数据库实例 382
连接 Lambda 函数 383
概览 385
连接 Lambda 函数 392
查看连接的计算资源 394
修改 Aurora 数据库集群 395
使用控制台、CLI 和 API 修改数据库集群 395
修改数据库集群中的数据库实例 397
更改主用户密码 399
可用的设置 401
不适用于 Aurora 数据库集群的设置 432
不适用于 Aurora 数据库实例的设置 433
添加 Aurora 副本 435
具有 Aurora 副本的 Auto Scaling 441
添加自动扩缩策略 446
编辑自动扩缩策略 457
删除自动扩缩策略 459

管理性能和扩展 462
存储扩展 462
实例扩展 468
读取扩展 468
管理连接 469
管理查询执行计划 469
克隆 Aurora 数据库集群卷 470
Aurora 克隆概述 470
Aurora 克隆的限制 471
Aurora 克隆的工作原理 472
创建 Aurora 克隆 475
跨 VPC 克隆 485
跨账户克隆 501
与 AWS 服务集成 516
Aurora MySQL 516
Aurora PostgreSQL 516
维护 Aurora 数据库集群 518
数据库集群维护更新概述 518
查看待处理维护 519
应用更新 522
维护时段 524
调整数据库集群的维护时段 527
Aurora 数据库集群的自动次要版本升级 528
选择 Aurora MySQL 维护更新的频率 531
重启 Aurora 数据库集群或实例 533
重启 Aurora 集群内的数据库实例 533
在具有读取可用性的情况下重启 Aurora 集群 534
在没有读取可用性功能的情况下重启 Aurora 集群 536
检查 Aurora 集群和实例的正常运行时间 537
Aurora 重启操作示例 540
对 Aurora 数据库集群进行失效转移 555
删除 Aurora 集群和实例 557
删除 Aurora 数据库集群 557
Aurora 集群的删除保护 564
删除已停止的 Aurora 集群 565
删除作为只读副本的 Aurora MySQL 集群 565

删除集群时的最终快照 565
从 Aurora 数据库集群中删除数据库实例 565
为 Aurora 和 RDS 资源添加标签 568
为什么要使用 RDS 标签？ 569
RDS 标签的工作原理 570
最佳实践 572
复制标签到数据库集群快照 573
在 Amazon RDS 中添加和删除标签 574
教程：使用标签指定要停止哪些 Aurora 数据库集群 578
Amazon RDS 中的 ARN 581
构建 ARN 581
获取现有 ARN 588
Aurora 更新 591
确定您的 Amazon Aurora 版本 591
RDS Extended Support 593
RDS 扩展支持概述 594
RDS 扩展支持费用 594
避免支付 RDS 扩展支持费用 595
提供 RDS Extended Support 版本 595
对于 RDS 扩展支持的责任 595
Amazon Aurora 的责任 596
您的责任 596
创建Aurora 数据库集群或全局集群 596
RDS 扩展支持行为 597
使用 RDS 扩展支持时的注意事项 597
使用 RDS 扩展支持创建Aurora 数据库集群或全局集群 598
查看 RDS 扩展支持注册情况 599
查看支持日期 601
还原Aurora 数据集群或全局集群 603
RDS 扩展支持行为 604
使用 RDS 扩展支持时的注意事项 604
使用 RDS 扩展支持还原 Aurora 数据库集群或全局集群 605
使用蓝绿部署进行数据库更新 607
Amazon RDS 蓝绿部署概述 608
区域和版本可用性 608
优势 609

工作流 ... 609
授予访问权限 ... 614
限制和注意事项 ... 615
最佳实践 ... 621
创建蓝绿部署 ... 624
准备进行蓝绿部署 ... 624
指定更改 ... 625
创建蓝绿部署 ... 626
可用设置 ... 628
查看蓝绿部署 ... 629
切换蓝绿部署 ... 633
切换超时 ... 634
切换防护机制 ... 634
切换操作 ... 635
切换最佳实践 ... 636
在切换之前验证 CloudWatch 指标 ... 637
在切换之前监控副本滞后 ... 637
切换蓝绿部署 ... 638
切换后 ... 640
删除蓝绿部署 ... 642
备份和还原 Aurora 数据库集群 ... 646
备份和还原概述 ... 647
备份 ... 647
备份时段 ... 648
还原数据 ... 650
数据库克隆 ... 651
回溯 ... 651
保留自动备份 ... 652
保留期 ... 652
查看保留的备份 ... 653
保留成本 ... 653
限制 ... 653
删除保留的自动备份 ... 654
备份存储 ... 656
自动备份存储 ... 656
快照存储 ... 656

CloudWatch 备份存储指标 656
计算备份存储使用量 657
常见问题解答 658
创建数据库集群快照 661
确定快照是否可用 663
从数据库集群快照还原 664
参数组 664
安全组 665
Aurora 注意事项 665
从快照还原 665
数据库集群快照复制 668
使用 AWS Management Console复制数据库集群快照 668
限制 669
注意事项 670
复制未加密的数据库集群快照 672
复制加密的数据库集群快照 674
跨账户复制数据库集群快照 677
共享数据库集群快照 681
共享快照 681
共享公有快照 685
共享加密的快照 686
停止快照共享 690
将数据库集群数据导出到 Amazon S3 692
注意事项 693
设置对 S3 存储桶的访问权限 695
创建数据库集群导出任务 698
监控数据库集群导出 702
取消数据库集群导出 704
故障排除 705
将数据库集群快照数据导出到 Amazon S3 708
注意事项 710
设置对 S3 存储桶的访问权限 720
创建快照导出任务 725
监控快照导出 729
取消快照导出 731
Aurora MySQL 中的导出性能 732

故障排除 733
时间点故障恢复 735
将数据库集群还原到某个时间点 736
从保留的自动备份中进行时间点恢复 738
使用 AWS Backup 的时间点恢复 741
删除数据库集群快照 747
删除数据库集群快照 747
教程：从快照还原数据库集群 749
使用控制台还原数据库集群 749
使用 AWS CLI 还原数据库集群 753
监控 Aurora 数据库集群中的指标 760
监控计划 760
性能基准 761
性能准则 761
监控工具 762
自动监控工具 762
手动监控工具 764
查看集群状态 765
查看数据库集群 766
查看数据库集群状态 772
查看 Aurora 集群中的数据库实例状态 775
来自 Amazon Aurora 的建议 781
查看建议 783
应用建议 790
忽略建议 795
将已忽略的建议修改为活动状态 796
建议参考 798
在 Amazon RDS 控制台中查看指标 810
查看性能详情控制面板 811
从“监控”选项卡中选择新版监控视图 812
从性能详情页面中选择新版监控视图 813
创建自定义控制面板 814
选择预配置控制面板 817
使用 CloudWatch 监控 Aurora 819
Amazon Aurora 和 Amazon CloudWatch 概述 820
查看 CloudWatch 指标 821

将 Performance Insights 指标导出到 CloudWatch 827
创建 CloudWatch 告警 832
使用数据库洞察进行监控 834
定价 834
引擎、区域和实例类支持 834
开启高级模式 839
开启标准模式 843
监控慢速查询 847
注意事项 849
使用 Performance Insights 监控数据库负载 850
Performance Insights 概述 851
打开和关闭性能详情 861
Aurora MySQL 的性能架构 867
Performance Insights 策略 871
使用 Performance Insights 控制面板分析指标 884
查看性能详情主动建议 915
使用 Performance Insights API 检索指标 917
使用 AWS CloudTrail 记录 Performance Insights 调用 940
VPC 端点 (AWS PrivateLink) 942
使用针对 RDS 的 DevOps Guru 分析性能 946
DevOps Guru for RDS 的优势 946
适用于 RDS 的 DevOps Guru 的工作原理 947
设置适用于 RDS 的 DevOps Guru 949
使用增强监控来监控操作系统 956
增强监测概述 956
设置和启用增强监控 957
在 RDS 控制台中查看操作系统指标 964
使用 CloudWatch Logs 查看操作系统指标 966
Aurora 指标参考 967
Aurora 的 CloudWatch 指标 967
Aurora 的 CloudWatch 维度 995
Amazon RDS 控制台中 Aurora 指标的可用性 996
Performance Insights 的 CloudWatch 指标 1000
性能详情的计数器指标 1003
Performance Insights 的 SQL 统计数据 1027
增强监控中的操作系统指标 1036

监控事件、日志和数据库活动流 1045
在 Amazon RDS 控制台中查看日志、事件和流 1046
监控 Aurora 事件 1050
Aurora 事件概述 1050
查看 Amazon RDS 事件 1052
使用 Amazon RDS 事件通知 1056
创建对 Amazon Aurora 事件触发的规则 1080
Aurora 的 Amazon RDS 事件类别和事件消息 1084
监控 Aurora 日志 1101
查看和列出数据库日志文件 1101
下载数据库日志文件 1102
监视数据库日志文件 1104
发布到 CloudWatch Logs 1105
使用 REST 读取日志文件内容 1108
MySQL 数据库日志文件 1109
PostgreSQL 数据库日志文件 1117
监控 CloudTrail 中的 Aurora API 调用 1126
CloudTrail 与 Amazon Aurora 集成 1126
Amazon Aurora 日志文件条目 1127
使用数据库活动流监控 Aurora 1131
概览 1131
Aurora MySQL 网络先决条件 1134
启动数据库活动流 1136
获取活动流状态 1139
停止数据库活动流 1140
监控活动流 1141
活动流的 IAM 策略示例 1175
使用 GuardDuty RDS 保护功能监控威胁 1178
使用 Aurora MySQL 1179
Aurora MySQL 概述 1179
Amazon Aurora MySQL 性能增强 1180
Aurora MySQL 和空间数据 1181
与 MySQL 8.0 兼容的 Aurora MySQL 版本 3 1182
与 MySQL 5.7 兼容的 Aurora MySQL 版本 2 1211
使用 Aurora MySQL 实现高安全性 1214
Aurora MySQL 中的主用户权限 1215

TLS 连接 1216
更新应用程序以使用新的 TLS 证书 1223
确定是否有任何应用程序正使用 TLS 连接到 Aurora MySQL 数据库集群 1223
确定客户端是否需要证书验证才能连接 1224
更新应用程序信任存储 1225
用于建立 TLS 连接的示例 Java 代码 1226
对 Aurora MySQL 使用 Kerberos 身份验证 1228
Aurora MySQL 的 Kerberos 身份验证概述 1229
限制 1230
为 Aurora MySQL 设置 Kerberos 身份验证 1231
使用 Kerberos 身份验证连接到 Aurora MySQL 1240
在域中管理数据库集群 1243
将数据迁移到 Aurora MySQL 1245
从外部 MySQL 数据库迁移到 Aurora MySQL 1250
从 MySQL 数据库实例迁移到 Aurora MySQL 1274
管理 Aurora MySQL 1296
管理 Amazon Aurora MySQL 性能和扩展 1296
回溯数据库集群 1306
使用错误注入查询测试 Amazon Aurora MySQL 1326
使用快速 DDL 在 Amazon Aurora 中修改表 1330
显示 Aurora 数据库集群的卷状态 1336
优化 Aurora MySQL 1337
Aurora MySQL 优化的基本概念 1337
使用等待事件优化 Aurora MySQL 1340
使用线程状态优化 Aurora MySQL 1387
使用 Amazon DevOps Guru 主动见解优化 Aurora MySQL 1393
Aurora MySQL 的并行查询 1399
并行查询概述 1399
创建并行查询集群 1402
打开和关闭并行查询 1406
优化并行查询 1409
验证并行查询使用情况 1413
监控并行查询 1417
用于并行查询的 SQL 构造 1421
在 Aurora MySQL 中使用高级审计 1440
启用高级审核 1440

查看审核日志 1443
审计日志详细信息 1443
使用 Aurora MySQL 进行复制 1445
Aurora 副本 1445
复制选项 1446
复制性能 1447
零停机重启 (ZDR) 1448
复制筛选条件 1449
监控 复制 1456
跨区域复制 1457
二进制日志（binlog）复制 1470
基于 GTID 的复制 1512
本地写入转发 1518
启用本地写入转发 1519
检查数据库集群是否启用了写入转发 1520
应用程序和 SQL 与写入转发的兼容性 1521
写入转发的隔离级别 1523
读取一致性 1523
运行带写入转发的多部分语句 1527
使用写入转发的事务 1527
写入转发的配置参数 1527
写入转发的指标 1529
识别转发的事务和查询 1532
将 Aurora MySQL 与AWS服务集成 1534
授权 Aurora MySQL 访问AWS服务 1534
从 Amazon S3 中的文本文件加载数据 1551
将数据保存到 Amazon S3 中的文本文件 1564
从 Aurora MySQL 中调用 Lambda 函数 1573
将 Aurora MySQL 日志发布到 CloudWatch Logs 1584
Aurora MySQL 实验室模式 1589
Aurora 实验室模式功能 1589
Aurora MySQL 最佳实践 1590
确定您连接到的数据库实例 1591
性能和扩展最佳实践 1591
高可用性最佳实践 1599
关于 MySQL 功能的建议 1601

使用 Amazon CloudWatch 指标评估 Aurora MySQL 数据库实例使用情况 1607
排除 Aurora MySQL 性能故障 1609
AWS 监控选项 1609
数据库性能问题的最常见原因 1610
排查工作负载问题 1611
Aurora MySQL 的日志记录 1638
数据库连接问题排查 1640
排除查询性能故障 1653
Aurora MySQL 参考 1658
配置参数 1658
全局状态变量 1704
等待事件 1715
线程状态 1720
隔离级别 1724
提示 1729
存储过程参考 1733
information_schema 表 1777
Aurora MySQL 更新 1783
检查版本号 1783
长期支持版本和测试版 1785
为终止使用 Aurora MySQL 版本 2 做好准备 1787
为终止使用 Aurora MySQL 版本 1 做好准备 1791
升级 Amazon Aurora MySQL 数据库集群 1794
Amazon Aurora MySQL 的数据库引擎更新和修复 1906
使用 Aurora PostgreSQL 1908
数据库预览环境 1909
支持的数据库实例类类型 1910
预览环境中不支持的功能 1911
在预览环境中创建新数据库集群 1911
数据库预览环境中的 PostgreSQL 版本 17 1913
使用 Aurora PostgreSQL 实现高安全性 1914
了解 PostgreSQL 角色和权限 1915
利用 SSL/TLS 保护 Aurora PostgreSQL 数据 1929
更新应用程序以使用新的 SSL/TLS 证书 1942
确定是否有应用程序正使用 SSL 连接到 Aurora PostgreSQL 数据库集群 1942
确定客户端是否需要证书验证才能连接 1943

更新应用程序信任存储 1944
对不同类型的应用程序使用 SSL/TLS 连接 1944
使用 Kerberos 身份验证 1945
区域和版本可用性 1946
Kerberos 身份验证概述 1946
设置 1947
在 Active Directory 域中管理 Aurora PostgreSQL 数据库集群 1960
使用 Kerberos 身份验证连接 1961
使用 AD 安全组进行 Aurora PostgreSQL 访问控制 1964
将数据迁移到 Aurora PostgreSQL 1974
使用快照迁移 RDS for PostgreSQL 数据库实例 1975
使用 Aurora 只读副本迁移 RDS for PostgreSQL 数据库实例 1981
优化 Aurora PostgreSQL 中的查询性能 1993
使用 Aurora 优化型读取功能提高查询性能 1993
优化 Aurora PostgreSQL 中的关联子查询 1999
使用自适应联接提高查询性能 2004
在 Aurora PostgreSQL 中使用未记录的表 2006
创建未记录的表 2006
将未记录的表转换为已记录的表 2007
未记录的表和逻辑复制 2007
使用 PostgreSQL Autovacuum 2008
为 Autovacuum 分配内存 2008
减少事务 ID 重叠的可能性 2010
确定数据库中的表是否需要 vacuum 操作 2010
确定哪些表当前符合 Autovacuum 条件 2012
确定 Autovacuum 当前是否正在运行以及运行时长 2013
执行手动 vacuum 冻结 2014
在 Autovacuum 正在运行时重新为表建立索引 2016
使用大型索引管理 autovacuum 2017
其他影响 Autovacuum 的参数 2020
设置表级别 Autovacuum 参数 2021
记录 autovacuum 和 vacuum 活动 2021
了解对无效数据库使用 autovacuum 的行为 2022
识别 vacuum 拦截器 2025
使用 Babelfish for Aurora PostgreSQL 2046
Babelfish 限制 2048

了解 Babelfish 架构和配置 2049
创建 Babelfish for Aurora PostgreSQL 数据库集群 2088
将 SQL Server 数据库迁移到 Babelfish 2097
适用于 Aurora PostgreSQL 的 Babelfish 的数据库身份验证 2107
连接到 Babelfish 数据库集群 2124
使用 Babelfish 2137
Babelfish 问题排查 2205
关闭 Babelfish 2206
管理 Babelfish 版本更新 2207
Babelfish 参考 2228
Aurora PostgreSQL 的性能和扩展 2292
扩展 Aurora PostgreSQL 数据库实例 2292
最大连接数 2293
临时存储限制 2294
Aurora PostgreSQL 的大页 2298
使用错误注入查询测试 Amazon Aurora PostgreSQL 2299
显示 Aurora 数据库集群的卷状态 2303
指定 stats_temp_directory 的 RAM 磁盘 2304
使用 PostgreSQL 管理临时文件 2305
使用 Aurora PostgreSQL 的等待事件进行优化 2310
Aurora PostgreSQL 优化的基本概念 2312
Aurora PostgreSQL 等待事件 2316
Client:ClientRead 2318
Client:ClientWrite 2321
CPU 2323
IO:BufFileRead 和 IO:BufFileWrite 2328
IO:DataFileRead 2335
IO:XactSync 2345
IPC:DamRecordTxAck 2347
IPC:并行等待事件 2348
Lock:advisory 2354
Lock:extend 2357
Lock:Relation 2359
Lock:transactionid 2365
Lock:tuple 2368
LWLock:buffer_content (BufferContent) 2372

LWLock:buffer_mapping 2373
LWLock:BufferIO (IPC:BufferIO) 2376
LWLock:lock_manager 2377
LWLock:MultiXact 2381
LWLock:pg_stat_statements 2385
Timeout:PgSleep 2388
使用 Amazon DevOps Guru 主动见解优化 Aurora PostgreSQL 2389
数据库具有长时间运行的事务空闲连接 2390
Aurora PostgreSQL 的最佳实践 2393
避免 Aurora PostgreSQL 数据库实例性能降低、自动重启和失效转移 2393
诊断表和索引膨胀 2393
改进了 Aurora PostgreSQL 中的内存管理 2397
快速故障转移 2399
故障转移后的快速恢复 2409
管理连接流失 2415
PostgreSQL 中的失效连接处理 2422
调整 Aurora PostgreSQL 的内存参数 2425
使用 CloudWatch 指标分析资源使用情况 2432
使用逻辑复制执行主要版本升级 2436
排查 Aurora PostgreSQL 中的存储问题 2444
使用 Aurora PostgreSQL 进行复制 2445
Aurora 副本 2445
提高 Aurora 副本的读取可用性 2446
监控 复制 2448
逻辑复制概述 2448
设置逻辑复制 2449
关闭逻辑复制 2451
监控直写缓存和逻辑插槽 2451
示例：使用逻辑复制 2453
示例：使用 Aurora PostgreSQL 和 AWS DMS 的逻辑复制 2455
Aurora PostgreSQL 中的本地写入转发 2457
Aurora PostgreSQL 中的本地写入转发的限制和注意事项 2458
配置 Aurora PostgreSQL 以进行本地写入转发 2459
在 Aurora PostgreSQL 中使用本地写入转发 2462
监控 Aurora PostgreSQL 中的本地写入转发 2465
使用 Aurora PostgreSQL 作为 Amazon Bedrock 的知识库 2470

先决条件 2471
准备将 Aurora PostgreSQL 用作知识库 2471
在 Bedrock 控制台中创建知识库 2474
为 Amazon Bedrock 快速创建 Aurora PostgreSQL 知识库 2475
将 Aurora PostgreSQL 与AWS服务集成 2477
将 Amazon S3 中的数据导入到 Aurora PostgreSQL 2478
将 PostgreSQL 数据导出到 Amazon S3 2496
从 Aurora PostgreSQL 中调用 Lambda 函数 2511
将 Aurora PostgreSQL 日志发布到 CloudWatch Logs 2525
监控 Aurora PostgreSQL 的查询执行计划和峰值内存 2537
使用 Aurora 函数访问查询执行计划和峰值内存 2537
Aurora PostgreSQL 查询执行计划的参数参考 2538
管理 Aurora PostgreSQL 的查询执行计划 2542
Aurora PostgreSQL 查询计划管理概览 2542
Aurora PostgreSQL 查询计划管理的最佳实践 2549
查询计划管理 2551
捕获 Aurora PostgreSQL 执行计划 2553
使用 Aurora PostgreSQL 托管式计划 2555
在 dba_plans 视图中检查 Aurora PostgreSQL 查询计划 2559
改进计划 2560
删除计划 2564
导出和导入托管式计划 2566
参数参考 2567
函数参考 2572
apg_plan_mgmt.dba_plans 视图参考 2583
查询计划管理高级功能 2586
使用扩展和外部数据包装器 2599
使用适用于 PostgreSQL 的 Amazon Aurora 委派扩展支持 2600
使用 lo 模块更高效地管理大型对象 2612
使用 PostGIS 管理空间数据 2615
使用 pg_partman 扩展名管理分区 2624
使用 pg_cron 扩展计划维护 2629
使用 pgAudit 记录数据库活动 2637
使用 pglogical 同步数据 2649
Amazon Aurora PostgreSQL 中支持的外部数据包装器 2661
使用适用于 PostgreSQL 的可信语言扩展 2676

术语 2677
使用可信语言扩展的要求 2677
设置可信语言扩展 2680
可信语言扩展概述 2683
创建 TLE 扩展 2685
从数据库中删除 TLE 扩展 2689
卸载可信语言扩展 2691
在您的 TLE 扩展中使用 PostgreSQL 挂钩 2691
可信语言扩展的函数参考 2697
可信语言扩展的挂钩参考 2709
Aurora PostgreSQL 参考 2713
用于 EBCDIC 和其他大型机迁移的 Aurora PostgreSQL 排序规则 2713
Aurora PostgreSQL 中支持的排序规则 2715
Aurora PostgreSQL 函数参考 2715
Aurora PostgreSQL 参数 2770
Aurora PostgreSQL 等待事件 2823
Aurora PostgreSQL 更新 2849
确定 Amazon Aurora PostgreSQL 版本 2849
Aurora PostgreSQL 版本 2851
Aurora PostgreSQL 的扩展版本 2852
升级 Amazon Aurora PostgreSQL 数据库集群 2852
使用长期支持（LTS）版本 2886
使用 Aurora PostgreSQL Limitless Database 2888
Limitless Database 架构 2889
关键术语 2891
表类型 2892
计费 2892
Limitless Database 入门 2893
Limitless Database 要求和注意事项 2894
要求 2894
注意事项 2895
不支持的功能 2897
使用 Limitless Database 的先决条件 2898
启用数据库分片组操作 2898
创建使用 Limitless Database 的数据库集群 2899
将最大容量与路由器和分片相关联 2899

创建数据库集群 2902
使用数据库分片组 2911
连接到您的 Aurora PostgreSQL Limitless Database 数据库集群 2912
查找数据库分片组中路由器和分片的数量 2913
描述数据库分片组 2913
重新启动数据库分片组 2914
更改数据库分片组的容量 2915
拆分分片 2917
添加路由器 2922
删除数据库分片组 2926
向现有 Limitless Database 数据库集群添加数据库分片组 2927
创建 Limitless Database 表 2932
使用变量创建无限表 2933
将标准表转换为无限表 2938
示例架构 2942
将数据加载到 Limitless Database 2943
在 Limitless Database 中使用 COPY 命令 2945
使用 Limitless Database 数据加载实用程序 2947
查询 Limitless Database 2974
单分片查询 2975
分布式查询 2984
分布式查询跟踪 2985
分布式死锁 2993
管理 Limitless Database 2995
数据库和表大小的考虑事项 2995
通过清理回收存储空间 2996
监控 Limitless Database 3000
使用 CloudWatch 监控 Limitless Database 3001
使用 Database Insights 监控无限制数据库 3005
使用 CloudWatch Logs 监控 Limitless Database 3016
使用增强监控指标监控 Limitless Database 3016
使用性能详情监控 Limitless Database 3018
使用 GuardDuty RDS 保护功能监控 Limitless Database 3026
Limitless Database 的函数和视图 3027
Limitless Database 的等待事件 3051
备份和还原 Limitless Database 3062

备份使用 Aurora PostgreSQL Limitless Database 的数据库集群 3062
还原使用 Aurora PostgreSQL Limitless Database 的数据库集群 3064
不支持 PostgreSQL 备份和还原实用程序 3067
升级 Limitless Database 3068
升级使用 Aurora PostgreSQL Limitless Database 的数据库集群 3068
Limitless Database 参考 3070
DDL 命令 3070
DDL 限制和信息 3079
DML 命令 3115
DML 限制和信息 3118
变量 3122
数据库集群参数 3123
使用 Aurora Global Database 3124
Aurora Global Database 概览 3124
Amazon Aurora Global Database 的优势 3126
区域和版本可用性 3126
Aurora Global Database 的限制 3126
Aurora Global Database 入门 3128
配置要求 3129
创建全球数据库 3130
添加辅助集群 3142
创建无外设辅助集群 3147
从快照创建全球数据库 3149
管理 Aurora 全局数据库 3150
修改 Aurora 全局数据库 3151
修改全局数据库参数 3153
从 Aurora 全局数据库删除集群 3153
删除 Aurora 全局数据库 3156
标记 Aurora Global Database 3158
连接到 Aurora Global Database 3162
选择满足您的应用程序需求的端点 3162
查看全局数据库的端点 3163
使用全局写入器端点时的注意事项 3166
在 Aurora 全局数据库中使用写入转发 3167
在 Aurora MySQL 中使用写入转发 3167
在 Aurora PostgreSQL 中使用写入转发 3184

在 Aurora Global Database 中使用切换或失效转移 3197
计划 BCDR 3198
执行切换 3199
从计划外停机中恢复 3204
管理 RPO（Aurora PostgreSQL） 3212
辅助集群的跨区域弹性 3217
监控 Aurora 全局数据库 3218
使用性能详情监控 Aurora 全局数据库 3219
使用数据库活动流监控 Aurora Global Database 3220
监控基于 Aurora MySQL 的全局数据库 3220
监控基于 Aurora PostgreSQL 的全局数据库 3223
将 Aurora Global Database 与其他AWS服务结合使用 3226
升级 Amazon Aurora Global Database 3227
主要版本升级 3227
次要版本升级 3228
Amazon RDS 代理 3231
区域和版本可用性 3232
配额和限制 3232
MySQL 限制 3233
PostgreSQL 限制 3234
规划在哪里使用 RDS 代理 3235
RDS 代理概念和术语 3236
RDS 代理概念概述 3237
连接池 3237
安全性 3238
失效转移 3239
事务 3240
开始使用 RDS 代理 3241
设置代理网络 3241
设置数据库凭证 3244
设置 IAM 策略 3248
创建代理 3250
查看代理 3257
通过 RDS Proxy 进行连接 3259
管理 RDS 代理 3262
修改 RDS 代理 3262

添加数据库用户 3268
RDS 代理连接注意事项 3269
避免固定 RDS 代理 3273
删除 RDS 代理 3277
使用 RDS Proxy 终端节点 3277
代理终端节点概述 3278
代理端点的限制 3279
将读取器终端节点与 Aurora 集群结合使用 3279
访问 VPC 中的 Aurora 数据库 3283
创建代理终端节点 3284
查看代理终端节点 3287
修改代理终端节点 3288
删除代理终端节点 3289
使用 CloudWatch 监控 RDS Proxy 3290
使用 RDS Proxy 事件 3295
RDS Proxy 事件 3296
RDS Proxy 示例 3298
RDS 代理故障排除 3301
验证代理的连接 3301
常见问题和解决方案 3303
RDS for MySQL 故障排除 3304
RDS for PostgreSQL 故障排除 3305
将 RDS Proxy 与 AWS CloudFormation 一起使用 3310
在 Aurora Global Database 中使用 RDS 代理 3310
对全局数据库使用 RDS 代理的限制 3311
RDS 代理端点如何与全局数据库配合使用 3311
零 ETL 集成 3313
优势 3315
重要概念 3315
限制 3316
一般限制 3316
Aurora MySQL 限制 3317
Aurora PostgreSQL 限制 3317
Amazon Redshift 限制 3317
Amazon SageMaker 智能湖仓限制 3317
限额 3318

支持的区域 3318
开始使用零 ETL 集成 3318
步骤 1：创建自定义数据库集群参数组 3319
步骤 2：选择或创建源数据库集群 3320
步骤 3：创建目标 Amazon Redshift 数据仓库 3320
使用 AWS SDK 设置集成（仅限 Aurora MySQL） 3322
后续步骤 3327
创建与 Amazon Redshift 的零 ETL 集成 3327
先决条件 3327
所需的权限 3328
创建零 ETL 集成 3331
加密集成 3334
后续步骤 3336
创建与 Amazon SageMaker 智能湖仓的零 ETL 集成 3336
先决条件 3336
所需的权限 3337
创建与 Amazon SageMaker 智能湖仓的零 ETL 集成 3339
加密集成 3344
后续步骤 3346
零 ETL 集成的数据筛选 3346
数据筛选条件的格式 3347
筛选条件逻辑 3349
筛选条件优先级 3349
Aurora MySQL 示例 3350
Aurora PostgreSQL 示例 3351
添加数据筛选条件 3351
移除数据筛选条件 3354
添加和查询数据 3354
在 Amazon Redshift 中创建目标数据库 3355
向源数据库集群中添加数据 3355
在 Amazon Redshift 中查询您的 Aurora 数据 3356
数据类型差异 3357
查看和监控零 ETL 集成 3363
查看集成 3364
使用系统表进行监控 3365
使用 EventBridge 监控 3366

修改零 ETL 集成 3366
删除零 ETL 集成 3367
零 ETL 集成问题排查 3368
我无法创建零 ETL 集成 3369
我的集成卡在 Syncing 状态 3369
我的表未复制到 Amazon Redshift 3369
我的一个或多个 Amazon Redshift 表需要重新同步 3370
使用 Aurora Serverless v2 3372
Aurora Serverless v2 使用案例 3372
转换预置工作负载 3374
Aurora Serverless v2 的优点 3374
Aurora Serverless v2 的工作原理 3375
概览 3376
集群配置 3377
容量 3377
扩缩 3379
高可用性 3381
存储 3382
配置参数 3382
Aurora Serverless v2 的要求和限制 3382
区域和版本可用性 3383
使用 Aurora Serverless v2 的集群必须已指定容量范围 3383
在 Aurora Serverless v2 中不支持某些预置功能 3384
Aurora Serverless v2 的某些方面与 Aurora Serverless v1 不同 3384
创建 Aurora Serverless v2 数据库集群 3384
设置 3385
创建 Aurora Serverless v2 数据库集群 3386
创建 Aurora Serverless v2 写入器 3390
管理 Aurora Serverless v2 3391
为集群设置 Aurora Serverless v2 容量范围 3392
检查 Aurora Serverless v2 容量范围 3396
添加 Aurora Serverless v2 读取器 3397
从预置转换为 Aurora Serverless v2 3398
从 Aurora Serverless v2 转换为预置 3399
为 Aurora Serverless v2 读取器选择提升层 3400
将 TLS/SSL 与 Aurora Serverless v2 结合使用 3401

查看 Aurora Serverless v2 写入器和读取器 3403
Aurora Serverless v2 的日志记录 3404
Aurora Serverless v2 的性能和扩缩 3407
选择容量范围 3408
使用 Aurora Serverless v2 的参数组 3420
避免内存不足错误 3425
重要的 CloudWatch 指标 3426
使用性能详情监控 Aurora Serverless v2 性能 3429
Aurora Serverless v2 容量问题疑难解答 3430
缩减到零 ACU 从而自动暂停和恢复 3432
自动暂停概述 3432
先决条件和限制 3433
开启和关闭自动暂停 3434
自动暂停功能的工作原理 3437
自动暂停和 Aurora 集群配置 3440
监控自动暂停 3443
自动暂停故障排查 3446
自动暂停的应用程序设计 3447
迁移到 Aurora Serverless v2 3448
将 Aurora Serverless v2 与现有集群配合使用 3449
从预置集群切换 3451
比较 Aurora Serverless v2 和 Aurora Serverless v1 3457
从 Aurora Serverless v1 升级到 Aurora Serverless v2 3465
从本地数据库迁移到 Aurora Serverless v2 3468
使用 Aurora Serverless v1 3469
Aurora Serverless v1 的区域和版本可用性 3470
Aurora Serverless v1 的优点 3470
Aurora Serverless v1 的使用案例 3470
Aurora Serverless v1 的限制 3471
Aurora Serverless v1 的配置要求 3473
将 TLS/SSL 与 Aurora Serverless v1 结合使用 3473
支持的用于 Aurora Serverless v1 数据库集群连接的密码套件 3476
Aurora Serverless v1 的工作原理 3476
Aurora Serverless v1 架构 3477
AutoScaling 3478
超时操作 3479

暂停和恢复 3481
确定 max_connections 3481
参数组 3484
日志系统 3486
维护 3489
失效转移 3490
快照 3491
创建 Aurora Serverless v1 数据库集群 3491
还原 Aurora Serverless v1 数据库集群 3494
修改 Aurora Serverless v1 数据库集群 3497
修改扩缩配置 3498
升级主要版本 3500
从 Aurora Serverless v1 转换为预置 3502
从 Aurora Serverless v1 数据库集群转换为预置集群时的注意事项 3504
手动扩展 Aurora Serverless v1 数据库集群容量 3505
查看 Aurora Serverless v1 数据库集群 3507
使用 CloudWatch 监控 Aurora Serverless v1 数据库集群 3510
删除 Aurora Serverless v1 数据库集群 3511
Aurora Serverless v1 和 Aurora 数据库引擎版本 3514
Aurora MySQL Serverless 3515
Aurora PostgreSQL Serverless 3515
自动次要版本升级 3515
使用 Amazon RDS 数据 API 3516
Amazon RDS 数据 API 的区域和版本可用性 3517
限制 3517
Aurora Serverless v2 与 Aurora Serverless v1 的数据 API 比较 3518
每秒最大请求数 3518
在现有数据库上启用或禁用 Amazon RDS 数据 API 3518
CloudTrail 事件 3519
多语句支持 3519
对于同一事务 ID 的并发请求 3520
BatchExecuteStatement 行为 3521
ExecuteSQL 行为 3521
ExecuteStatement 行为 3522
Schema 参数行为 3522
身份验证和授权 3523

基于标签的授权 3524
将凭证存储在 Secrets Manager 密钥中 3526
启用数据 API 3527
在创建数据库时启用 RDS 数据 API 3527
在现有数据库上启用 RDS 数据 API 3528
创建 Amazon VPC 端点（PrivateLink） 3531
调用数据 API 3534
操作参考 3534
从 AWS CLI 调用 3539
从 Python 应用程序调用 3549
从 Java 应用程序调用 3553
控制超时行为 3557
Java 客户端库 3559
下载适用于 Data API 的 Java 客户端库 3559
Java 客户端库示例 3559
处理 JSON 格式的查询结果 3561
检索 JSON 格式的查询结果 3562
数据类型映射 3562
故障排除 3563
示例 3563
数据 API 故障排除 3567
未找到事务 <transaction_ID> 3568
用于查询的包太大 3568
数据库响应超出大小限制 3568
没有为集群 <cluster_ID> 启用 HttpEndpoint 3569
DatabaseErrorException：事务仍在运行查询 3569
不支持的结果异常 3569
不支持多语句 3570
不支持架构参数 3570
记录数据 API 调用 3570
在 CloudTrail 中使用数据 API 信息 3570
从 CloudTrail 跟踪记录中包含和排除数据 API 事件 3571
了解数据 API 日志文件条目 3573
监控数据 API 查询 3575
如何在性能详情中表示 RDS 数据 API 查询 3575
使用 查询编辑器 3577

查询编辑器的可用性 3577
授予访问权限 3577
运行查询 3579
DBQMS API 参考 3583
CreateFavoriteQuery 3584
CreateQueryHistory 3584
CreateTab 3584
DeleteFavoriteQueries 3584
DeleteQueryHistory 3584
DeleteTab 3584
DescribeFavoriteQueries 3584
DescribeQueryHistory 3584
DescribeTabs 3584
GetQueryString 3584
UpdateFavoriteQuery 3585
UpdateQueryHistory 3585
UpdateTab 3585
使用 Aurora 机器学习 3586
将 Aurora 机器学习与 Aurora MySQL 结合使用 3587
有关使用 Aurora 机器学习的要求 3588
区域和版本可用性 3589
支持的功能和限制 3589
设置 Aurora 集群以使用 Aurora 机器学习 3590
将 Amazon Bedrock 与 Aurora MySQL 数据库集群结合使用 3603
将 Amazon Comprehend 与 Aurora MySQL 数据库集群结合使用 3606
将 SageMaker AI 与 Aurora MySQL 数据库集群结合使用 3608
性能注意事项 3611
监控 3612
将 Aurora 机器学习与 Aurora PostgreSQL 结合使用 3613
有关使用 Aurora 机器学习的要求 3614
支持的功能和限制 3615
设置 Aurora 数据库集群以使用 Aurora 机器学习 3615
将 Amazon Bedrock 与 Aurora PostgreSQL 数据库集群结合使用 3628
将 Amazon Comprehend 与 Aurora PostgreSQL 数据库集群结合使用 3630
将 SageMaker AI 与 Aurora PostgreSQL 数据库集群结合使用 3632
将数据导出到 Amazon S3 以进行 SageMaker AI 模型训练（高级） 3636

性能注意事项 3636
监控 3641
代码示例 3642
操作 3653
CreateDBCluster 3653
CreateDBClusterParameterGroup 3672
CreateDBClusterSnapshot 3682
CreateDBInstance 3700
DeleteDBCluster 3718
DeleteDBClusterParameterGroup 3732
DeleteDBInstance 3747
DescribeDBClusterParameterGroups 3762
DescribeDBClusterParameters 3768
DescribeDBClusterSnapshots 3780
DescribeDBClusters 3787
DescribeDBEngineVersions 3806
DescribeDBInstances 3817
DescribeOrderableDBInstanceOptions 3832
ModifyDBClusterParameterGroup 3843
场景 3853
开始使用数据库集群 3853
跨服务示例 4023
创建借阅图书馆 REST API 4023
创建 Aurora Serverless 工作项跟踪器 4024
Aurora 的最佳实践 4029
Amazon Aurora 的基本操作指导方针 4029
数据库实例 RAM 建议 4030
AWS 数据库驱动程序 4031
监控 Amazon Aurora 4031
使用数据库参数组和数据库集群参数组 4031
Amazon Aurora 最佳实践视频 4031
执行 Aurora 概念验证 4032
Aurora 概念验证概述 4032
1. 确定目标 4032
2. 了解工作负载特性 4033
3. 使用控制台或 CLI 进行练习 4034

使用控制台进行练习 4034
使用 AWS CLI进行练习 4035
4. 创建 Aurora 集群 4035
5. 设置架构 4036
6. 导入数据 4037
7. 移植 SQL 代码 4037
8. 指定配置设置 4038
9. 连接到 Aurora 4039
10. 运行工作负载 4040
11. 衡量性能 4040
12. 练习使用 Aurora 高可用性 4042
13. 后续操作 4044
安全性 4046
Database authentication 4048
密码验证 4049
IAM 数据库身份验证 4049
Kerberos 身份验证 4049
使用 Aurora 和 Secrets Manager 管理密码 4051
区域和版本可用性 4051
限制 4051
概览 4051
优势 4052
权限 4053
强制实施 Aurora 管理 4053
管理数据库集群的密码 4054
轮换数据库集群的密钥 4058
查看数据库集群的密钥详细信息 4060
数据保护 4063
数据加密 4064
互联网络流量隐私 4090
身份和访问管理 4091
受众 4091
使用身份进行身份验证 4092
使用策略管理访问 4094
Amazon Aurora 如何与 IAM 协同工作 4096
基于身份的策略示例 4102

AWS 托管式策略 4122
策略更新 4128
防止跨服务混淆代理 4136
IAM 数据库身份验证 4138
问题排查 4185
日志记录和监控 4187
合规性验证 4189
故障恢复能力 4190
备份与还原 4190
复制 4190
故障转移 4190
基础设施安全性 4192
安全组 4192
公开可用性 4192
VPC 终端节点 (AWS PrivateLink) 4194
注意事项 943
可用性 943
创建接口 VPC 终端节点 943
创建 VPC 终端节点策略 943
安全最佳实践 4197
使用安全组控制访问权限 4198
VPC 安全组概述 4198
安全组情况 4199
创建 VPC 安全组 4200
与数据库集群关联 4201
主用户账户权限 4201
服务相关角色 4203
Amazon Aurora 的服务相关角色权限 4203
Amazon RDS 测试版的服务相关角色权限 4206
Amazon RDS 预览版的服务相关角色 4207
配合使用 Amazon Aurora 和 Amazon VPC 4208
在 VPC 中使用数据库集群 4208
在 VPC 中访问数据库集群的场景 4222
教程：创建 VPC 以用于数据库集群（仅限 IPv4） 4228
教程：创建 VPC 以用于数据库集群（双堆栈模式） 4235
配额和限制 4244

Amazon Aurora 中的配额 4244
Amazon Aurora 中的命名约束 4248
Amazon Aurora 大小限制 4249
故障排除 4250
无法连接到 数据库实例 4250
测试数据库实例连接 4252
排查连接身份验证错误 4253
安全性问题 4253
错误消息“无法检索账户属性，某些控制台功能可能受损。” 4253
重置数据库实例拥有者密码 4253
数据库实例中断或重新引导 4254
参数更改未生效 4254
Aurora 可用内存问题 4255
Aurora MySQL 复制问题 4255
诊断并解决只读副本之间的滞后 4256
诊断并解决 MySQL 读取复制故障 4257
复制已停止错误 4259
只读副本复制未能初始化元数据结构 4259
Amazon RDS API 参考 4260
使用查询 API 4260
查询参数 4260
查询请求身份验证 4260
对应用程序进行问题排查 4261
检索错误 4261
故障排除技巧 4261
文档历史记录 4263
AWS 术语表 4324

什么是 Amazon Aurora？

Amazon Aurora (Aurora) 是一个与 MySQL 和 PostgreSQL 兼容的完全托管的关系数据库引擎。您已了解了 MySQL 和 PostgreSQL 不仅具有高端商用数据库的速度和可靠性，同时还具有开源数据库的简单性和成本效益。您目前用于现有 MySQL 和 PostgreSQL 数据库的代码、工具和应用程序可用于 Aurora。在某些工作负载条件下，Aurora 最多可以将 MySQL 吞吐量增加 5 倍，将 PostgreSQL 的吞吐量增加 3 倍，而无需对大多数现有应用程序进行更改。

Aurora 包括一个高性能的存储子系统。已自定义其 MySQL 和 PostgreSQL 兼容数据库引擎以利用该快速分布式存储。基础存储会根据需要自动增长。Aurora 集群卷可增大到最大大小 128 tebibytes (TiB)。Aurora 还会自动执行和标准化数据库集群和复制，这通常是数据库配置和管理方面的最大问题。

Aurora 是托管式数据库服务 Amazon Relational Database Service（Amazon RDS）的一部分。Amazon RDS 让用户能够在云中更轻松地设置、操作和扩展关系数据库。如果您还不熟悉 Amazon RDS，请参阅 Amazon Relational Database Service 用户指南。要详细了解 Amazon Web Services 上提供的各种数据库选项，请参阅为 AWS 上的组织选择合适的数据库。

主题

- Amazon RDS 责任共担模式
- Amazon Aurora 如何与 Amazon RDS 协同工作
- Amazon Aurora 数据库集群
- Amazon Aurora 版本
- 区域及可用区
- Amazon Aurora 中受 AWS 区域 和 Aurora 数据库引擎支持的功能
- Amazon Aurora 端点连接
- Amazon Aurora 数据库实例类
- Amazon Aurora 存储
- Amazon Aurora 可靠性
- Amazon Aurora 安全性
- Amazon Aurora 的高可用性
- 使用 Amazon Aurora 进行复制
- Aurora 的数据库实例计费

Amazon RDS 责任共担模式

Amazon RDS 负责托管数据库实例和数据库集群的软件组件和基础设施。您负责查询优化，这是调整 SQL 查询以提高性能的过程。查询性能高度依赖于数据库设计、数据大小、数据分布、应用程序工作负载和查询模式，这些可能会有很大差异。监控和优化是您自己的 RDS 数据库所拥有的高度个性化流程。您可以使用 Amazon RDS 性能详情和其他工具来识别有问题的查询。

Amazon Aurora 如何与 Amazon RDS 协同工作

以下几点说明了 Amazon Aurora 如何与 Amazon RDS 中提供的标准 MySQL 和 PostgreSQL 引擎相关：

- 在通过 Amazon RDS 设置新的数据库服务器时，您选择 Aurora MySQL 或 Aurora PostgreSQL 作为数据库引擎选项。
- Aurora 利用熟悉的 Amazon Relational Database Service（Amazon RDS）功能进行管理。Aurora 使用 Amazon RDS AWS Management Console 接口、AWS CLI 命令和 API 操作来处理日常数据库任务，如预置、修补、备份、恢复、故障检测和修复。
- Aurora 管理操作通常涉及通过复制同步的整个数据库服务器集群，而不是单个数据库实例。自动集群、复制和存储分配以简单且经济高效的方式设置、运行和扩展非常大的 MySQL 和 PostgreSQL 部署。
- 您可以创建和还原快照或设置单向复制，以将数据从 Amazon RDS for MySQL 和 Amazon RDS for PostgreSQL 复制到 Aurora。您可以使用按钮迁移工具将现有的 RDS for MySQL 和 RDS for PostgreSQL 应用程序转换为 Aurora。

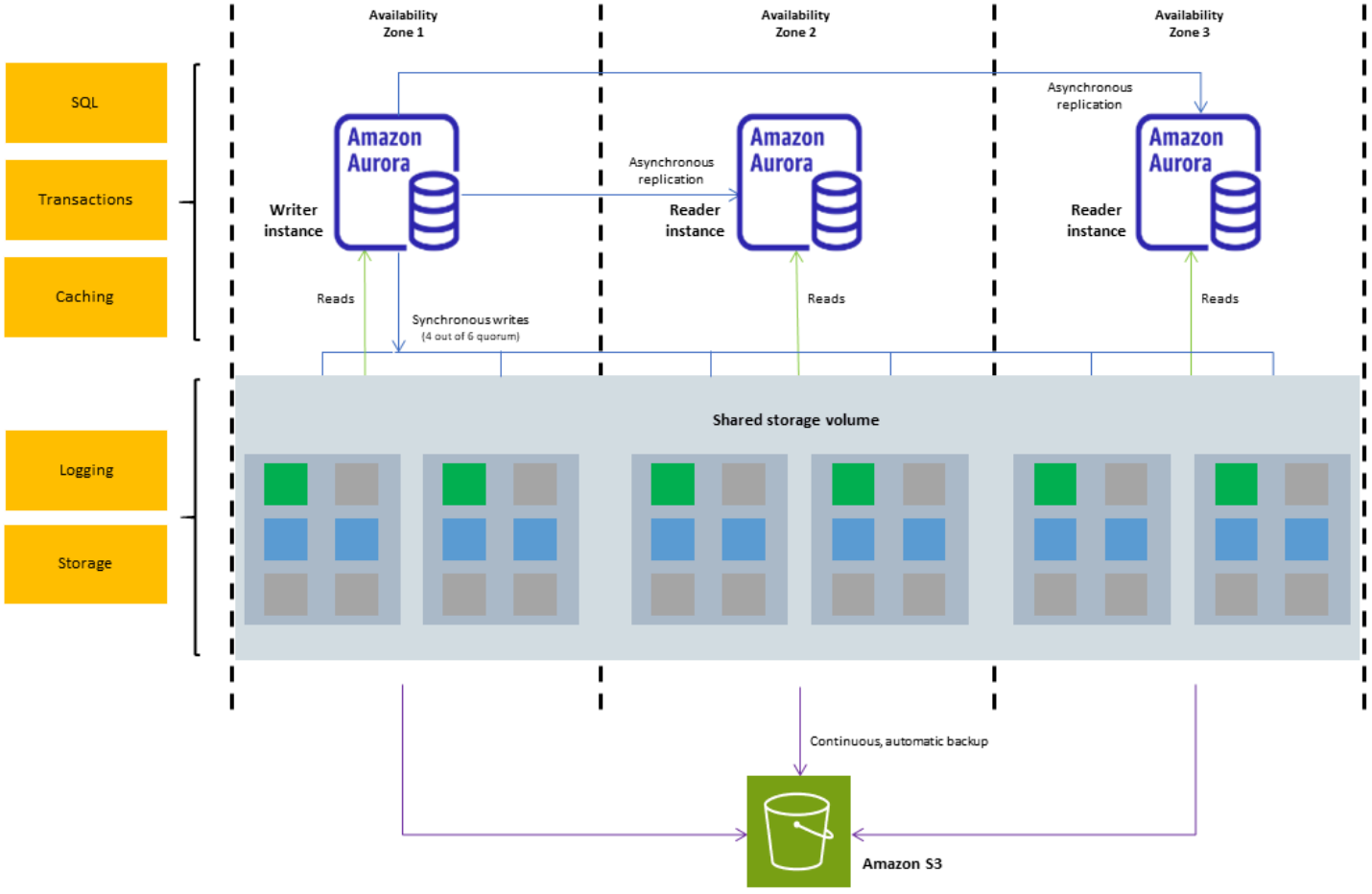
在使用 Amazon Aurora 之前，先完成为 Amazon Aurora 设置环境中的步骤，然后查看 Amazon Aurora 数据库集群中的 Aurora 概念和特性。

Amazon Aurora 数据库集群

Amazon Aurora 数据库集群包含一个或多个数据库实例以及一个管理这些数据库实例的数据的集群卷。Aurora 集群卷 是一个跨多个可用区的虚拟数据库存储卷，每个可用区具有一个数据库集群数据副本。Aurora 数据库集群由两类数据库实例组成：

- 主（写入器）数据库实例 – 支持读取和写入操作，并执行针对集群卷的所有数据修改。每个 Aurora 数据库集群均有一个主数据库实例。
- Aurora 副本（读取器数据库实例）– 连接到同一存储卷作为主数据库实例，但仅支持读取操作。除主数据库实例之外，每个 Aurora 数据库集群最多可拥有 15 个 Aurora 副本。通过将 Aurora 副本放在单独的可用区中维护高可用性。当主数据库实例不可用时，Aurora 自动故障转移到 Aurora 副本。您可以为 Aurora 副本指定故障转移优先级。Aurora 副本还可以从主数据库实例分载读取工作负载。

下图说明了 Aurora 数据库集群中的集群卷、写入器数据库实例和读取器数据库实例之间的关系。



> **Note**
>
> 上述信息适用于所有 Aurora 数据库集群：预置、并行查询、Aurora Global Database、Aurora Serverless、Aurora MySQL 兼容和 Aurora PostgreSQL 兼容。

此 Aurora 数据库集群说明了计算容量和存储的分离。例如，仅具有单个数据库实例的 Aurora 配置仍是集群，因为基础存储卷涉及跨多个可用区 (AZ) 分布的多个存储节点。

Aurora 数据库集群中的输入/输出（I/O）操作以相同的方式计数，无论它们是在写入器还是读取器数据库实例上。有关更多信息，请参阅 Amazon Aurora 数据库集群的存储配置。

Amazon Aurora 版本

借助 Amazon Aurora，您可以选择最适合您的应用程序要求的支持的关系数据库引擎，同时保持与底层引擎的兼容性。Aurora 重复使用来自支持的引擎的数据库引擎代码，因此您可以利用这些引擎的现有技能、工具和库。创建集群时，您需要指定要使用的 Amazon Aurora 数据库引擎版本。您选择的版本决定了兼容性和可用功能。

本文档解释了 Amazon Aurora 和相应数据库引擎之间的共同点和区别。利用这些信息，您可以确定应选择的软件版本以及如何验证每个版本中的可用功能和错误修复。您也可以使用此参考来确定适当的升级节奏和升级计划。

Amazon Aurora 数据库集群支持的数据库引擎

以下关系数据库在 Amazon Aurora 上可用：Aurora 重复使用代码并保持与底层数据库引擎的兼容性。但是，Aurora 具有独特的版本号、发布周期和版本弃用时间表。每个 Aurora 新版本都附带发行说明，其中列出了该版本的新功能、修复以及其他更改和增强功能。

Aurora 数据库	用户指南	可用版本	发行说明
Amazon Aurora MySQL 兼容版	使用 Amazon Aurora MySQL	Amazon Aurora MySQL 的数据库引擎更新	Aurora MySQL 发布说明
Amazon Aurora PostgreSQL 兼容版	使用 Amazon Aurora PostgreSQL	Amazon Aurora PostgreSQL 的数据库引擎更新	Aurora PostgreSQL 发布说明

为您的数据库集群指定 Amazon Aurora 数据库版本

当您在 AWS Management Console或 AWS CLI 中使用创建数据库操作或者使用 `CreateDBCluster` API 操作创建新的数据库集群时，可以指定任意当前可用的 Aurora 数据库版本（主要或次要版本）。

要了解如何创建 Aurora 集群，请参阅创建 Amazon Aurora 数据库集群。要了解如何更改现有 Aurora 集群的版本，请参阅修改 Amazon Aurora 数据库集群。

> **Note**
>
> 并非每个 Aurora 数据库版本在所有 AWS 区域区域都可用。要了解区域和每个 AWS 区域中可用版本的更多信息，请参阅区域及可用区和支持 Aurora 全球数据库的区域和数据库引擎。

Amazon Aurora 版本控制

Amazon Aurora 版本不同于它们所兼容的上游社群数据库。为了帮助您维护应用程序兼容性并利用最新的数据库引擎功能，以下各节介绍了 Aurora 版本控制约定以及 Aurora 版本如何映射到其各自的社群数据库。

有关 Amazon Aurora 上可用的关系数据库的列表，请参阅 Amazon Aurora 数据库集群支持的数据库引擎。

社群数据库和 Aurora 之间的版本号差异

每个 Amazon Aurora 版本都与其相应社群数据库的某个特定版本兼容。您可以使用 `version` 函数查找您的数据库的社群版本，使用 `aurora_version` 函数查找 Aurora 版本。

以下示例演示了如何查找 Aurora MySQL 和 Aurora PostgreSQL 数据库的社群版本。

Aurora MySQL

`version` 函数返回 Aurora MySQL 数据库的社群版本。

```
mysql> select version();
```

输出示例：

```
+------------------+
|   version()      |
+------------------+
|  8.0.32          |
+------------------+
```

`aurora_version` 函数返回 Aurora 版本：

```
mysql> select aurora_version(), @@aurora_version;
```

输出示例：

```
+------------------+------------------+
| aurora_version() | @@aurora_version |
+------------------+------------------+
| 3.05.2           | 3.05.2           |
+------------------+------------------+
```

Aurora PostgreSQL

`version` 函数返回 Aurora PostgreSQL 数据库的社群版本。

```
postgres=> select version();
```

输出示例：

```
-----------------------------------------------------------------------------
PostgreSQL 11.7 on x86_64-pc-linux-gnu, compiled by gcc (GCC) 4.9.3, 64-bit
(1 row)
```

`aurora_version` 函数返回 Aurora 版本：

```
postgres=> select aurora_version();
```

输出示例：

```
aurora_version
----------------
3.2.2
```

有关更多信息，请参阅使用 SQL 检查 Aurora MySQL 版本和确定 Amazon Aurora PostgreSQL 版本。

Amazon Aurora 默认版本

默认版本是 Aurora 在您未手动指定目标引擎版本时自动选择的用于创建或升级数据库的版本。例如，以下命令显示了 Aurora PostgreSQL 的默认引擎版本（包括示例输出）。

```
aws rds describe-db-engine-versions \
    --engine aurora-postgresql \
    --default-only \
    --query 'DBEngineVersions[0].EngineVersion' \
    --output text
```

```
16.4
```

每个主要版本都有相应的默认次要版本。因此，对于 Aurora PostgreSQL 16，默认次要版本是 16.n，当 Aurora 发布新的默认次要版本时，版本号 n 会发生变化。通常，Aurora 每年会为每个主要版本发布两个默认次要版本。以下 bash Shell 脚本显示了一组 Aurora PostgreSQL 主要版本的默认次要版本（包括示例输出）。

```
for major in 16 15 14 13 12 11; do
  echo -n "Default for Aurora PostgreSQL major version $major: "
  aws rds describe-db-engine-versions \
    --engine aurora-postgresql \
    --engine-version "$major" \
    --default-only \
    --query 'DBEngineVersions[0].EngineVersion' \
    --output text
done

Default for Aurora PostgreSQL major version 16: 16.4
Default for Aurora PostgreSQL major version 15: 15.8
Default for Aurora PostgreSQL major version 14: 14.13
Default for Aurora PostgreSQL major version 13: 13.16
Default for Aurora PostgreSQL major version 12: 12.20
Default for Aurora PostgreSQL major version 11: 11.21
```

如果您为 Aurora 数据库集群启用自动次要版本升级，Aurora 会为给定的主要版本使用默认次要版本或较新的次要版本。例如，如果 Aurora PostgreSQL 15 的默认次要版本为 15.8，并且较新的版本 15.10 也可用，那么 Aurora 可以自动升级到 15.8 或 15.10。

Amazon Aurora 主要版本

Aurora 版本使用 *major*.*minor*.*patch* 方案。Aurora 主要版本是指 Aurora 兼容的社群版 MySQL 或 PostgreSQL 的主要版本。在标准支持下，Aurora MySQL 和 Aurora PostgreSQL 主要版本至少会在相应社群版本的社区生命周期终止前保持可用状态。您可以继续运行过了 Aurora 标准支持结束日期的主要版本，但需要付费。有关更多信息，请参阅Amazon Aurora 的 Amazon RDS 扩展支持和 Amazon Aurora 定价。

有关 Aurora MySQL 和 Aurora PostgreSQL 主要版本和发布日历的更多信息，请参阅相应发行说明中的下列页面：

- Aurora MySQL 主要版本的发布日历

- Aurora PostgreSQL 主要版本的发布日历

您还可以运行 describe-db-major-engine-versions AWS CLI 命令或者使用 DescribeDBMajorEngineVersions RDS API 操作，来查看有关主要引擎版本的支持日期的信息。

> **Note**
>
> Amazon RDS 对 Aurora MySQL 版本 2 的扩展支持将于 2024 年 11 月 1 日开始，但要等到 2024 年 12 月 1 日才会向您收费。在 2024 年 11 月 1 日至 11 月 30 日期间，所有 Aurora MySQL 版本 2 的数据库集群都包含在 Amazon RDS 扩展支持中。有关更多信息，请参阅 选定 Aurora 版本的 Amazon RDS 扩展支持。

Amazon Aurora 主要版本可用时间

Amazon Aurora 主要版本至少会在相应社群版本的社群终止使用前保持可用状态。您可以使用 Aurora 标准支持终止日期来计划您的测试和升级周期。这些日期表示可能需要升级到较新版本的最早日期。有关日期的更多信息，请参阅Amazon Aurora 主要版本。

在 Aurora 要求您升级到较新的主要版本并协助您进行规划之前，您至少会提前 12 个月收到提醒。提醒提供以下有关升级过程的信息。

- 某些里程碑的时机
- 对数据库集群的影响
- 建议的操作

我们建议您在将集群升级到新的主要版本之前，使用新的数据库版本彻底测试您的应用程序。

在主要版本到达 Aurora 标准支持终止日期后，任何仍在运行较早版本的数据库集群都会在计划的维护时段内自动升级到扩展支持版本。可能收取 Extended Support 费用。有关 Amazon RDS 扩展支持的更多信息，请参阅使用 Amazon RDS 扩展支持。

Amazon Aurora 次要版本

Aurora 版本使用 *major*.*minor*.*patch* 方案。Aurora 次要版本包含了由社群提供的增量更新，以及特定于 Aurora 的服务改进，例如新功能和修复。

有关 Aurora MySQL 和 Aurora PostgreSQL 次要版本和发布日历的更多信息，请参阅相应发行说明中的下列页面：

- Aurora MySQL 次要版本的发布日历
- Aurora PostgreSQL 次要版本的发布日历

以下各节详细介绍了您可以预计的 Aurora 次要版本的节奏和生命周期。

主题

- Amazon Aurora 次要版本发布频率
- Amazon Aurora 次要版本可用时间

Amazon Aurora 次要版本发布频率

一般来说，Amazon Aurora 次要版本每季度发布一次。发布时间可能会有所变化，以加入额外功能或修复。

Amazon Aurora 次要版本可用时间

通常，Amazon Aurora 会为特定主要版本的每个次要版本提供至少 12 个月的可用期。可用期结束后，Aurora 可能会自动将数据库升级到默认次要版本或更高版本。对于正在运行较早次要版本的任何数据库集群，Aurora 会在计划的维护时段内开始升级。

在某些情况下，Aurora 可能会早于通常的 12 个月期限替换特定主要版本的次要版本。原因可能包括关键的安全问题或主要版本的支持终止日期。

在开始对生命周期即将结束的次要版本进行自动升级之前，Aurora 通常会提前三个月发出提醒。Aurora 详细介绍了以下有关升级过程的信息。

- 某些里程碑的时机
- 对数据库集群的影响
- 建议的操作

告知时间少于三个月的通知描述了需要更快采取行动的严重问题，如安全问题。

如果启用了自动次要版本升级设置，您将收到提醒，但不会收到 RDS 事件通知。Aurora 会在强制升级截止日期过后的维护时段内升级您的数据库。

如果未启用自动次要版本升级设置，则您会收到提醒以及类别为 `maintenance`、ID 为 `RDS-EVENT-0156` 的 Amazon RDS 数据库集群事件。Aurora 将在下一个维护时段内升级您的数据库。

请注意，在次要版本达到 Aurora 标准支持终止日期后，将不再发布该次要版本的补丁版本。要获得严重错误修复或 CVE，您必须升级到具有标准支持的次要版本。

有关自动次要版本升级的更多信息，请参阅 Aurora 数据库集群的自动次要版本升级。

Amazon Aurora 补丁版本

Aurora 版本使用 *major*.*minor*.*patch* 方案。Aurora 补丁版本包括在次要版本初始发布（例如 Aurora MySQL 3.04.0、3.04.1、...3.04.3）后添加到其中的重要修复。虽然每个新次要版本都会为 Aurora 提供新功能，但特定次要版本中的新补丁版本主要用于解决重要问题。

有关补丁的更多信息，请参阅维护 Amazon Aurora 数据库集群。

升级 Amazon Aurora 数据库集群

借助 Amazon Aurora，您可以控制和测试数据库集群升级。Amazon Aurora 提供自动次要版本升级、手动升级控制、必需的升级和升级前测试选项。您可以使用最新的次要版本使集群保持最新状态，以推迟非关键升级，针对严重问题强制升级，以及在非生产环境中验证升级行为。以下各节详细介绍了如何使用这些功能管理和测试 Aurora 数据库集群升级。

Aurora 的自动次要版本升级

自动次要版本升级会定期将数据库更新到新近的数据库引擎版本。但是，升级可能并不始终包括最新的数据库引擎版本。如果您需要在特定时间将数据库保持在特定版本上，我们建议您根据所需的时间表，手动升级到您需要的数据库版本。在出现严重安全问题的情况下或当版本达到其终止支持日期时，Amazon Aurora 可能会应用次要版本升级，即使您尚未启用自动次要版本升级选项也是如此。有关更多信息，请参阅特定数据库引擎的升级文档。

请参阅 升级 Aurora MySQL 数据库集群的次要版本或补丁程序级别和执行次要版本升级。

您可以为 Aurora 集群中的每个数据库实例启用自动次要版本升级，以保持 Aurora 次要版本处于最新状态。仅当集群中的所有数据库实例均启用了该设置时，Aurora 才会进行自动升级。

如果数据库集群的自动次要版本升级为是，则 Aurora 会自动升级到默认次要版本或较新的次要版本。例如，如果 Aurora PostgreSQL 15 的默认次要版本为 15.8，并且存在版本 15.10，则自动升级的目标可能是 15.8 或 15.10。

通常情况下，对于启用了自动次要版本升级的数据库集群，Aurora 每年会安排两次自动升级。自动升级将在您为集群指定的维护时段内进行。有关更多信息，请参阅 Aurora 数据库集群的自动次要版本升级。

事先通过 Amazon RDS 数据库集群事件（类别为 `maintenance`，ID 为 `RDS-EVENT-0156`）与自动次要版本升级进行通信。有关更多信息，请参阅 Aurora 的 Amazon RDS 事件类别和事件消息。

手动控制将数据库集群升级到新版本

如果您启用了自动次要版本升级设置，Aurora 会自动将您的数据库集群升级到默认次要版本或较新的次要版本。通常情况下，对于启用了自动次要版本升级设置的数据库集群，Aurora 每年会安排两次自动升级。自动升级会在客户指定的维护时段内启动。

要关闭自动次要版本升级，请在 Aurora 集群中禁用任意数据库实例的自动次要版本升级。只有在集群中的所有数据库实例均启用了该设置时，Aurora 才会执行自动次要版本升级。

> **Note**
>
> 对于强制升级（例如次要版本生命周期终止），即使禁用了自动次要版本升级设置，Aurora 也会升级数据库集群。您会收到提醒，但没有 RDS 事件通知。Aurora 对集群的升级将在强制升级截止日期过后的维护时段内进行。

因为主要版本升级会涉及到某些兼容性风险，因此主要版本升级不会自动发生。必须启动这些升级，除非是在主要版本弃用的情况下。我们建议您在将集群升级到主要版本之前，使用新的数据库版本彻底测试您的应用程序。

有关将数据库集群升级到新 Aurora 主要版本的更多信息，请参阅升级 Amazon Aurora MySQL 数据库集群和升级 Amazon Aurora PostgreSQL 数据库集群。

必需的 Amazon Aurora 升级

对于某些关键修复，Aurora 可能执行托管式升级，以升级到同一次要版本中较新的补丁级别。在这种情况下，即使关闭了自动次要版本升级，Aurora 也会升级您的集群。在执行升级前，Aurora 会告知详细的升级过程。详细信息包括某些里程碑的时间安排、对数据库集群的影响以及建议的操作。此类托管式升级会在集群维护时段内自动进行。

在升级前测试 Aurora 新版本中的数据库集群

您可以测试升级过程，以及新版本与您的应用程序和工作负载的配合情况。使用以下方法之一：

- 使用 Amazon Aurora 快速数据库克隆功能，克隆您的集群。在新集群上进行升级以及任何升级后的测试。

- 恢复集群快照，以创建新的 Aurora 集群。您可以自行为现有 Aurora 集群创建集群快照。Aurora 也会为您的每个集群定期自动创建快照。然后，您可以为新集群启动版本升级。在决定是否升级原集群之前，您可以在升级后的集群副本上进行试验。

有关创建测试用新集群的详细方法，请参阅克隆 Amazon Aurora 数据库集群卷和创建数据库集群快照。

Amazon Aurora 版本支持

如果您的 Amazon Aurora 数据库对特定的数据库引擎行为有复杂的依赖性，建议您在升级到更新的数据库引擎版本之前进行大量测试。我们提供有长期支持选项，因此，即使在选定的 Aurora 引擎版本被更新版本取代之后，您仍然可以在选定的版本上维护数据库集群。以下各节介绍了针对 Aurora 数据库集群的长期支持选项。

对所选 Amazon Aurora 次要版本的长期支持

每个 Aurora 主要版本中都有特定的次要版本被指定为长期支持（LTS）版本，支持期至少三年。也就是说，每个主要版本中至少有一个次要版本的可用时间超过通常的 12 个月。我们通常会在此期限结束前六个月发出提醒。Aurora 提供了以下有关升级过程的信息。

- 某些里程碑的时机
- 对数据库集群的影响
- 建议的操作

告知时间不到六个月的通知用于传达需要更快采取行动的严重问题（例如安全问题）。

LTS 次要版本仅包含关键修复（通过补丁版本）。LTS 版本不包含在该版本推出之后发布的新功能。每年一次，在 LTS 次要版本上运行的数据库集群都会被修补为该 LTS 版本的最新补丁版本。Aurora 会修补您的集群，以便您从累积的安全性和稳定性修复中受益。如果有关键的修复程序，Aurora 可能会更频繁地修补需要应用的 LTS 次要版本，例如出于安全考虑。

> **Note**
>
> 如果您希望在 LTS 次要版本的生命周期内保持在该版本中，请确保禁用数据库实例的自动次要版本升级功能。要避免数据库集群从 LTS 次要版本自动进行升级，请对于 Aurora 集群中的任何数据库实例清除允许自动次要版本升级复选框。

有关所有 Aurora LTS 版本的版本号，请参阅 Aurora MySQL 长期支持 (LTS) 版本 和 使用 Aurora PostgreSQL 长期支持（LTS）版本。

选定 Aurora 版本的 Amazon RDS 扩展支持

借助 Amazon RDS 扩展支持，您可以在过了 Aurora 标准支持结束日期的主要引擎版本上继续运行您的数据库，但需要额外付费。在 RDS Extended Support 期间，Amazon RDS 将根据美国国家漏洞数据库（NVD）的 CVSS 严重性评级的定义为严重和高 CVE 提供补丁。有关更多信息，请参阅 Amazon Aurora 的 Amazon RDS 扩展支持。

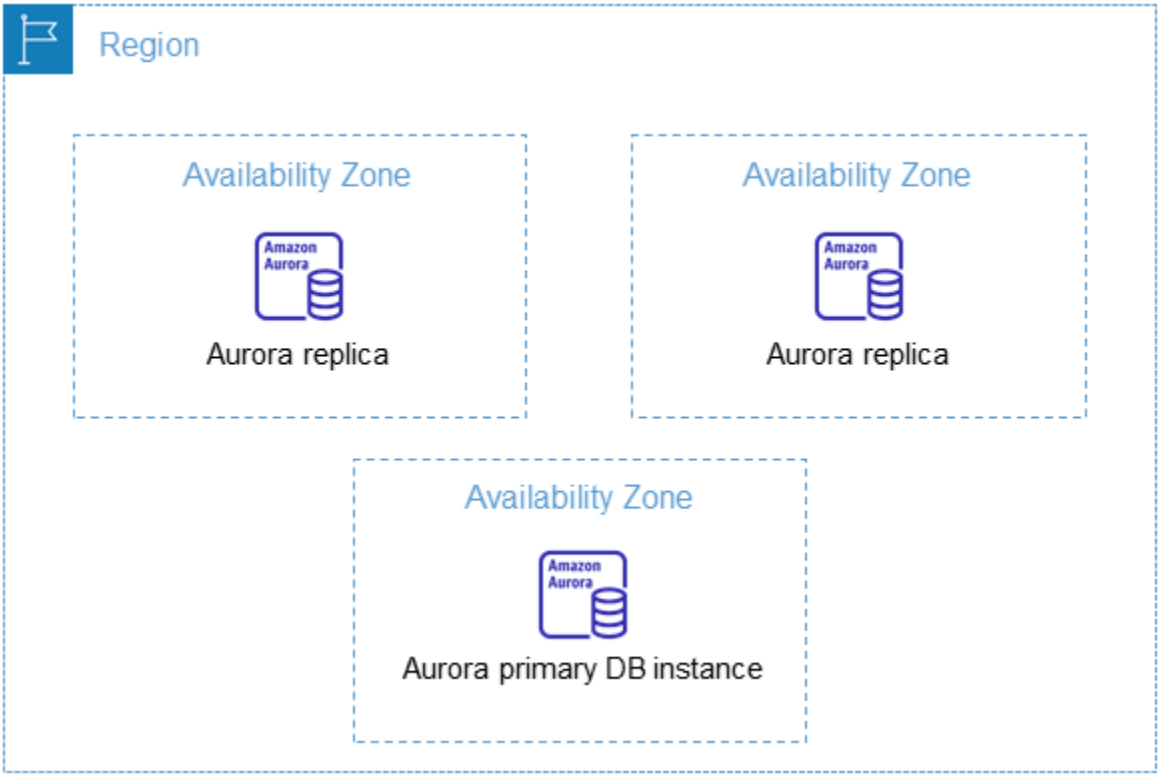
RDS Extended Support 仅适用于某些 Aurora 版本。有关更多信息，请参阅 Amazon Aurora 主要版本。

区域及可用区

Amazon 云计算资源在全球多个位置托管。这些位置由 AWS 区域和可用区构成。每个 AWS 区域 都是一个单独的地理区域。每个 AWS 区域都有多个相互隔离的位置，称为可用区。

> **Note**
>
> 有关查找 AWS 区域的可用区的信息，请参阅 Amazon EC2 文档中的描述可用区。

Amazon 运行着具有高可用性的先进数据中心。数据中心有时会发生影响托管于同一位置的所有数据库实例的可用性的故障，虽然这种故障极少发生。如果您将所有数据库实例都托管在受此类故障影响的同一个位置，则您的所有数据库实例都将不可用。



需要牢记的是，每个 AWS 区域都是完全独立的，这一点非常重要。您启动的任何 Amazon RDS 活动（如创建数据库实例或列出可用的数据库实例）都只会在您当前的默认AWS区域运行。可以在控制台中或通过设置 `AWS_DEFAULT_REGION` 环境变量更改原定设置 AWS 区域。或者，可以通过 AWS Command Line Interface（AWS CLI）使用 `--region` 参数来覆盖它。有关更多信息，请参阅配置 AWS Command Line Interface，特别是有关环境变量和命令行选项的章节。

Amazon RDS 支持名为 AWS GovCloud (US) 的特殊 AWS 区域。此类区域旨在使美国政府机构和客户可以将较为敏感的工作负载移至云中。AWS GovCloud (US) 区域满足美国政府的特定法规和合规性要求。有关更多信息，请参阅什么是 AWS GovCloud (US)？

要在特定AWS区域中创建或使用 Amazon RDS 数据库实例，请使用相应的区域服务端点。

> **Note**
>
> Aurora 不支持 Local Zones。

AWS 地区

从设计而言，每个 AWS 区域都与其他 AWS 区域隔离。此设计可实现最大程度的容错能力和稳定性。

当您查看资源时，只会看到与您指定的 AWS 区域关联的资源。这是因为 AWS 区域间彼此隔离，而且我们不会自动跨 AWS 区域复制资源。

区域可用性

使用命令行界面或 API 操作处理 Aurora 数据库集群时，请确保您指定了其区域端点。

主题

- Aurora MySQL 区域可用性
- Aurora PostgreSQL 区域可用性

Aurora MySQL 区域可用性

下表显示了当前可以使用 Aurora MySQL 的AWS区域以及每个区域的端点。

区域名称	区域	端点	协议	
美国东部（俄亥俄州）	us-east-2	rds.us-east-2.amazonaws.com	HTTPS	
美国东部（弗吉尼	us-east-1	rds.us-east-1.amazonaws.com	HTTPS	

区域名称	区域	端点	协议	
亚州北部）				
美国西部（加利福尼亚北部）	us-west-1	rds.us-west-1.amazonaws.com	HTTPS	
美国西部（俄勒冈州）	us-west-2	rds.us-west-2.amazonaws.com	HTTPS	
非洲（开普敦）	af-south-1	rds.af-south-1.amazonaws.com	HTTPS	
亚太地区（香港）	ap-east-1	rds.ap-east-1.amazonaws.com	HTTPS	
亚太地区（海得拉巴）	ap-south-2	rds.ap-south-2.amazonaws.com	HTTPS	
亚太地区（雅加达）	ap-southeast-3	rds.ap-southeast-3.amazonaws.com	HTTPS	
亚太地区（墨尔本）	ap-southeast-4	rds.ap-southeast-4.amazonaws.com	HTTPS	
亚太地区（孟买）	ap-south-1	rds.ap-south-1.amazonaws.com	HTTPS	
亚太地区（大阪）	ap-northeast-3	rds.ap-northeast-3.amazonaws.com	HTTPS	

区域名称	区域	端点	协议	
亚太地区（首尔）	ap-northeast-2	rds.ap-northeast-2.amazonaws.com	HTTPS	
亚太地区（新加坡）	ap-southeast-1	rds.ap-southeast-1.amazonaws.com	HTTPS	
亚太地区（悉尼）	ap-southeast-2	rds.ap-southeast-2.amazonaws.com	HTTPS	
亚太地区（东京)	ap-northeast-1	rds.ap-northeast-1.amazonaws.com	HTTPS	
加拿大（中部）	ca-central-1	rds.ca-central-1.amazonaws.com	HTTPS	
加拿大西部（卡尔加里）	ca-west-1	rds.ca-west-1.amazonaws.com	HTTPS	
欧洲地区（法兰克福）	eu-central-1	rds.eu-central-1.amazonaws.com	HTTPS	
欧洲地区（爱尔兰）	eu-west-1	rds.eu-west-1.amazonaws.com	HTTPS	
欧洲地区（伦敦）	eu-west-2	rds.eu-west-2.amazonaws.com	HTTPS	
欧洲地区（米兰）	eu-south-1	rds.eu-south-1.amazonaws.com	HTTPS	
欧洲地区（巴黎）	eu-west-3	rds.eu-west-3.amazonaws.com	HTTPS	

区域名称	区域	端点	协议	
欧洲（西班牙）	eu-south-2	rds.eu-south-2.amazonaws.com	HTTPS	
欧洲地区（斯德哥尔摩）	eu-north-1	rds.eu-north-1.amazonaws.com	HTTPS	
欧洲（苏黎世）	eu-central-2	rds.eu-central-2.amazonaws.com	HTTPS	
以色列（特拉维夫）	il-central-1	rds.il-central-1.amazonaws.com	HTTPS	
中东（巴林）	me-south-1	rds.me-south-1.amazonaws.com	HTTPS	
中东（阿联酋）	me-central-1	rds.me-central-1.amazonaws.com	HTTPS	
南美洲（圣保罗）	sa-east-1	rds.sa-east-1.amazonaws.com	HTTPS	
AWS GovCloud（美国东部）	us-gov-east-1	rds.us-gov-east-1.amazonaws.com	HTTPS	
AWS GovCloud（美国西部）	us-gov-west-1	rds.us-gov-west-1.amazonaws.com	HTTPS	

Aurora PostgreSQL 区域可用性

下表显示了当前可以使用 Aurora PostgreSQL 的AWS区域以及每个区域的端点。

区域名称	区域	端点	协议	
美国东部（俄亥俄州）	us-east-2	rds.us-east-2.amazonaws.com	HTTPS	
美国东部（弗吉尼亚州北部）	us-east-1	rds.us-east-1.amazonaws.com	HTTPS	
美国西部（加利福尼亚北部）	us-west-1	rds.us-west-1.amazonaws.com	HTTPS	
美国西部（俄勒冈州）	us-west-2	rds.us-west-2.amazonaws.com	HTTPS	
非洲（开普敦）	af-south-1	rds.af-south-1.amazonaws.com	HTTPS	
亚太地区（香港）	ap-east-1	rds.ap-east-1.amazonaws.com	HTTPS	
亚太地区（海得拉巴）	ap-south-2	rds.ap-south-2.amazonaws.com	HTTPS	
亚太地区（雅加达）	ap-southeast-3	rds.ap-southeast-3.amazonaws.com	HTTPS	
亚太地区（墨尔本）	ap-southeast-4	rds.ap-southeast-4.amazonaws.com	HTTPS	

区域名称	区域	端点	协议	
亚太地区（孟买）	ap-south-1	rds.ap-south-1.amazonaws.com	HTTPS	
亚太地区（大阪）	ap-northeast-3	rds.ap-northeast-3.amazonaws.com	HTTPS	
亚太地区（首尔）	ap-northeast-2	rds.ap-northeast-2.amazonaws.com	HTTPS	
亚太地区（新加坡）	ap-southeast-1	rds.ap-southeast-1.amazonaws.com	HTTPS	
亚太地区（悉尼）	ap-southeast-2	rds.ap-southeast-2.amazonaws.com	HTTPS	
亚太地区（东京)	ap-northeast-1	rds.ap-northeast-1.amazonaws.com	HTTPS	
加拿大（中部）	ca-central-1	rds.ca-central-1.amazonaws.com	HTTPS	
加拿大西部（卡尔加里）	ca-west-1	rds.ca-west-1.amazonaws.com	HTTPS	
欧洲地区（法兰克福）	eu-central-1	rds.eu-central-1.amazonaws.com	HTTPS	
欧洲地区（爱尔兰）	eu-west-1	rds.eu-west-1.amazonaws.com	HTTPS	
欧洲地区（伦敦）	eu-west-2	rds.eu-west-2.amazonaws.com	HTTPS	

区域名称	区域	端点	协议	
欧洲地区（米兰）	eu-south-1	rds.eu-south-1.amazonaws.com	HTTPS	
欧洲地区（巴黎）	eu-west-3	rds.eu-west-3.amazonaws.com	HTTPS	
欧洲（西班牙）	eu-south-2	rds.eu-south-2.amazonaws.com	HTTPS	
欧洲地区（斯德哥尔摩）	eu-north-1	rds.eu-north-1.amazonaws.com	HTTPS	
欧洲（苏黎世）	eu-central-2	rds.eu-central-2.amazonaws.com	HTTPS	
以色列（特拉维夫）	il-central-1	rds.il-central-1.amazonaws.com	HTTPS	
中东（巴林）	me-south-1	rds.me-south-1.amazonaws.com	HTTPS	
中东（阿联酋）	me-central-1	rds.me-central-1.amazonaws.com	HTTPS	
南美洲（圣保罗）	sa-east-1	rds.sa-east-1.amazonaws.com	HTTPS	
AWS GovCloud（美国东部）	us-gov-east-1	rds.us-gov-east-1.amazonaws.com	HTTPS	

区域名称	区域	端点	协议	
AWS GovCloud（美国西部）	us-gov-west-1	rds.us-gov-west-1.amazonaws.com	HTTPS	

可用区

可用区是指定 AWS 区域中的隔离位置。每个区域都有多个可用区 (AZ)，旨在为该区域提供高可用性。可用区由 AWS 区域代码后跟字母标识符（例如，`us-east-1a`）标识。如果您创建 VPC 和子网而不是使用默认 VPC，则可以在特定可用区中定义每个子网。当您创建 Aurora 数据库集群时，Aurora 会在 VPC 的数据库子网组的其中一个子网中创建主实例。这样，它就可以将该实例与 Aurora 选择的特定可用区相关联。

每个 Aurora 数据库集群都将其存储副本托管在三个独立的可用区中，这些可用区由 Aurora 自动从数据库子网组中的可用区中选择。集群中的每个数据库实例必须位于这三个可用区之一。

在集群中创建数据库实例时，如果未指定可用区，Aurora 会自动为该实例选择适当的可用区。

使用如下的 describe-availability-zones Amazon EC2 命令描述在指定区域内已为您的账户启用的可用区。

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

例如，要描述在美国东部（弗吉尼亚州北部）区域（us-east-1）为您的账户启用的可用区域，请运行以下命令：

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

要了解如何在创建集群或向集群添加实例时指定可用区，请参阅 为数据库集群配置网络。

Amazon Aurora 数据库集群的本地时区

默认情况下，Amazon Aurora 数据库群集的时区是协调世界时 (UTC)。您可以改为将数据库群集中实例的时区设置为您的应用程序的本地时区。

要为数据库集群设置本地时区，请将时区参数设置为受支持的值之一。您可以在数据库集群的集群参数组中设置此参数。

- 对于 Aurora MySQL，此参数的名称是 `time_zone`。有关设置 `time_zone` 参数的最佳实践的信息，请参阅优化时间戳操作。
- 对于 Aurora PostgreSQL，此参数的名称是 `timezone`。

在为数据库集群设置时区参数时，数据库集群中的所有实例都将改为使用新的本地时区。在某些情况下，其他 Aurora 数据库集群可能正在使用相同的集群参数组。如果是这样，则这些数据库集群中的所有实例也将改为使用新的本地时区。有关群集级参数的信息，请参阅Amazon Aurora 数据库集群和数据库实例参数。

设置本地时区之后，所有新数据库连接都会反映更改。在某些情况下，在更改本地时区时，您可能有打开的数据库连接。如果是这样，则直至关闭连接再打开新连接之后才会看到本地时区更新。

如果要跨 AWS 区域进行复制，则复制源数据库集群和副本使用不同的参数组。参数组对于 AWS 区域是唯一的。要对每个实例使用相同的本地时区，请确保同时在复制源和副本的参数组中设置时区参数。

从数据库集群快照还原数据库集群时，本地时区设置为 UTC。还原完成之后，可以将时区更新为本地时区。在某些情况下，您可能将数据库集群还原到某个时间点。如果是这样，还原的数据库集群的本地时区是来自还原的数据库集群的参数组的时区设置。

下表列出了一些可以设置本地时区的值。要列出所有可用时区，可以使用以下 SQL 查询：

- Aurora MySQL：`select * from mysql.time_zone_name;`
- Aurora PostgreSQL：`select * from pg_timezone_names;`

> **Note**
>
> 对于一些时区，可能会错误报告某些日期范围的时间值，如表中所述。对于澳大利亚时区，返回的时区缩写为过时的值，如表中所述。

时区	备注
`Africa/Harare`	此时区设置回返回 28 Feb 1903 21:49:40 GMT 和 28 Feb 1903 21:55:48 GMT 之间的错误值。
`Africa/Monrovia`	

时区	备注
Africa/Nairobi	此时区设置会返回 31 Dec 1939 21:30:00 GMT 和 31 Dec 1959 21:15:15 GMT 之间的错误值。
Africa/Windhoek	
America/Bogota	此时区设置会返回 23 Nov 1914 04:56:16 GMT 和 23 Nov 1914 04:56:20 GMT 之间的错误值。
America/Caracas	
America/Chihuahua	
America/Cuiaba	
America/Denver	
America/Fortaleza	在某些情况下，针对南美洲（圣保罗）区域中的数据库集群，时间对于最近更改的巴西时区显示不正确。如果是这样，请将数据库集群的时区参数重置为 America/Fortaleza 。
America/Guatemala	
America/Halifax	此时区设置会返回 27 Oct 1918 05:00:00 GMT 和 31 Oct 1918 05:00:00 GMT 之间的错误值。
America/Manaus	如果数据库集群位于南美洲（库亚巴）区域，并且最近更改的巴西时区未按预期正确显示时间，请将数据库集群的时区参数重置为 America/Manaus 。
America/Matamoros	
America/Monterrey	

时区	备注
America/Montevideo	
America/Phoenix	
America/Tijuana	
Asia/Ashgabat	
Asia/Baghdad	
Asia/Baku	
Asia/Bangkok	
Asia/Beirut	
Asia/Calcutta	
Asia/Kabul	
Asia/Karachi	
Asia/Kathmandu	
Asia/Muscat	此时区设置会返回 31 Dec 1919 20:05:36 GMT 和 31 Dec 1919 20:05:40 GMT 之间的错误值。
Asia/Riyadh	此时区设置会返回 13 Mar 1947 20:53:08 GMT 和 31 Dec 1949 20:53:08 GMT 之间的错误值。
Asia/Seoul	此时区设置会返回 30 Nov 1904 15:30:00 GMT 和 07 Sep 1945 15:00:00 GMT 之间的错误值。
Asia/Shanghai	此时区设置会返回 31 Dec 1927 15:54:08 GMT 和 02 Jun 1940 16:00:00 GMT 之间的错误值。
Asia/Singapore	

时区	备注
Asia/Taipei	此时区设置会返回 30 Sep 1937 16:00:00 GMT 和 29 Sep 1979 15:00:00 GMT 之间的错误值。
Asia/Tehran	
Asia/Tokyo	此时区设置会返回 30 Sep 1937 15:00:00 GMT 和 31 Dec 1937 15:00:00 GMT 之间的错误值。
Asia/Ulaanbaatar	
Atlantic/Azores	此时区设置会返回 24 May 1911 01:54:32 GMT 和 01 Jan 1912 01:54:32 GMT 之间的错误值。
Australia/Adelaide	此时区的缩写以 CST 而非 ACDT/ACST 形式返回。
Australia/Brisbane	此时区的缩写以 EST 而非 AEDT/AEST 形式返回。
Australia/Darwin	此时区的缩写以 CST 而非 ACDT/ACST 形式返回。
Australia/Hobart	此时区的缩写以 EST 而非 AEDT/AEST 形式返回。
Australia/Perth	此时区的缩写以 WST 而非 AWDT/AWST 形式返回。
Australia/Sydney	此时区的缩写以 EST 而非 AEDT/AEST 形式返回。
Brazil/East	
Canada/Saskatchewan	此时区设置会返回 27 Oct 1918 08:00:00 GMT 和 31 Oct 1918 08:00:00 GMT 之间的错误值。
Europe/Amsterdam	

时区	备注
Europe/Athens	
Europe/Dublin	
Europe/Helsinki	此时区设置会返回 30 Apr 1921 22:20:08 GMT 和 30 Apr 1921 22:20:11 GMT 之间的错误值。
Europe/Paris	
Europe/Prague	
Europe/Sarajevo	
Pacific/Auckland	
Pacific/Guam	
Pacific/Honolulu	此时区设置会返回 21 May 1933 11:30:00 GMT 和 30 Sep 1945 11:30:00 GMT 之间的错误值。
Pacific/Samoa	此时区设置会返回 01 Jan 1911 11:22:48 GMT 和 01 Jan 1950 11:30:00 GMT 之间的错误值。
US/Alaska	
US/Central	
US/Eastern	
US/East-Indiana	
US/Pacific	
UTC	

Amazon Aurora 中受 AWS 区域 和 Aurora 数据库引擎支持的功能

Aurora 与 MySQL 和 PostgreSQL 兼容的数据库引擎支持多个 Amazon Aurora 和 Amazon RDS 功能和选项。支持因每个数据库引擎的特定版本以及 AWS 区域而异。要确定针对给定 AWS 区域中某个功能的 Aurora 数据库引擎版本支持和可用性，可以使用以下各个部分。

其中一些功能是只 Aurora 功能。例如，Amazon RDS 不支持 Aurora Serverless、Aurora Global Database 以及对于与AWS机器学习服务集成的支持。还有一些 Amazon Aurora 和 Amazon RDS 都支持的功能，例如 Amazon RDS Proxy。

支持的区域和数据库引擎

- 表约定
- 支持蓝绿部署的区域和 Aurora 数据库引擎
- 支持集群存储配置的区域和 Aurora 数据库引擎
- 支持数据库活动流的区域和 Aurora 数据库引擎
- 支持将集群数据导出到 Amazon S3 的区域和 Aurora 数据库引擎
- 支持将快照数据导出到 Amazon S3 的区域和 Aurora 数据库引擎
- 支持 Aurora 全球数据库的区域和数据库引擎
- 支持 IAM 数据库身份验证的区域和 Aurora 数据库引擎
- 支持 Kerberos 身份验证的区域和 Aurora 数据库引擎
- 支持 Aurora 机器学习的区域和数据库引擎
- 支持 Performance Insights 的区域和 Aurora 数据库引擎
- 支持与 Amazon Redshift 进行零 ETL 集成的区域和 Amazon 数据库引擎
- 支持 Amazon RDS 代理的区域和 Aurora 数据库引擎
- 支持 Secrets Manager 集成的区域和 Aurora 数据库引擎
- 支持 Aurora Serverless v2 的区域和 Aurora 数据库引擎
- Aurora Serverless v1
- 支持 RDS 数据 API 的区域和 Aurora 数据库引擎
- 支持零停机修补（ZDP）的区域和 Aurora 数据库引擎
- 支持 Aurora 引擎原生功能的区域和数据库引擎

表约定

这些功能部分中的表使用以下模式来指定版本号和支持级别：

- x.y 版 – 仅支持特定版本。
- 版本 x.y 及更高版本 – 支持指定的版本及其主要版本的所有更高次要版本。例如，“10.11 版及更高版本”表示支持 10.11、10.11.1 和 10.12 版本。

支持蓝绿部署的区域和 Aurora 数据库引擎

蓝绿部署将生产数据库环境复制到单独的同步暂存环境中。通过使用 Amazon RDS 蓝绿部署，您可以在不影响生产环境的情况下对暂存环境中的数据库进行更改。例如，您可以升级主要或次要数据库引擎版本、更改数据库参数或在暂存环境中更改模式。准备就绪后，可以将暂存环境升级为新的生产数据库环境。有关更多信息，请参阅 使用 Amazon Aurora 蓝绿部署进行数据库更新。

Aurora MySQL 的蓝绿部署

蓝绿部署功能适用于所有 AWS 区域中所有版本的 Aurora MySQL。

Aurora PostgreSQL 的蓝绿部署

以下区域和引擎版本可用于 Aurora PostgreSQL 的蓝绿部署。

区域	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
所有 AWS 区域	版本 16.1 及更高版本	版本 15.4 及更高版本	版本 14.9 及更高版本	版本 13.12 及更高版本	版本 12.16 及更高版本	版本 11.21 及更高版本

支持集群存储配置的区域和 Aurora 数据库引擎

Amazon Aurora 有两种数据库集群存储配置：Aurora I/O-Optimized 和 Aurora Standard。有关更多信息，请参阅 Amazon Aurora 数据库集群的存储配置。

Aurora I/O-Optimized

Aurora I/O-Optimized 在所有 AWS 区域中可用于以下 Amazon Aurora 版本：

- Aurora MySQL 版本 3.03.1 及更高版本
- Aurora PostgreSQL 16.1 及更高版本、15.2 及更高版本、14.7 及更高版本以及 13.10 及更高版本

Aurora Standard

Aurora Standard 在所有 AWS 区域中适用于所有 Aurora MySQL 和 Aurora PostgreSQL 版本。

支持数据库活动流的区域和 Aurora 数据库引擎

利用 Aurora 中的数据库活动流，您可以在 Aurora 数据库中监控审计活动并为其设置告警。有关更多信息，请参阅 使用数据库活动流监控 Amazon Aurora。

以下功能不支持数据库活动流：

- Aurora Serverless v1
- Aurora Serverless v2
- 适用于 Aurora PostgreSQL 的 Babelfish

主题

- 适用于 Aurora MySQL 的数据库活动流
- 适用于 Aurora PostgreSQL 的数据库活动流

适用于 Aurora MySQL 的数据库活动流

以下区域和引擎可用性可用于面向 Aurora MySQL 的数据库活动流。

区域	Aurora MySQL 版本 3	Aurora MySQL 版本 2
美国东部（俄亥俄州）	所有可用版本	Aurora 版本 2.11 及更高版本
美国东部（弗吉尼亚州北部）	所有可用版本	Aurora 版本 2.11 及更高版本
美国西部（加利福尼亚北部）	所有可用版本	Aurora 版本 2.11 及更高版本
美国西部（俄勒冈州）	所有可用版本	Aurora 版本 2.11 及更高版本
非洲（开普敦）	所有可用版本	Aurora 版本 2.11 及更高版本

区域	Aurora MySQL 版本 3	Aurora MySQL 版本 2
亚太地区（香港）	所有可用版本	Aurora 版本 2.11 及更高版本
亚太地区（海得拉巴）	所有可用版本	Aurora 版本 2.11 及更高版本
亚太地区（雅加达）	所有可用版本	Aurora 版本 2.11 及更高版本
亚太地区（孟买）	所有可用版本	Aurora 版本 2.11 及更高版本
亚太地区（大阪）	所有可用版本	Aurora 版本 2.11 及更高版本
亚太地区（首尔）	所有可用版本	Aurora 版本 2.11 及更高版本
亚太地区（新加坡）	所有可用版本	Aurora 版本 2.11 及更高版本
亚太地区（悉尼）	所有可用版本	Aurora 版本 2.11 及更高版本
Asia Pacific (Tokyo)	所有可用版本	Aurora 版本 2.11 及更高版本
加拿大（中部）	所有可用版本	Aurora 版本 2.11 及更高版本
加拿大西部（卡尔加里）	不可用	不可用
中国（北京）	不可用	不可用
中国（宁夏）	不可用	不可用
欧洲地区（法兰克福）	所有可用版本	Aurora 版本 2.11 及更高版本
欧洲地区（爱尔兰）	所有可用版本	Aurora 版本 2.11 及更高版本
欧洲地区（伦敦）	所有可用版本	Aurora 版本 2.11 及更高版本
欧洲地区（米兰）	所有可用版本	Aurora 版本 2.11 及更高版本
欧洲地区（巴黎）	所有可用版本	Aurora 版本 2.11 及更高版本
欧洲（西班牙）	所有可用版本	Aurora 版本 2.11 及更高版本
欧洲地区（斯德哥尔摩）	所有可用版本	Aurora 版本 2.11 及更高版本

区域	Aurora MySQL 版本 3	Aurora MySQL 版本 2
欧洲（苏黎世）	不可用	不可用
以色列（特拉维夫）	不可用	不可用
中东（巴林）	所有可用版本	Aurora 版本 2.11 及更高版本
中东（阿联酋）	所有可用版本	Aurora 版本 2.11 及更高版本
南美洲（圣保罗）	所有可用版本	Aurora 版本 2.11 及更高版本

适用于 Aurora PostgreSQL 的数据库活动流

以下区域和引擎可用性可用于面向 Aurora PostgreSQL 的数据库活动流。

区域	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
美国东部（俄亥俄州）	版本 16.1 及更高版本	版本 15.2 及更高版本	所有可用版本	所有可用版本	所有可用版本	版本 11.9 和版本 11.13 及更高版本
美国东部（弗吉尼亚州北部）	版本 16.1 及更高版本	版本 15.2 及更高版本	所有可用版本	所有可用版本	所有可用版本	版本 11.9 和版本 11.13 及更高版本
美国西部（加利福尼亚北部）	版本 16.1 及更高版本	版本 15.2 及更高版本	所有可用版本	所有可用版本	所有可用版本	版本 11.9 和版本 11.13 及更高版本
美国西部（俄勒冈州）	版本 16.1 及更高版本	版本 15.2 及更高版本	所有可用版本	所有可用版本	所有可用版本	版本 11.9 和版本

区域	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
						11.13 及更高版本
非洲（开普敦）	版本 16.1 及更高版本	版本 15.2 及更高版本	所有可用版本	所有可用版本	所有可用版本	版本 11.9 和版本 11.13 及更高版本
亚太地区（香港）	版本 16.1 及更高版本	版本 15.2 及更高版本	所有可用版本	所有可用版本	所有可用版本	版本 11.9 和版本 11.13 及更高版本
亚太地区（海得拉巴）	版本 16.1 及更高版本	版本 15.2 及更高版本	所有可用版本	所有可用版本	所有可用版本	版本 11.9 和版本 11.13 及更高版本
亚太地区（雅加达）	版本 16.1 及更高版本	版本 15.2 及更高版本	所有可用版本	所有可用版本	所有可用版本	版本 11.9 和版本 11.13 及更高版本
亚太地区（墨尔本）	不可用	不可用	不可用	不可用	不可用	不可用
亚太地区（孟买）	版本 16.1 及更高版本	版本 15.2 及更高版本	所有可用版本	所有可用版本	所有可用版本	版本 11.9 和版本 11.13 及更高版本
亚太地区（大阪）	版本 16.1 及更高版本	版本 15.2 及更高版本	所有可用版本	所有可用版本	所有可用版本	版本 11.9 和版本 11.13 及更高版本

区域	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
亚太地区（首尔）	版本 16.1 及更高版本	版本 15.2 及更高版本	所有可用版本	所有可用版本	所有可用版本	版本 11.9 和版本 11.13 及更高版本
亚太地区（新加坡）	版本 16.1 及更高版本	版本 15.2 及更高版本	所有可用版本	所有可用版本	所有可用版本	版本 11.9 和版本 11.13 及更高版本
亚太地区（悉尼）	版本 16.1 及更高版本	版本 15.2 及更高版本	所有可用版本	所有可用版本	所有可用版本	版本 11.9 和版本 11.13 及更高版本
Asia Pacific (Tokyo)	版本 16.1 及更高版本	版本 15.2 及更高版本	所有可用版本	所有可用版本	所有可用版本	版本 11.9 和版本 11.13 及更高版本
加拿大（中部）	版本 16.1 及更高版本	版本 15.2 及更高版本	所有可用版本	所有可用版本	所有可用版本	版本 11.9 和版本 11.13 及更高版本
加拿大西部（卡尔加里）	不可用	不可用	不可用	不可用	不可用	不可用
中国（北京）	不可用	不可用	不可用	不可用	不可用	不可用
中国（宁夏）	不可用	不可用	不可用	不可用	不可用	不可用

区域	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
欧洲地区（法兰克福）	版本 16.1 及更高版本	版本 15.2 及更高版本	所有可用版本	所有可用版本	所有可用版本	版本 11.9 和版本 11.13 及更高版本
欧洲地区（爱尔兰）	版本 16.1 及更高版本	版本 15.2 及更高版本	所有可用版本	所有可用版本	所有可用版本	版本 11.9 和版本 11.13 及更高版本
欧洲地区（伦敦）	版本 16.1 及更高版本	版本 15.2 及更高版本	所有可用版本	所有可用版本	所有可用版本	版本 11.9 和版本 11.13 及更高版本
欧洲地区（米兰）	版本 16.1 及更高版本	版本 15.2 及更高版本	所有可用版本	所有可用版本	所有可用版本	版本 11.9 和版本 11.13 及更高版本
欧洲地区（巴黎）	版本 16.1 及更高版本	版本 15.2 及更高版本	所有可用版本	所有可用版本	所有可用版本	版本 11.9 和版本 11.13 及更高版本
欧洲（西班牙）	版本 16.1 及更高版本	版本 15.2 及更高版本	所有可用版本	所有可用版本	所有可用版本	版本 11.9 和版本 11.13 及更高版本
欧洲地区（斯德哥尔摩）	版本 16.1 及更高版本	版本 15.2 及更高版本	所有可用版本	所有可用版本	所有可用版本	版本 11.9 和版本 11.13 及更高版本

区域	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
欧洲（苏黎世）	不可用	不可用	不可用	不可用	不可用	不可用
以色列（特拉维夫）	不可用	不可用	不可用	不可用	不可用	不可用
中东（巴林）	版本 16.1 及更高版本	版本 15.2 及更高版本	所有可用版本	所有可用版本	所有可用版本	版本 11.9 和版本 11.13 及更高版本
中东（阿联酋）	版本 16.1 及更高版本	版本 15.2 及更高版本	所有可用版本	所有可用版本	所有可用版本	版本 11.9 和版本 11.13 及更高版本
南美洲（圣保罗）	版本 16.1 及更高版本	版本 15.2 及更高版本	所有可用版本	所有可用版本	所有可用版本	版本 11.9 和版本 11.13 及更高版本

支持将集群数据导出到 Amazon S3 的区域和 Aurora 数据库引擎

您可以将 Aurora 数据库集群数据导出到 Amazon S3 桶。导出数据后，您可以通过 Amazon Athena 或 Amazon Redshift Spectrum 等工具直接分析导出的数据。有关更多信息，请参阅 将数据库集群数据导出到 Amazon S3。

将集群数据导出到 S3 在以下 AWS 区域 中可用：

- 美国东部（弗吉尼亚州北部）
- 美国东部（俄亥俄州）
- 美国西部（加利福尼亚北部）
- 美国西部（俄勒冈）

- 亚太地区（香港）
- 亚太地区（海得拉巴）
- 亚太地区（雅加达）
- 亚太地区（墨尔本）
- 亚太地区（孟买）
- 亚太地区（大阪）
- 亚太地区（首尔）
- 亚太地区（新加坡）
- 亚太地区（悉尼）
- 亚太地区（东京）
- 加拿大（中部）
- 加拿大西部（卡尔加里）
- 中国（宁夏）
- 欧洲地区（法兰克福）
- 欧洲地区（爱尔兰）
- 欧洲地区（伦敦）
- Europe (Paris)
- 欧洲（西班牙）
- 欧洲地区（斯德哥尔摩）
- 欧洲（苏黎世）
- 以色列（特拉维夫）
- 中东（阿联酋）
- 南美洲（圣保罗）

主题

- 使用 Aurora MySQL 将集群数据导出到 S3
- 使用 Aurora PostgreSQL 将集群数据导出到 S3

使用 Aurora MySQL 将集群数据导出到 S3

所有当前可用的 Aurora MySQL 引擎版本都支持将数据库集群数据导出到 Amazon S3。有关版本的更多信息，请参阅 Aurora MySQL 版本注释。

使用 Aurora PostgreSQL 将集群数据导出到 S3

所有当前可用的 Aurora PostgreSQL 引擎版本都支持将数据库集群数据导出到 Amazon S3。有关版本的更多信息，请参阅 Aurora PostgreSQL 版本注释。

支持将快照数据导出到 Amazon S3 的区域和 Aurora 数据库引擎

您可以将 Aurora 数据库集群快照数据导出到 Amazon S3 桶。您可以导出手动快照和自动系统快照。导出数据后，您可以通过 Amazon Athena 或 Amazon Redshift Spectrum 等工具直接分析导出的数据。有关更多信息，请参阅 将数据库集群快照数据导出到 Amazon S3。

在除以下区域外的所有 AWS 区域 中都支持将快照导出 S3：

- 亚太地区（马来西亚）
- AWS GovCloud（美国东部）
- AWS GovCloud（美国西部）

主题

- 使用 Aurora MySQL 将快照数据导出到 S3
- 使用 Aurora PostgreSQL 将快照数据导出到 S3

使用 Aurora MySQL 将快照数据导出到 S3

所有当前可用的 Aurora MySQL 引擎版本都支持将数据库集群快照数据导出到 Amazon S3。有关版本的更多信息，请参阅 Aurora MySQL 版本注释。

使用 Aurora PostgreSQL 将快照数据导出到 S3

所有当前可用的 Aurora PostgreSQL 引擎版本都支持将数据库集群快照数据导出到 Amazon S3。有关版本的更多信息，请参阅 Aurora PostgreSQL 版本注释。

支持 Aurora 全球数据库的区域和数据库引擎

Aurora Global Database 是跨多个 AWS 区域的单一数据库，能够在发生任何区域级的故障时提供低延迟的全局读取和灾难恢复能力。它为您的部署提供了内置的容错功能，因为数据库实例不依赖于单个 AWS 区域，而是依赖于多个区域和不同的可用区。有关更多信息，请参阅 使用 Amazon Aurora Global Database。

主题

- 使用 Aurora MySQL 的 Aurora 全局数据库
- 使用 Aurora PostgreSQL 的 Aurora 全局数据库

使用 Aurora MySQL 的 Aurora 全局数据库

以下区域和引擎版本可用于面向 Aurora MySQL 的 Aurora 全局数据库。

区域	Aurora MySQL 版本 3	Aurora MySQL 版本 2
美国东部（弗吉尼亚州北部）	版本 3.01.0 及更高版本	版本 2.07.0 及更高版本
美国东部（俄亥俄州）	版本 3.01.0 及更高版本	版本 2.07.0 及更高版本
美国西部（加利福尼亚北部）	版本 3.01.0 及更高版本	版本 2.07.0 及更高版本
美国西部（俄勒冈州）	版本 3.01.0 及更高版本	版本 2.07.0 及更高版本
非洲（开普敦）	版本 3.01.0 及更高版本	版本 2.07.1 及更高版本
亚太地区（香港）	版本 3.01.0 及更高版本	版本 2.07.1 及更高版本
亚太地区（海得拉巴）	版本 3.02.0 及更高版本	版本 2.11.2 及更高版本
亚太地区（雅加达）	版本 3.01.0 及更高版本	版本 2.07.6 及更高版本
亚太地区（马来西亚）	版本 3.04.0 及更高版本	版本 2.07.6 及更高版本
亚太地区（墨尔本）	版本 3.03.0 及更高版本	版本 2.07.6 及更高版本
亚太地区（孟买）	版本 3.01.0 及更高版本	版本 2.07.0 及更高版本
亚太地区（大阪）	版本 3.01.0 及更高版本	版本 2.07.3 及更高版本

区域	Aurora MySQL 版本 3	Aurora MySQL 版本 2
亚太地区（首尔）	版本 3.01.0 及更高版本	版本 2.07.0 及更高版本
亚太地区（新加坡）	版本 3.01.0 及更高版本	版本 2.07.0 及更高版本
亚太地区（悉尼）	版本 3.01.0 及更高版本	版本 2.07.0 及更高版本
亚太地区（东京）	版本 3.01.0 及更高版本	版本 2.07.0 及更高版本
加拿大（中部）	版本 3.01.0 及更高版本	版本 2.07.0 及更高版本
加拿大西部（卡尔加里）	版本 3.01.0 及更高版本	版本 2.07.0 及更高版本
中国（北京）	版本 3.01.0 及更高版本	版本 2.07.2 及更高版本
中国（宁夏）	版本 3.01.0 及更高版本	版本 2.07.2 及更高版本
欧洲地区（法兰克福）	版本 3.01.0 及更高版本	版本 2.07.0 及更高版本
欧洲地区（爱尔兰）	版本 3.01.0 及更高版本	版本 2.07.0 及更高版本
欧洲地区（伦敦）	版本 3.01.0 及更高版本	版本 2.07.0 及更高版本
欧洲地区（米兰）	版本 3.01.0 及更高版本	版本 2.07.1 及更高版本
欧洲地区（巴黎）	版本 3.01.0 及更高版本	版本 2.07.0 及更高版本
欧洲地区（西班牙）	版本 3.02.0 及更高版本	版本 2.07.6 及更高版本
欧洲地区（斯德哥尔摩）	版本 3.01.0 及更高版本	版本 2.07.0 及更高版本
欧洲（苏黎世）	版本 3.02.0 及更高版本	版本 2.07.6 及更高版本
以色列（特拉维夫）	版本 3.02.0 及更高版本	版本 2.07.6 及更高版本
中东（巴林）	版本 3.01.0 及更高版本	版本 2.07.1 及更高版本
中东（阿联酋）	版本 3.02.0 及更高版本	版本 2.07.6 及更高版本
南美洲（圣保罗）	版本 3.01.0 及更高版本	版本 2.07.1 及更高版本

区域	Aurora MySQL 版本 3	Aurora MySQL 版本 2
AWS GovCloud（美国东部）	版本 3.01.0 及更高版本	版本 2.07.0 及更高版本
AWS GovCloud（美国西部）	版本 3.01.0 及更高版本	版本 2.07.0 及更高版本

使用 Aurora PostgreSQL 的 Aurora 全局数据库

以下区域和引擎版本可用于面向 Aurora PostgreSQL 的 Aurora 全局数据库。

区域	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
美国东部（弗吉尼亚州北部）	版本 17.4 及更高版本	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本
美国东部（俄亥俄州）	版本 17.4 及更高版本	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本
美国西部（加利福尼亚北部）	版本 17.4 及更高版本	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本
美国西部（俄勒冈州）	版本 17.4 及更高版本	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本
非洲（开普敦）	版本 17.4 及更高版本	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本

区域	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
							11.13 及更高版本
亚太地区（香港）	版本 17.4 及更高版本	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本
亚太地区（海得拉巴）	版本 17.4 及更高版本	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本
亚太地区（雅加达）	版本 17.4 及更高版本	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本
亚太地区（马来西亚）	版本 17.4 及更高版本	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本
亚太地区（墨尔本）	版本 17.4 及更高版本	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本
亚太地区（孟买）	版本 17.4 及更高版本	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本

区域	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
亚太地区（大阪）	版本 17.4 及更高版本	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本
亚太地区（首尔）	版本 17.4 及更高版本	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本
亚太地区（新加坡）	版本 17.4 及更高版本	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本
亚太地区（悉尼）	版本 17.4 及更高版本	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本
亚太地区（东京）	版本 17.4 及更高版本	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本
加拿大（中部）	版本 17.4 及更高版本	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本
加拿大西部（卡尔加里）	版本 17.4 及更高版本	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本

区域	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
中国（北京）	版本 17.4 及更高版本	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本
中国（宁夏）	版本 17.4 及更高版本	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本
欧洲地区（法兰克福）	版本 17.4 及更高版本	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本
欧洲地区（爱尔兰）	版本 17.4 及更高版本	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本
欧洲地区（伦敦）	版本 17.4 及更高版本	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本
欧洲地区（米兰）	版本 17.4 及更高版本	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本
欧洲地区（巴黎）	版本 17.4 及更高版本	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本

区域	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
欧洲地区（西班牙）	版本 17.4 及更高版本	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本
欧洲地区（斯德哥尔摩）	版本 17.4 及更高版本	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本
欧洲（苏黎世）	版本 17.4 及更高版本	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本
以色列（特拉维夫）	不可用	不可用	不可用	不可用	不可用	不可用	不可用
中东（巴林）	版本 17.4 及更高版本	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本
中东（阿联酋）	版本 17.4 及更高版本	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本
南美洲（圣保罗）	版本 17.4 及更高版本	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本

区域	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
AWS GovCloud（美国东部）	版本 17.4 及更高版本	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本
AWS GovCloud（美国西部）	版本 17.4 及更高版本	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本

支持 IAM 数据库身份验证的区域和 Aurora 数据库引擎

使用 Aurora 中的 IAM 数据库身份验证，您可以使用 AWS Identity and Access Management（IAM）数据库身份验证对您的数据库集群进行身份验证。利用此身份验证方法，您在连接到数据库集群时将无需使用密码。而是使用身份验证令牌。有关更多信息，请参阅 IAM 数据库身份验证。

主题

- 适用于 Aurora MySQL 的 IAM 数据库身份验证
- 适用于 Aurora PostgreSQL 的 IAM 数据库身份验证

适用于 Aurora MySQL 的 IAM 数据库身份验证

适用于 Aurora MySQL 的 IAM 数据库身份验证可用于所有区域中的以下版本：

- Aurora MySQL 3 – 所有可用版本
- Aurora MySQL 2 – 所有可用版本

适用于 Aurora PostgreSQL 的 IAM 数据库身份验证

适用于 Aurora PostgreSQL 的 IAM 数据库身份验证可用于所有区域中的以下引擎版本：

- Aurora PostgreSQL 16 – 所有可用版本

- Aurora PostgreSQL 15 – 所有可用版本
- Aurora PostgreSQL 14 – 所有可用版本
- Aurora PostgreSQL 13 – 所有可用版本
- Aurora PostgreSQL 12 – 所有可用版本
- Aurora PostgreSQL 11 – 所有可用版本

支持 Kerberos 身份验证的区域和 Aurora 数据库引擎

通过使用适用于 Aurora 的 Kerberos 身份验证，可以支持使用 Kerberos 和 Microsoft Active Directory 对数据库用户进行外部身份验证。使用 Kerberos 和 Active Directory 为数据库用户提供了单点登录和集中身份验证的优势。Kerberos 和 Active Directory 适用于 AWS Directory Service for Microsoft Active Directory，这是 AWS Directory Service 的一项功能。有关更多信息，请参阅 Kerberos 身份验证。

主题

- 适用于 Aurora MySQL 的 Kerberos 身份验证
- 适用于 Aurora PostgreSQL 的 Kerberos 身份验证

适用于 Aurora MySQL 的 Kerberos 身份验证

以下区域和引擎版本可用于面向 Aurora MySQL 的 Kerberos 身份验证。

区域	Aurora MySQL 版本 3
美国东部（俄亥俄州）	版本 3.03.0 及更高版本
美国东部（弗吉尼亚州北部）	版本 3.03.0 及更高版本
美国西部（加利福尼亚北部）	版本 3.03.0 及更高版本
美国西部（俄勒冈州）	版本 3.03.0 及更高版本
非洲（开普敦）	不可用
亚太地区（香港）	不可用
亚太地区（雅加达）	不可用

区域	Aurora MySQL 版本 3
亚太地区（孟买）	版本 3.03.0 及更高版本
亚太地区（大阪）	不可用
亚太地区（首尔）	版本 3.03.0 及更高版本
亚太地区（新加坡）	版本 3.03.0 及更高版本
亚太地区（悉尼）	版本 3.03.0 及更高版本
Asia Pacific (Tokyo)	版本 3.03.0 及更高版本
加拿大（中部）	版本 3.03.0 及更高版本
加拿大西部（卡尔加里）	不可用
中国（北京）	版本 3.03.0 及更高版本
中国（宁夏）	版本 3.03.0 及更高版本
欧洲地区（法兰克福）	版本 3.03.0 及更高版本
欧洲地区（爱尔兰）	版本 3.03.0 及更高版本
欧洲地区（伦敦）	版本 3.03.0 及更高版本
欧洲地区（米兰）	不可用
欧洲地区（巴黎）	版本 3.03.0 及更高版本
欧洲（西班牙）	不可用
欧洲地区（斯德哥尔摩）	版本 3.03.0 及更高版本
欧洲（苏黎世）	不可用
以色列（特拉维夫）	不可用
中东（巴林）	不可用

区域	Aurora MySQL 版本 3
中东（阿联酋）	不可用
南美洲（圣保罗）	版本 3.03.0 及更高版本
AWS GovCloud（美国东部）	版本 3.03.0 及更高版本
AWS GovCloud（美国西部）	版本 3.03.0 及更高版本

适用于 Aurora PostgreSQL 的 Kerberos 身份验证

以下区域和引擎版本可用于面向 Aurora PostgreSQL 的 Kerberos 身份验证。

区域	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
美国东部（俄亥俄州）	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
美国东部（弗吉尼亚州北部）	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
美国西部（加利福尼亚北部）	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
美国西部（俄勒冈州）	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
非洲（开普敦）	不可用	不可用	不可用	不可用	不可用	不可用

区域	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
亚太地区（香港）	不可用	不可用	不可用	不可用	不可用	不可用
亚太地区（海得拉巴）	不可用	不可用	不可用	不可用	不可用	不可用
亚太地区（雅加达）	不可用	不可用	不可用	不可用	不可用	不可用
亚太地区（墨尔本）	不可用	不可用	不可用	不可用	不可用	不可用
亚太地区（孟买）	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
亚太地区（大阪）	不可用	不可用	不可用	不可用	不可用	不可用
亚太地区（首尔）	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
亚太地区（新加坡）	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
亚太地区（悉尼）	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
Asia Pacific (Tokyo)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
加拿大（中部）	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本

区域	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
加拿大西部（卡尔加里）	不可用	不可用	不可用	不可用	不可用	不可用
中国（北京）	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
中国（宁夏）	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
欧洲地区（法兰克福）	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
欧洲地区（爱尔兰）	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
欧洲地区（伦敦）	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
欧洲地区（米兰）	不可用	不可用	不可用	不可用	不可用	不可用
欧洲地区（巴黎）	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
欧洲（西班牙）	不可用	不可用	不可用	不可用	不可用	不可用
欧洲地区（斯德哥尔摩）	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
欧洲（苏黎世）	不可用	不可用	不可用	不可用	不可用	不可用

区域	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
以色列（特拉维夫）	不可用	不可用	不可用	不可用	不可用	不可用
中东（巴林）	不可用	不可用	不可用	不可用	不可用	不可用
中东（阿联酋）	不可用	不可用	不可用	不可用	不可用	不可用
南美洲（圣保罗）	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
AWS GovCloud（美国东部）	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
AWS GovCloud（美国西部）	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本

支持 Aurora 机器学习的区域和数据库引擎

通过使用 Amazon Aurora 机器学习，您可以将您的 Aurora 数据库集群与以下 AWS 机器学习服务之一集成，具体取决于您的需求。这些服务分别支持特定的机器学习使用案例。

Amazon Bedrock 是一项完全托管式服务，可通过 API 提供人工智能公司的领先基础模型，并附带有助于构建和扩展生成式人工智能应用程序的开发工具。

Amazon Comprehend 是一项自然语言处理（NLP）服务，用于从文档中提取见解。通过将 Aurora 机器学习与 Amazon Comprehend 结合使用，您可以确定数据库表中文本的情绪。

SageMaker 是一项功能齐全的机器学习服务。数据科学家使用 Amazon SageMaker 为各种推理任务（例如欺诈检测）构建、训练和测试机器学习模型。通过将 Aurora 机器学习与 SageMaker 结合使用，数据库开发人员可以在 SQL 代码中调用 SageMaker 功能。

并非所有 AWS 区域都支持所有机器学习服务。只有某些 AWS 区域支持 Aurora 机器学习，因此可以从 Aurora 数据库集群访问这些服务。Aurora 机器学习的集成过程也因数据库引擎不同而异。有关更多信息，请参阅 使用 Amazon Aurora 机器学习。

主题

- 使用 Aurora MySQL 的 Aurora 机器学习
- 使用 Aurora PostgreSQL 的 Aurora 机器学习

使用 Aurora MySQL 的 Aurora 机器学习

仅 Aurora MySQL 版本 3.06 及更高版本支持 Amazon Bedrock。有关 Amazon Bedrock 的区域可用性的信息，请参阅《Amazon Bedrock 用户指南》中的 Model support by AWS 区域。

在表所列出的 AWS 区域中，Aurora MySQL 支持将 Aurora 机器学习与 Amazon Comprehend 和 Amazon SageMaker 结合使用。除了提供您的 Aurora MySQL 版本外，AWS 区域还必须支持您要使用的服务。有关 Amazon SageMaker 可用的 AWS 区域的列表，请参阅《Amazon Web Services 一般参考》中的 Amazon SageMaker 端点和限额。有关 Amazon Comprehend 可用的 AWS 区域的列表，请参阅《Amazon Web Services 一般参考》中的 Amazon Comprehend endpoints and quotas。

区域	Aurora MySQL 版本 3	Aurora MySQL 版本 2
美国东部（俄亥俄州）	版本 3.01.0 及更高版本	版本 2.07 及更高版本
美国东部（弗吉尼亚州北部）	版本 3.01.0 及更高版本	版本 2.07 及更高版本
美国西部（加利福尼亚北部）	版本 3.01.0 及更高版本	版本 2.07 及更高版本
美国西部（俄勒冈州）	版本 3.01.0 及更高版本	版本 2.07 及更高版本
非洲（开普敦）	不可用	不可用
亚太地区（香港）	版本 3.01.0 及更高版本	版本 2.07 及更高版本
亚太地区（海得拉巴）	版本 3.01.0 及更高版本	版本 2.07 及更高版本
亚太地区（雅加达）	版本 3.01.0 及更高版本	版本 2.07 及更高版本
亚太地区（墨尔本）	版本 3.01.0 及更高版本	版本 2.07 及更高版本

区域	Aurora MySQL 版本 3	Aurora MySQL 版本 2
亚太地区（孟买）	版本 3.01.0 及更高版本	版本 2.07 及更高版本
亚太地区（大阪）	版本 3.01.0 及更高版本	版本 2.07.3 及更高版本
亚太地区（首尔）	版本 3.01.0 及更高版本	版本 2.07 及更高版本
亚太地区（新加坡）	版本 3.01.0 及更高版本	版本 2.07 及更高版本
亚太地区（悉尼）	版本 3.01.0 及更高版本	版本 2.07 及更高版本
Asia Pacific (Tokyo)	版本 3.01.0 及更高版本	版本 2.07 及更高版本
加拿大（中部）	版本 3.01.0 及更高版本	版本 2.07 及更高版本
加拿大西部（卡尔加里）	版本 3.01.0 及更高版本	版本 2.07 及更高版本
中国（北京）	版本 3.01.0 及更高版本	版本 2.07 及更高版本
中国（宁夏）	版本 3.01.0 及更高版本	版本 2.07 及更高版本
欧洲地区（法兰克福）	版本 3.01.0 及更高版本	版本 2.07 及更高版本
欧洲地区（爱尔兰）	版本 3.01.0 及更高版本	版本 2.07 及更高版本
欧洲地区（伦敦）	版本 3.01.0 及更高版本	版本 2.07 及更高版本
欧洲地区（米兰）	不可用	不可用
欧洲地区（巴黎）	版本 3.01.0 及更高版本	版本 2.07 及更高版本
欧洲（西班牙）	版本 3.01.0 及更高版本	版本 2.07 及更高版本
欧洲地区（斯德哥尔摩）	版本 3.01.0 及更高版本	版本 2.07 及更高版本
欧洲（苏黎世）	版本 3.01.0 及更高版本	版本 2.07 及更高版本
以色列（特拉维夫）	版本 3.01.0 及更高版本	版本 2.07 及更高版本
中东（巴林）	版本 3.01.0 及更高版本	版本 2.07 及更高版本

区域	Aurora MySQL 版本 3	Aurora MySQL 版本 2
中东（阿联酋）	版本 3.01.0 及更高版本	版本 2.07 及更高版本
南美洲（圣保罗）	版本 3.01.0 及更高版本	版本 2.07 及更高版本
AWS GovCloud（美国东部）	版本 3.01.0 及更高版本	版本 2.07 及更高版本
AWS GovCloud（美国西部）	版本 3.01.0 及更高版本	版本 2.07 及更高版本

使用 Aurora PostgreSQL 的 Aurora 机器学习

有关 Aurora PostgreSQL 上 Amazon Bedrock 版本支持的信息，请参阅使用 Aurora PostgreSQL 作为 Amazon Bedrock 的知识库。

在表所列出的 AWS 区域中，Aurora PostgreSQL 支持将 Aurora 机器学习与 Amazon Comprehend 和 Amazon SageMaker 结合使用。除了提供您的 Aurora PostgreSQL 版本外，AWS 区域还必须支持您要使用的服务。有关 Amazon SageMaker 可用的 AWS 区域的列表，请参阅《Amazon Web Services 一般参考》中的 Amazon SageMaker 端点和限额。有关 Amazon Comprehend 可用的 AWS 区域的列表，请参阅《Amazon Web Services 一般参考》中的 Amazon Comprehend endpoints and quotas。

以下区域和引擎版本可用于面向 Aurora PostgreSQL 的 Aurora 机器学习。

区域	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
美国东部（俄亥俄州）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3	版本 13.3 及更高版本	版本 12.4 及更高版本	版本 11.9，11.12 及更高版本
美国东部（弗吉尼亚州北部）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3	版本 13.3 及更高版本	版本 12.4 及更高版本	版本 11.9，11.12 及更高版本

区域	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
美国西部（加利福尼亚北部）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3	版本 13.3 及更高版本	版本 12.4 及更高版本	版本 11.9，11.12 及更高版本
美国西部（俄勒冈州）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3	版本 13.3 及更高版本	版本 12.4 及更高版本	版本 11.9，11.12 及更高版本
非洲（开普敦）	不可用	不可用	不可用	不可用	不可用	不可用
亚太地区（香港）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3	版本 13.3 及更高版本	版本 12.4 及更高版本	版本 11.9，11.12 及更高版本
亚太地区（海得拉巴）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3	版本 13.3 及更高版本	版本 12.4 及更高版本	版本 11.9，11.12 及更高版本
亚太地区（雅加达）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3	版本 13.3 及更高版本	版本 12.4 及更高版本	版本 11.9，11.12 及更高版本
亚太地区（墨尔本）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3	版本 13.3 及更高版本	版本 12.4 及更高版本	版本 11.9，11.12 及更高版本

区域	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
亚太地区（孟买）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3	版本 13.3 及更高版本	版本 12.4 及更高版本	版本 11.9，11.12 及更高版本
亚太地区（大阪）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3	版本 13.3 及更高版本	版本 12.4 及更高版本	版本 11.9，11.12 及更高版本
亚太地区（首尔）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3	版本 13.3 及更高版本	版本 12.4 及更高版本	版本 11.9，11.12 及更高版本
亚太地区（新加坡）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3	版本 13.3 及更高版本	版本 12.4 及更高版本	版本 11.9，11.12 及更高版本
亚太地区（悉尼）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3	版本 13.3 及更高版本	版本 12.4 及更高版本	版本 11.9，11.12 及更高版本
Asia Pacific (Tokyo)	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3	版本 13.3 及更高版本	版本 12.4 及更高版本	版本 11.9，11.12 及更高版本
加拿大（中部）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3	版本 13.3 及更高版本	版本 12.4 及更高版本	版本 11.9，11.12 及更高版本

区域	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
加拿大西部（卡尔加里）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3	版本 13.3 及更高版本	版本 12.4 及更高版本	版本 11.9，11.12 及更高版本
中国（北京）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3	版本 13.3 及更高版本	版本 12.4 及更高版本	版本 11.9，11.12 及更高版本
中国（宁夏）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3	版本 13.3 及更高版本	版本 12.4 及更高版本	版本 11.9，11.12 及更高版本
欧洲地区（法兰克福）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3	版本 13.3 及更高版本	版本 12.4 及更高版本	版本 11.9，11.12 及更高版本
欧洲地区（爱尔兰）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3	版本 13.3 及更高版本	版本 12.4 及更高版本	版本 11.9，11.12 及更高版本
欧洲地区（伦敦）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3	版本 13.3 及更高版本	版本 12.4 及更高版本	版本 11.9，11.12 及更高版本
欧洲地区（米兰）	不可用	不可用	不可用	不可用	不可用	不可用

区域	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
欧洲地区（巴黎）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3	版本 13.3 及更高版本	版本 12.4 及更高版本	版本 11.9，11.12 及更高版本
欧洲（西班牙）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3	版本 13.3 及更高版本	版本 12.4 及更高版本	版本 11.9，11.12 及更高版本
欧洲地区（斯德哥尔摩）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3	版本 13.3 及更高版本	版本 12.4 及更高版本	版本 11.9，11.12 及更高版本
欧洲（苏黎世）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3	版本 13.3 及更高版本	版本 12.4 及更高版本	版本 11.9，11.12 及更高版本
以色列（特拉维夫）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3	版本 13.3 及更高版本	版本 12.4 及更高版本	版本 11.9，11.12 及更高版本
中东（巴林）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3	版本 13.3 及更高版本	版本 12.4 及更高版本	版本 11.9，11.12 及更高版本
中东（阿联酋）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3	版本 13.3 及更高版本	版本 12.4 及更高版本	版本 11.9，11.12 及更高版本

区域	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
南美洲（圣保罗）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3	版本 13.3 及更高版本	版本 12.4 及更高版本	版本 11.9，11.12 及更高版本
AWS GovCloud（美国东部）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3	版本 13.3 及更高版本	版本 12.4 及更高版本	版本 11.9，11.12 及更高版本
AWS GovCloud（美国西部）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3	版本 13.3 及更高版本	版本 12.4 及更高版本	版本 11.9，11.12 及更高版本

支持 Performance Insights 的区域和 Aurora 数据库引擎

Performance Insights 在现有 Amazon RDS 监控功能的基础上进行了扩展，以便展示并帮助您分析数据库性能。利用 Performance Insights 控制面板，您可以可视化 Amazon RDS 数据库实例负载上的数据库负载，并按等待状态、SQL 语句、主机或用户来筛选负载。有关更多信息，请参阅 Amazon Aurora 上的 Performance Insights 概述。

有关 Performance Insights 功能的区域、数据库引擎和实例类支持信息，请参阅支持性能详情功能的 Amazon Aurora 数据库引擎、区域和实例类。

主题

- 适用于 Aurora MySQL 的 Performance Insights
- 适用于 Aurora PostgreSQL 的 Performance Insights
- 适用于 Aurora Serverless 的 Performance Insights

适用于 Aurora MySQL 的 Performance Insights

> **Note**
>
> 如果您开启了并行查询，则对适用于 Aurora MySQL 的 Performance Insights 的引擎版本支持会有所不同。有关并行查询的更多信息，请参阅 Amazon Aurora MySQL 的并行查询。

主题

- 适用于 Aurora MySQL 的 Performance Insights 且关闭了并行查询
- 适用于 Aurora MySQL 的 Performance Insights 且开启了并行查询

适用于 Aurora MySQL 的 Performance Insights 且关闭了并行查询

以下区域和引擎版本可用于关闭了 Aurora MySQL 和并行查询的 Performance Insights。

区域	Aurora MySQL 版本 3	Aurora MySQL 版本 2
美国东部（俄亥俄州）	所有版本	所有版本
美国东部（弗吉尼亚州北部）	所有版本	所有版本
美国西部（加利福尼亚北部）	所有版本	所有版本
美国西部（俄勒冈州）	所有版本	所有版本
非洲（开普敦）	所有版本	所有版本
亚太地区（香港）	所有版本	所有版本
亚太地区（海得拉巴）	所有版本	所有版本
亚太地区（雅加达）	所有版本	所有版本
亚太地区（墨尔本）	所有版本	所有版本
亚太地区（孟买）	所有版本	所有版本
亚太地区（大阪）	所有版本	所有版本

区域	Aurora MySQL 版本 3	Aurora MySQL 版本 2
亚太地区（首尔）	所有版本	所有版本
亚太地区（新加坡）	所有版本	所有版本
亚太地区（悉尼）	所有版本	所有版本
Asia Pacific (Tokyo)	所有版本	所有版本
加拿大（中部）	所有版本	所有版本
加拿大西部（卡尔加里）	所有版本	所有版本
中国（北京）	所有版本	所有版本
中国（宁夏）	所有版本	所有版本
欧洲地区（法兰克福）	所有版本	所有版本
欧洲地区（爱尔兰）	所有版本	所有版本
欧洲地区（伦敦）	所有版本	所有版本
欧洲地区（米兰）	所有版本	所有版本
欧洲地区（巴黎）	所有版本	所有版本
欧洲（西班牙）	所有版本	所有版本
欧洲地区（斯德哥尔摩）	所有版本	所有版本
欧洲（苏黎世）	所有版本	所有版本
以色列（特拉维夫）	所有版本	所有版本
中东（巴林）	所有版本	所有版本
中东（阿联酋）	所有版本	所有版本
南美洲（圣保罗）	所有版本	所有版本

区域	Aurora MySQL 版本 3	Aurora MySQL 版本 2
AWS GovCloud（美国东部）	所有版本	所有版本
AWS GovCloud（美国西部）	所有版本	所有版本

适用于 Aurora MySQL 的 Performance Insights 且开启了并行查询

以下区域和引擎版本可用于开启了 Aurora MySQL 和并行查询的 Performance Insights。

区域	Aurora MySQL 版本 3	Aurora MySQL 版本 2
美国东部（俄亥俄州）	不可用	版本 2.09.0 及更高版本
美国东部（弗吉尼亚州北部）	不可用	版本 2.09.0 及更高版本
美国西部（加利福尼亚北部）	不可用	版本 2.09.0 及更高版本
美国西部（俄勒冈州）	不可用	版本 2.09.0 及更高版本
非洲（开普敦）	不可用	版本 2.09.0 及更高版本
亚太地区（香港）	不可用	版本 2.09.0 及更高版本
亚太地区（海得拉巴）	不可用	所有版本
亚太地区（雅加达）	不可用	版本 2.09.0 及更高版本
亚太地区（墨尔本）	不可用	版本 2.09.0 及更高版本
亚太地区（孟买）	不可用	版本 2.09.0 及更高版本
亚太地区（大阪）	不可用	版本 2.09.0 及更高版本
亚太地区（首尔）	不可用	版本 2.09.0 及更高版本
亚太地区（新加坡）	不可用	版本 2.09.0 及更高版本
亚太地区（悉尼）	不可用	版本 2.09.0 及更高版本
Asia Pacific (Tokyo)	不可用	版本 2.09.0 及更高版本

区域	Aurora MySQL 版本 3	Aurora MySQL 版本 2
加拿大（中部）	不可用	版本 2.09.0 及更高版本
加拿大西部（卡尔加里）	不可用	版本 2.09.0 及更高版本
中国（北京）	不可用	版本 2.09.0 及更高版本
中国（宁夏）	不可用	版本 2.09.0 及更高版本
欧洲地区（法兰克福）	不可用	版本 2.09.0 及更高版本
欧洲地区（爱尔兰）	不可用	版本 2.09.0 及更高版本
欧洲地区（伦敦）	不可用	版本 2.09.0 及更高版本
欧洲地区（米兰）	不可用	版本 2.09.0 及更高版本
欧洲地区（巴黎）	不可用	版本 2.09.0 及更高版本
欧洲（西班牙）	不可用	版本 2.09.0 及更高版本
欧洲地区（斯德哥尔摩）	不可用	版本 2.09.0 及更高版本
欧洲（苏黎世）	不可用	版本 2.09.0 及更高版本
以色列（特拉维夫）	不可用	版本 2.09.0 及更高版本
中东（巴林）	不可用	版本 2.09.0 及更高版本
中东（阿联酋）	不可用	版本 2.09.0 及更高版本
南美洲（圣保罗）	不可用	版本 2.09.0 及更高版本
AWS GovCloud（美国东部）	不可用	版本 2.09.0 及更高版本
AWS GovCloud（美国西部）	不可用	版本 2.09.0 及更高版本

适用于 Aurora PostgreSQL 的 Performance Insights

以下区域和引擎版本可用于面向 Aurora PostgreSQL 的 Performance Insights。

区域	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11	Aurora PostgreSQL 10
美国东部（俄亥俄州）	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
美国东部（弗吉尼亚州北部）	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
美国西部（加利福尼亚北部）	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
美国西部（俄勒冈州）	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
非洲（开普敦）	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
亚太地区（香港）	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
亚太地区（海得拉巴）	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
亚太地区（雅加达）	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本

区域	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11	Aurora PostgreSQL 10
亚太地区（墨尔本）	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
亚太地区（孟买）	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
亚太地区（大阪）	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
亚太地区（首尔）	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
亚太地区（新加坡）	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
亚太地区（悉尼）	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
Asia Pacific (Tokyo)	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
加拿大（中部）	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
加拿大西部（卡尔加里）	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
中国（北京）	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本

区域	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11	Aurora PostgreSQL 10
中国（宁夏）	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
欧洲地区（法兰克福）	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
欧洲地区（爱尔兰）	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
欧洲地区（伦敦）	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
欧洲地区（米兰）	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
欧洲地区（巴黎）	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
欧洲（西班牙）	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
欧洲地区（斯德哥尔摩）	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
欧洲（苏黎世）	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
以色列（特拉维夫）	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本

区域	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11	Aurora PostgreSQL 10
中东（巴林）	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
中东（阿联酋）	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
南美洲（圣保罗）	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
AWS GovCloud（美国东部）	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本
AWS GovCloud（美国西部）	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本	所有版本

适用于 Aurora Serverless 的 Performance Insights

Aurora Serverless v2 对于所有 MySQL 兼容版本和 PostgreSQL 兼容版本支持 Performance Insights。我们建议您将最小容量设置为至少 2 个 Aurora 容量单元（ACU）。

Aurora Serverless v1 不支持 Performance Insights。

支持与 Amazon Redshift 进行零 ETL 集成的区域和 Amazon 数据库引擎

Amazon Aurora 与 Amazon Redshift 的零 ETL 集成是一个完全托管式解决方案，用于在将事务数据写入 Aurora 集群后，在 Amazon Redshift 中提供这些数据。有关更多信息，请参阅 零 ETL 集成。

主题

- Aurora MySQL 零 ETL 集成

- Aurora PostgreSQL 零 ETL 集成

Aurora MySQL 零 ETL 集成

以下区域和引擎版本可用于 Aurora MySQL 与 Amazon Redshift 的零 ETL 集成。

区域	Aurora MySQL 版本 3
美国东部（俄亥俄州）	版本 3.05.2 及更高版本
美国东部（弗吉尼亚州北部）	版本 3.05.2 及更高版本
美国西部（加利福尼亚北部）	版本 3.05.2 及更高版本
美国西部（俄勒冈州）	版本 3.05.2 及更高版本
非洲（开普敦）	版本 3.05.2 及更高版本
亚太地区（香港）	版本 3.05.2 及更高版本
亚太地区（海得拉巴）	不可用
亚太地区（雅加达）	不可用
亚太地区（墨尔本）	不可用
亚太地区（孟买）	版本 3.05.2 及更高版本
亚太地区（大阪）	版本 3.05.2 及更高版本
亚太地区（首尔）	版本 3.05.2 及更高版本
亚太地区（新加坡）	版本 3.05.2 及更高版本
亚太地区（悉尼）	版本 3.05.2 及更高版本
Asia Pacific (Tokyo)	版本 3.05.2 及更高版本
加拿大（中部）	版本 3.05.2 及更高版本
加拿大西部（卡尔加里）	不可用

区域	Aurora MySQL 版本 3
中国（北京）	版本 3.05.2 及更高版本
中国（宁夏）	版本 3.05.2 及更高版本
欧洲地区（法兰克福）	版本 3.05.2 及更高版本
欧洲地区（爱尔兰）	版本 3.05.2 及更高版本
欧洲地区（伦敦）	版本 3.05.2 及更高版本
欧洲地区（米兰）	版本 3.05.2 及更高版本
欧洲地区（巴黎）	版本 3.05.2 及更高版本
欧洲（西班牙）	不可用
欧洲地区（斯德哥尔摩）	版本 3.05.2 及更高版本
欧洲（苏黎世）	不可用
以色列（特拉维夫）	不可用
中东（巴林）	版本 3.05.2 及更高版本
中东（阿联酋）	不可用
南美洲（圣保罗）	版本 3.05.2 及更高版本
AWS GovCloud（美国东部）	不可用
AWS GovCloud（美国西部）	不可用

Aurora PostgreSQL 零 ETL 集成

要获得 Aurora PostgreSQL 与 Amazon Redshift 的零 ETL 集成的预览版，您必须在美国东部（俄亥俄州）（us-east-2）AWS 区域的 Amazon RDS 数据库预览环境中创建集成。预览环境允许您测试 PostgreSQL 数据库引擎软件的测试版、候选发布版和早期生产版本。

您的源数据库集群必须运行 Aurora PostgreSQL（与 PostgreSQL 15.4 兼容和支持零 ETL）。

支持 Amazon RDS 代理的区域和 Aurora 数据库引擎

Amazon RDS 代理是一种完全托管的高可用性数据库代理，它通过汇集和共享已建立的数据库连接，提高应用程序的可扩展性。有关 RDS 代理的更多信息，请参阅 适用于 Aurora 的 Amazon RDS 代理。

主题

- Amazon RDS Proxy with Aurora MySQL
- Amazon RDS Proxy with Aurora PostgreSQL

Amazon RDS Proxy with Aurora MySQL

以下区域和引擎版本可用于面向 Aurora MySQL 的 RDS 代理。

区域	Aurora MySQL 版本 3	Aurora MySQL 版本 2
美国东部（俄亥俄州）	版本 3.01.0 及更高版本	版本 2.07 和版本 2.11 及更高版本
美国东部（弗吉尼亚州北部）	版本 3.01.0 及更高版本	版本 2.07 和版本 2.11 及更高版本
美国西部（加利福尼亚北部）	版本 3.01.0 及更高版本	版本 2.07 和版本 2.11 及更高版本
美国西部（俄勒冈州）	版本 3.01.0 及更高版本	版本 2.07 和版本 2.11 及更高版本
非洲（开普敦）	版本 3.01.0 及更高版本	版本 2.07 和版本 2.11 及更高版本
亚太地区（香港）	版本 3.01.0 及更高版本	版本 2.07 和版本 2.11 及更高版本
亚太地区（海得拉巴）	版本 3.01.0 及更高版本	版本 2.07 和版本 2.11 及更高版本

区域	Aurora MySQL 版本 3	Aurora MySQL 版本 2
亚太地区（雅加达）	版本 3.01.0 及更高版本	版本 2.07 和版本 2.11 及更高版本
亚太地区（墨尔本）	版本 3.01.0 及更高版本	版本 2.07 和版本 2.11 及更高版本
亚太地区（孟买）	版本 3.01.0 及更高版本	版本 2.07 和版本 2.11 及更高版本
亚太地区（大阪）	版本 3.01.0 及更高版本	版本 2.07 和版本 2.11 及更高版本
亚太地区（首尔）	版本 3.01.0 及更高版本	版本 2.07 和版本 2.11 及更高版本
亚太地区（新加坡）	版本 3.01.0 及更高版本	版本 2.07 和版本 2.11 及更高版本
亚太地区（悉尼）	版本 3.01.0 及更高版本	版本 2.07 和版本 2.11 及更高版本
Asia Pacific (Tokyo)	版本 3.01.0 及更高版本	版本 2.07 和版本 2.11 及更高版本
加拿大（中部）	版本 3.01.0 及更高版本	版本 2.07 和版本 2.11 及更高版本
加拿大西部（卡尔加里）	版本 3.01.0 及更高版本	版本 2.07 和版本 2.11 及更高版本
中国（北京）	版本 3.01.0 及更高版本	版本 2.07 和版本 2.11 及更高版本
中国（宁夏）	版本 3.01.0 及更高版本	版本 2.07 和版本 2.11 及更高版本
欧洲地区（法兰克福）	版本 3.01.0 及更高版本	版本 2.07 和版本 2.11 及更高版本

区域	Aurora MySQL 版本 3	Aurora MySQL 版本 2
欧洲地区（爱尔兰）	版本 3.01.0 及更高版本	版本 2.07 和版本 2.11 及更高版本
欧洲地区（伦敦）	版本 3.01.0 及更高版本	版本 2.07 和版本 2.11 及更高版本
欧洲地区（米兰）	版本 3.01.0 及更高版本	版本 2.07 和版本 2.11 及更高版本
欧洲地区（巴黎）	版本 3.01.0 及更高版本	版本 2.07 和版本 2.11 及更高版本
欧洲（西班牙）	版本 3.01.0 及更高版本	版本 2.07 和版本 2.11 及更高版本
欧洲地区（斯德哥尔摩）	版本 3.01.0 及更高版本	版本 2.07 和版本 2.11 及更高版本
欧洲（苏黎世）	版本 3.01.0 及更高版本	版本 2.07 和版本 2.11 及更高版本
以色列（特拉维夫）	版本 3.01.0 及更高版本	版本 2.07 和版本 2.11 及更高版本
中东（巴林）	版本 3.01.0 及更高版本	版本 2.07 和版本 2.11 及更高版本
中东（阿联酋）	版本 3.01.0 及更高版本	版本 2.07 和版本 2.11 及更高版本
南美洲（圣保罗）	版本 3.01.0 及更高版本	版本 2.07 和版本 2.11 及更高版本
AWS GovCloud（美国东部）	不可用	不可用
AWS GovCloud（美国西部）	不可用	不可用

Amazon RDS Proxy with Aurora PostgreSQL

以下区域和引擎版本可用于面向 Aurora PostgreSQL 的 RDS 代理。

区域	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
美国东部（俄亥俄州）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本
美国东部（弗吉尼亚州北部）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本
美国西部（加利福尼亚北部）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本
美国西部（俄勒冈州）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本
非洲（开普敦）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本
亚太地区（香港）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本

区域	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
亚太地区（海得拉巴）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本
亚太地区（雅加达）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本
亚太地区（墨尔本）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本
亚太地区（孟买）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本
亚太地区（大阪）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本
亚太地区（首尔）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本
亚太地区（新加坡）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本

区域	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
亚太地区（悉尼）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本
Asia Pacific (Tokyo)	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本
加拿大（中部）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本
加拿大西部（卡尔加里）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本
中国（北京）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本
中国（宁夏）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本
欧洲地区（法兰克福）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本

区域	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
欧洲地区（爱尔兰）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本
欧洲地区（伦敦）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本
欧洲地区（米兰）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本
欧洲地区（巴黎）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本
欧洲（西班牙）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本
欧洲地区（斯德哥尔摩）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本
欧洲（苏黎世）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本

区域	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
以色列（特拉维夫）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本
中东（巴林）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本
中东（阿联酋）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本
南美洲（圣保罗）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.4 及更高版本	版本 12.8 及更高版本	版本 11.9 和版本 11.13 及更高版本
AWS GovCloud（美国东部）	不可用	不可用	不可用	不可用	不可用	不可用
AWS GovCloud（美国西部）	不可用	不可用	不可用	不可用	不可用	不可用

支持 Secrets Manager 集成的区域和 Aurora 数据库引擎

借助 AWS Secrets Manager，您可以将代码中的硬编码凭证（包括数据库密码）替换为对 Secrets Manager 的 API 调用，从而以编程方式检索密钥。有关 Secrets Manager 的更多信息，请参阅《AWS Secrets Manager 用户指南》https://docs.aws.amazon.com/secretsmanager/latest/userguide/。

您可以指定 Amazon Aurora 在 Secrets Manager 中管理 Aurora 数据库集群的主用户密码。Aurora 生成密码，将其存储在 Secrets Manager 中，然后定期轮换密码。有关更多信息，请参阅 使用 Amazon Aurora 和 AWS Secrets Manager 管理密码。

Secrets Manager 集成适用于所有 AWS 区域。

支持 Aurora Serverless v2 的区域和 Aurora 数据库引擎

Aurora Serverless v2 是一种按需、自动扩展功能，旨在成为 Amazon Aurora 上运行间歇性或不可预测的工作负载的经济高效的方法。它会根据应用程序的需求自动扩展或缩减容量。与 Aurora Serverless v1 相比，扩展速度更快且更精细。使用 Aurora Serverless v2，每个集群可以包含一个写入器数据库实例和多个读取器数据库实例。您可以在同一集群中组合使用 Aurora Serverless v2 和传统预置数据库实例。有关更多信息，请参阅 使用 Aurora Serverless v2。

主题

- 适用于 Aurora MySQL 的 Aurora Serverless v2
- 适用于 Aurora PostgreSQL 的 Aurora Serverless v2

适用于 Aurora MySQL 的 Aurora Serverless v2

以下区域和引擎版本可用于面向 Aurora MySQL 的 Aurora Serverless v2。

区域	Aurora MySQL 版本 3
美国东部（弗吉尼亚州北部）	版本 3.02.0 及更高版本
美国东部（俄亥俄州）	版本 3.02.0 及更高版本
美国西部（加利福尼亚北部）	版本 3.02.0 及更高版本
美国西部（俄勒冈州）	版本 3.02.0 及更高版本
非洲（开普敦）	版本 3.02.0 及更高版本
亚太地区（香港）	版本 3.02.0 及更高版本
亚太地区（海得拉巴）	版本 3.02.3 及更高版本
亚太地区（雅加达）	版本 3.02.0 及更高版本

区域	Aurora MySQL 版本 3
亚太地区（马来西亚）	版本 3.04.3、3.05.2、3.06.1、3.07.1 及更高版本
亚太地区（墨尔本）	版本 3.02.3 及更高版本
亚太地区（孟买）	版本 3.02.0 及更高版本
亚太地区（大阪）	版本 3.02.0 及更高版本
亚太地区（首尔）	版本 3.02.0 及更高版本
亚太地区（新加坡）	版本 3.02.0 及更高版本
亚太地区（悉尼）	版本 3.02.0 及更高版本
亚太地区（东京）	版本 3.02.0 及更高版本
加拿大（中部）	版本 3.02.0 及更高版本
加拿大西部（卡尔加里）	版本 3.04.0 及更高版本
中国（北京）	版本 3.02.2 及更高版本
中国（宁夏）	版本 3.02.2 及更高版本
欧洲地区（法兰克福）	版本 3.02.0 及更高版本
欧洲地区（爱尔兰）	版本 3.02.0 及更高版本
欧洲地区（伦敦）	版本 3.02.0 及更高版本
欧洲地区（米兰）	版本 3.02.0 及更高版本
欧洲地区（巴黎）	版本 3.02.0 及更高版本
欧洲地区（西班牙）	版本 3.02.3 及更高版本
欧洲地区（斯德哥尔摩）	版本 3.02.0 及更高版本
欧洲（苏黎世）	版本 3.02.3 及更高版本

区域	Aurora MySQL 版本 3
以色列（特拉维夫）	版本 3.02.3 及更高版本，3.03.1 及更高版本
中东（巴林）	版本 3.02.0 及更高版本
中东（阿联酋）	版本 3.02.3 及更高版本
南美洲（圣保罗）	版本 3.02.0 及更高版本
AWS GovCloud（美国东部）	版本 3.02.2 及更高版本
AWS GovCloud（美国西部）	版本 3.02.2 及更高版本

ACU 的 Aurora Serverless v2 容量上限和下限可能会因您的引擎版本而异。有关更多信息，请参阅 Aurora Serverless v2 容量。

适用于 Aurora PostgreSQL 的 Aurora Serverless v2

以下区域和引擎版本可用于面向 Aurora PostgreSQL 的 Aurora Serverless v2。

区域	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13
美国东部（弗吉尼亚州北部）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.6 及更高版本
美国东部（俄亥俄州）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.6 及更高版本
美国西部（加利福尼亚北部）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.6 及更高版本
美国西部（俄勒冈州）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.6 及更高版本
非洲（开普敦）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.6 及更高版本

区域	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13
亚太地区（香港）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.6 及更高版本
亚太地区（海得拉巴）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.6 及更高版本	版本 13.9 及更高版本
亚太地区（雅加达）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.6 及更高版本
亚太地区（马来西亚）	版本 16.1 及更高版本	版本 15.4 及更高版本	版本 14.6、14.9 及更高版本	版本 13.9、13.12 及更高版本
亚太地区（墨尔本）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.6 及更高版本	版本 13.9 及更高版本
亚太地区（孟买）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.6 及更高版本
亚太地区（大阪）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.6 及更高版本
亚太地区（首尔）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.6 及更高版本
亚太地区（新加坡）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.6 及更高版本
亚太地区（悉尼）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.6 及更高版本
亚太地区（东京）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.6 及更高版本
加拿大（中部）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.6 及更高版本

区域	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13
加拿大西部（卡尔加里）	版本 16.1 及更高版本	版本 15.3 及更高版本	版本 14.6、14.8 及更高版本	版本 13.9、13.11 及更高版本
中国（北京）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.6 及更高版本
中国（宁夏）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.6 及更高版本
欧洲地区（法兰克福）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.6 及更高版本
欧洲地区（爱尔兰）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.6 及更高版本
欧洲地区（伦敦）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.6 及更高版本
欧洲地区（米兰）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.6 及更高版本
欧洲地区（巴黎）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.6 及更高版本
欧洲地区（西班牙）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.6 及更高版本	版本 13.9 及更高版本
欧洲地区（斯德哥尔摩）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.6 及更高版本
欧洲（苏黎世）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.6 及更高版本	版本 13.9 及更高版本
以色列（特拉维夫）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.6 及更高版本	版本 13.9 及更高版本

区域	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13
中东（巴林）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.6 及更高版本
中东（阿联酋）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.6 及更高版本	版本 13.9 及更高版本
南美洲（圣保罗）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.6 及更高版本
AWS GovCloud（美国东部）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.6 及更高版本
AWS GovCloud（美国西部）	版本 16.1 及更高版本	版本 15.2 及更高版本	版本 14.3 及更高版本	版本 13.6 及更高版本

ACU 的 Aurora Serverless v2 容量上限和下限可能会因您的引擎版本而异。有关详细信息，请参阅Aurora Serverless v2 容量。

Aurora Serverless v1

Important

AWS 已宣布 Aurora Serverless v1 的生命周期终止日期为 2025 年 3 月 31 日。强烈建议您在该日期之前将所有 Aurora Serverless v1 数据库集群升级到 Aurora Serverless v2。升级可能涉及数据库引擎主版本号的更改。因此，在生命周期终止日期之前做好规划、测试和实施这种切换非常重要。从 2025 年 1 月 8 日起，客户将无法再使用 AWS Management Console 或 CLI 创建新的 Aurora Serverless v1 集群或实例。有关迁移过程的信息，请参阅从 Aurora Serverless v1 集群升级到 Aurora Serverless v2。

Aurora Serverless v2 可以更快和更精细地进行扩展。Aurora Serverless v2 还能更好地兼容其他 Aurora 功能，例如读取器数据库实例。您可以在使用 Aurora Serverless v2 中了解 Aurora Serverless v2。

Aurora Serverless v1 是一种按需、自动扩展功能，旨在成为 Amazon Aurora 上运行间歇性或不可预测的工作负载的经济高效的方法。它会根据应用程序的需求，使用每个集群中的单个数据库实例自动启动、关闭以及扩展或缩减容量。有关更多信息，请参阅 使用 Amazon Aurora Serverless v1。

主题

- 适用于 Aurora MySQL 的 Aurora Serverless v1
- 适用于 Aurora PostgreSQL 的 Aurora Serverless v1

适用于 Aurora MySQL 的 Aurora Serverless v1

以下区域和引擎版本可用于面向 Aurora MySQL 的 Aurora Serverless v1。

区域	Aurora MySQL 版本 3	Aurora MySQL 版本 2
美国东部（弗吉尼亚州北部）	不可用	版本 2.11.4
美国东部（俄亥俄州）	不可用	版本 2.11.4
美国西部（加利福尼亚北部）	不可用	版本 2.11.4
美国西部（俄勒冈州）	不可用	版本 2.11.4
非洲（开普敦）	不可用	不可用
亚太地区（香港）	不可用	不可用
亚太地区（海得拉巴）	不可用	不可用
亚太地区（雅加达）	不可用	不可用
亚太地区（马来西亚）	不可用	不可用
亚太地区（墨尔本）	不可用	不可用
亚太地区（孟买）	不可用	版本 2.11.4
亚太地区（大阪）	不可用	不可用
亚太地区（首尔）	不可用	版本 2.11.4

区域	Aurora MySQL 版本 3	Aurora MySQL 版本 2
亚太地区（新加坡）	不可用	版本 2.11.4
亚太地区（悉尼）	不可用	版本 2.11.4
亚太地区（东京）	不可用	版本 2.11.4
加拿大（中部）	不可用	版本 2.11.4
加拿大西部（卡尔加里）	不可用	不可用
中国（北京）	不可用	不可用
中国（宁夏）	不可用	版本 2.11.4
欧洲地区（法兰克福）	不可用	版本 2.11.4
欧洲地区（爱尔兰）	不可用	版本 2.11.4
欧洲地区（伦敦）	不可用	版本 2.11.4
欧洲地区（米兰）	不可用	不可用
欧洲地区（巴黎）	不可用	版本 2.11.4
欧洲地区（西班牙）	不可用	不可用
欧洲地区（斯德哥尔摩）	不可用	不可用
欧洲（苏黎世）	不可用	不可用
以色列（特拉维夫）	不可用	不可用
中东（巴林）	不可用	不可用
中东（阿联酋）	不可用	不可用
南美洲（圣保罗）	不可用	不可用
AWS GovCloud（美国东部）	不可用	不可用

区域	Aurora MySQL 版本 3	Aurora MySQL 版本 2
AWS GovCloud（美国西部）	不可用	不可用

适用于 Aurora PostgreSQL 的 Aurora Serverless v1

以下区域和引擎版本可用于面向 Aurora PostgreSQL 的 Aurora Serverless v1。

区域	Aurora PostgreSQL 13
美国东部（弗吉尼亚州北部）	版本 13.12
美国东部（俄亥俄州）	版本 13.12
美国西部（加利福尼亚北部）	版本 13.12
美国西部（俄勒冈州）	版本 13.12
非洲（开普敦）	不可用
亚太地区（香港）	不可用
亚太地区（海得拉巴）	不可用
亚太地区（雅加达）	不可用
亚太地区（马来西亚）	不可用
亚太地区（墨尔本）	不可用
亚太地区（孟买）	版本 13.12
亚太地区（大阪）	不可用
亚太地区（首尔）	版本 13.12
亚太地区（新加坡）	版本 13.12
亚太地区（悉尼）	版本 13.12

区域	Aurora PostgreSQL 13
亚太地区（东京）	版本 13.12
加拿大（中部）	版本 13.12
加拿大西部（卡尔加里）	不可用
中国（北京）	不可用
中国（宁夏）	不可用
欧洲地区（法兰克福）	版本 13.12
欧洲地区（爱尔兰）	版本 13.12
欧洲地区（伦敦）	版本 13.12
欧洲地区（米兰）	不可用
欧洲地区（巴黎）	版本 13.12
欧洲地区（西班牙）	不可用
欧洲地区（斯德哥尔摩）	不可用
欧洲（苏黎世）	不可用
以色列（特拉维夫）	不可用
中东（巴林）	不可用
中东（阿联酋）	不可用
南美洲（圣保罗）	不可用
AWS GovCloud（美国东部）	不可用
AWS GovCloud（美国西部）	不可用

支持 RDS 数据 API 的区域和 Aurora 数据库引擎

RDS 数据 API（数据 API）为 Amazon Aurora 数据库集群提供了 Web 服务接口。您可以针对 HTTPS 端点运行 SQL 命令，而不必管理来自客户端应用程序的数据库连接。有关更多信息，请参阅 使用 Amazon RDS 数据 API。

对于 Aurora MySQL，Aurora Serverless v2 或预调配数据库集群不支持数据 API。

主题

- 适用于 Aurora MySQL Serverless v1 的数据 API
- 适用于 Aurora PostgreSQL Serverless v2 和预调配的数据 API
- 适用于 Aurora PostgreSQL Serverless v1 的数据 API

适用于 Aurora MySQL Serverless v1 的数据 API

以下区域和引擎版本可用于面向 Aurora MySQL Serverless v1 的数据 API。

区域	Aurora MySQL 版本 3	Aurora MySQL 版本 2
美国东部（俄亥俄州）	不可用	版本 2.11.3
美国东部（弗吉尼亚州北部）	不可用	版本 2.11.3
美国西部（加利福尼亚北部）	不可用	版本 2.11.3
美国西部（俄勒冈州）	不可用	版本 2.11.3
非洲（开普敦）	不可用	不可用
亚太地区（香港）	不可用	不可用
亚太地区（海得拉巴）	不可用	不可用
亚太地区（雅加达）	不可用	不可用
亚太地区（墨尔本）	不可用	不可用
亚太地区（孟买）	不可用	版本 2.11.3
亚太地区（大阪）	不可用	不可用

区域	Aurora MySQL 版本 3	Aurora MySQL 版本 2
亚太地区（首尔）	不可用	版本 2.11.3
亚太地区（新加坡）	不可用	版本 2.11.3
亚太地区（悉尼）	不可用	版本 2.11.3
Asia Pacific (Tokyo)	不可用	版本 2.11.3
加拿大（中部）	不可用	版本 2.11.3
加拿大西部（卡尔加里）	不可用	不可用
中国（北京）	不可用	不可用
中国（宁夏）	不可用	版本 2.11.3
欧洲地区（法兰克福）	不可用	版本 2.11.3
欧洲地区（爱尔兰）	不可用	版本 2.11.3
欧洲地区（伦敦）	不可用	版本 2.11.3
欧洲地区（米兰）	不可用	不可用
欧洲地区（巴黎）	不可用	版本 2.11.3
欧洲（西班牙）	不可用	不可用
欧洲地区（斯德哥尔摩）	不可用	不可用
欧洲（苏黎世）	不可用	不可用
以色列（特拉维夫）	不可用	不可用
中东（巴林）	不可用	不可用
中东（阿联酋）	不可用	不可用
南美洲（圣保罗）	不可用	不可用

区域	Aurora MySQL 版本 3	Aurora MySQL 版本 2
AWS GovCloud（美国东部）	不可用	不可用
AWS GovCloud（美国西部）	不可用	不可用

适用于 Aurora PostgreSQL Serverless v2 和预调配的数据 API

以下区域和引擎版本可用于面向 Aurora PostgreSQL Serverless v2 和预调配数据库集群的数据 API。

区域	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13
美国东部（俄亥俄州）	版本 16.1 及更高版本	版本 15.3 及更高版本	版本 14.8 及更高版本	版本 13.11 及更高版本
美国东部（弗吉尼亚州北部）	版本 16.1 及更高版本	版本 15.3 及更高版本	版本 14.8 及更高版本	版本 13.11 及更高版本
美国西部（加利福尼亚北部）	版本 16.1 及更高版本	版本 15.3 及更高版本	版本 14.8 及更高版本	版本 13.11 及更高版本
美国西部（俄勒冈州）	版本 16.1 及更高版本	版本 15.3 及更高版本	版本 14.8 及更高版本	版本 13.11 及更高版本
非洲（开普敦）	不可用	不可用	不可用	不可用
亚太地区（香港）	不可用	不可用	不可用	不可用
亚太地区（海得拉巴）	不可用	不可用	不可用	不可用
亚太地区（雅加达）	不可用	不可用	不可用	不可用
亚太地区（墨尔本）	不可用	不可用	不可用	不可用

区域	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13
亚太地区（孟买）	版本 16.1 及更高版本	版本 15.3 及更高版本	版本 14.8 及更高版本	版本 13.11 及更高版本
亚太地区（大阪）	不可用	不可用	不可用	不可用
亚太地区（首尔）	版本 16.1 及更高版本	版本 15.3 及更高版本	版本 14.8 及更高版本	版本 13.11 及更高版本
亚太地区（新加坡）	版本 16.1 及更高版本	版本 15.3 及更高版本	版本 14.8 及更高版本	版本 13.11 及更高版本
亚太地区（悉尼）	版本 16.1 及更高版本	版本 15.3 及更高版本	版本 14.8 及更高版本	版本 13.11 及更高版本
Asia Pacific (Tokyo)	版本 16.1 及更高版本	版本 15.3 及更高版本	版本 14.8 及更高版本	版本 13.11 及更高版本
加拿大（中部）	版本 16.1 及更高版本	版本 15.3 及更高版本	版本 14.8 及更高版本	版本 13.11 及更高版本
加拿大西部（卡尔加里）	不可用	不可用	不可用	不可用
中国（北京）	不可用	不可用	不可用	不可用
中国（宁夏）	不可用	不可用	不可用	不可用
欧洲地区（法兰克福）	版本 16.1 及更高版本	版本 15.3 及更高版本	版本 14.8 及更高版本	版本 13.11 及更高版本
欧洲地区（爱尔兰）	版本 16.1 及更高版本	版本 15.3 及更高版本	版本 14.8 及更高版本	版本 13.11 及更高版本
欧洲地区（伦敦）	版本 16.1 及更高版本	版本 15.3 及更高版本	版本 14.8 及更高版本	版本 13.11 及更高版本

区域	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13
欧洲地区（米兰）	不可用	不可用	不可用	不可用
欧洲地区（巴黎）	版本 16.1 及更高版本	版本 15.3 及更高版本	版本 14.8 及更高版本	版本 13.11 及更高版本
欧洲（西班牙）	不可用	不可用	不可用	不可用
欧洲地区（斯德哥尔摩）	不可用	不可用	不可用	不可用
欧洲（苏黎世）	不可用	不可用	不可用	不可用
以色列（特拉维夫）	不可用	不可用	不可用	不可用
中东（巴林）	不可用	不可用	不可用	不可用
中东（阿联酋）	不可用	不可用	不可用	不可用
南美洲（圣保罗）	不可用	不可用	不可用	不可用
AWS GovCloud（美国东部）	不可用	不可用	不可用	不可用
AWS GovCloud（美国西部）	不可用	不可用	不可用	不可用

适用于 Aurora PostgreSQL Serverless v1 的数据 API

以下区域和引擎版本可用于面向 Aurora PostgreSQL Serverless v1 的数据 API。

区域	Aurora PostgreSQL 13	Aurora PostgreSQL 11
美国东部（俄亥俄州）	版本 13.9	版本 11.18
美国东部（弗吉尼亚州北部）	版本 13.9	版本 11.18
美国西部（加利福尼亚北部）	版本 13.9	版本 11.18
美国西部（俄勒冈州）	版本 13.9	版本 11.18
非洲（开普敦）	不可用	不可用
亚太地区（香港）	不可用	不可用
亚太地区（海得拉巴）	不可用	不可用
亚太地区（雅加达）	不可用	不可用
亚太地区（墨尔本）	不可用	不可用
亚太地区（孟买）	版本 13.9	版本 11.18
亚太地区（大阪）	不可用	不可用
亚太地区（首尔）	版本 13.9	版本 11.18
亚太地区（新加坡）	版本 13.9	版本 11.18
亚太地区（悉尼）	版本 13.9	版本 11.18
Asia Pacific (Tokyo)	版本 13.9	版本 11.18
加拿大（中部）	版本 13.9	版本 11.18
中国（北京）	不可用	不可用
中国（宁夏）	不可用	不可用
欧洲地区（法兰克福）	版本 13.9	版本 11.18
欧洲地区（爱尔兰）	版本 13.9	版本 11.18

区域	Aurora PostgreSQL 13	Aurora PostgreSQL 11
欧洲地区（伦敦）	版本 13.9	版本 11.18
欧洲地区（米兰）	不可用	不可用
欧洲地区（巴黎）	版本 13.9	版本 11.18
欧洲（西班牙）	不可用	不可用
欧洲地区（斯德哥尔摩）	不可用	不可用
欧洲（苏黎世）	不可用	不可用
以色列（特拉维夫）	不可用	不可用
中东（巴林）	不可用	不可用
中东（阿联酋）	不可用	不可用
南美洲（圣保罗）	不可用	不可用
AWS GovCloud（美国东部）	不可用	不可用
AWS GovCloud（美国西部）	不可用	不可用

支持零停机修补（ZDP）的区域和 Aurora 数据库引擎

对 Aurora 数据库集群执行升级时可能会在数据库关闭和升级期间发生中断。默认情况下，如果在数据库运行时开始升级，则会丢失数据库集群正在处理的所有连接和事务。如果等到数据库处于空闲状态再执行升级，则可能需要等待很长时间。

零停机时间修补（ZDP）功能尝试在 Aurora 升级期间尽力保留客户端连接。如果 ZDP 成功完成，则应用程序会话将保留，并且数据库引擎将在升级过程中重新启动。数据库引擎重新启动可能会导致吞吐量下降，持续时间从几秒钟到大约一分钟不等。

有关 ZDP 可用于 Aurora MySQL 升级的条件和引擎版本的详细信息，请参阅使用零停机时间修补。

有关 ZDP 可用于 Aurora PostgreSQL 升级的条件和引擎版本的详细信息，请参阅次要版本升级和零停机时间修补。

支持 Aurora 引擎原生功能的区域和数据库引擎

Aurora 数据库引擎还支持其他特性以及专门针对 Aurora 的功能。对于特定的 Aurora 数据库引擎、版本或区域，某些引擎原生特性可能仅提供有限的支持或受限制的权限。

主题

- Aurora MySQL 的引擎原生特性
- Aurora PostgreSQL 的引擎原生特性

Aurora MySQL 的引擎原生特性

以下是 Aurora MySQL 的引擎原生特性。

- 高级审计
- 回溯
- 错误注入查询
- 集群内写入转发
- 并行查询

Aurora PostgreSQL 的引擎原生特性

以下是 Aurora PostgreSQL 的引擎原生特性。

- Babelfish
- 错误注入查询
- 查询计划管理

Amazon Aurora 端点连接

Amazon Aurora 通常涉及数据库实例集群而不是单个实例。每个连接均由特定的数据库实例处理。在连接到 Aurora 集群时，您指定的主机名和端口将指向名为终端节点的中间处理程序。Aurora 使用终端节点机制来提取这些连接。因此，当一些数据库实例不可用时，您不必对所有主机名进行硬编码，也不必编写自己的逻辑来平衡连接和重新路由连接。

对于某些 Aurora 任务，不同的实例或实例组执行不同的角色。例如，主实例处理所有数据定义语言 (DDL) 和数据操作语言 (DML) 语句。最多 15 个 Aurora 副本处理只读查询流量。

主题

- Aurora 终端节点的类型
- 查看 Aurora 集群的终端节点
- Aurora 终端节点如何使用高可用性
- Amazon Aurora 的集群端点
- Amazon Aurora 的读取器端点
- Amazon Aurora 的实例端点
- Amazon Aurora 的自定义端点

Aurora 终端节点的类型

通过使用终端节点，您可以根据使用案例将每个连接映射到相应的实例或实例组。例如，要执行 DDL 语句，您可以连接到作为主实例的任一实例。要执行查询，您可以连接到读取器端点，并通过 Aurora 自动在所有 Aurora 副本之间执行连接平衡。对于具有不同容量或配置的数据库实例的集群，您可以连接到与不同的数据库实例子集关联的自定义终端节点。对于诊断或优化，您可以连接到特定实例终端节点以检查有关特定数据库实例的详细信息。

终端节点表示为包含主机地址和端口的 Aurora 特定的 URL。可从 Aurora 数据库集群使用以下类型的终端节点。

集群端点

连接到集群的主实例来开发和测试应用程序，并执行诸如 `INSERT` 语句等转换以及 DDL、DML 和 ETL 操作。使用 AWS Management Console、AWS CLI 或 Amazon RDS API 查找集群端点位置，如查看 Aurora 集群的终端节点中所述。

有关集群端点的更多信息，请参阅 Amazon Aurora 的集群端点。

读取器端点

执行查询。Aurora 会自动在所有 Aurora 副本之间执行连接平衡。使用 AWS Management Console、AWS CLI 或 Amazon RDS API 查找读取器端点位置，如查看 Aurora 集群的终端节点中所述。

有关读取器端点的更多信息，请参阅 Amazon Aurora 的读取器端点。

实例端点

检查有关特定数据库实例的详细信息以进行诊断或调整。只能在 AWS Management Console中实例的实例详细信息页面上查找每个实例的实例端点位置。

有关实例端点的更多信息，请参阅 Amazon Aurora 的实例端点。

自定义终端节点

连接到数据库集群上不同的数据库实例子集。当数据库集群内具有不同的实例容量和配置时，这非常有用。使用 AWS Management Console、AWS CLI 或 Amazon RDS API 查找自定义端点位置，如查看 Aurora 集群的终端节点中所述。

有关自定义端点的更多信息，请参阅 Amazon Aurora 的自定义端点。

Aurora Global Database 写入器端点

Aurora Global Database 有一种特殊的端点，其用途与独立 Aurora 集群的集群端点相同。它可同时处理写入请求和读取请求。当辅助集群由于切换或失效转移而成为新的主集群时，Aurora 会自动将此端点切换为指向另一个 AWS 区域中的新主集群的集群端点。这样，您就不必将 AWS 区域编码为应用程序的连接字符串，也不必在全局数据库的布局发生变化时更改连接字符串。Aurora 会在您设置 Aurora Global Database 时创建此端点，例如在 AWS Management Console中为 Aurora 集群选择添加区域。

有关如何在 Aurora Global Database 中使用此类端点的信息，请参阅 连接到 Amazon Aurora Global Database。

查看 Aurora 集群的终端节点

虽然您只能在 AWS Management Console中的实例详细信息页面上查找实例端点位置，但可以使用控制台、AWS CLI 或 Amazon RDS API 来查找集群端点、读取器端点和自定义端点的位置。

Console

在 AWS Management Console中，在集群的实例详细信息页面中查找集群端点、读取器端点和任何自定义端点。可在每个实例的详细信息页面中查看实例终端节点。在连接时，将关联的端口号（跟在冒号后面）附加到详细信息页面上显示的端点名称。

AWS CLI

利用 AWS CLI，您可以在 describe-db-clusters 命令的输出中查找写入器端点、读取器端点以及任何自定义端点。例如，以下命令显示当前 AWS 区域中所有集群的端点属性。

```
aws rds describe-db-clusters --query '*[].
{Endpoint:Endpoint,ReaderEndpoint:ReaderEndpoint,CustomEndpoints:CustomEndpoints}'
```

Amazon RDS API

利用 Amazon RDS API，您可以通过调用 DescribeDBClusterEndpoints 操作来检索端点。

Aurora 终端节点如何使用高可用性

对于高可用性非常重要的集群，使用集群端点进行读/写或通用连接，并使用读取器端点进行只读连接。写入器和读取器端点比实例端点更好地管理数据库实例失效转移。与实例端点不同，如果集群中的数据库实例变得不可用，写入器和读取器端点会自动更改其连接到的数据库实例。有关集群和读取器端点的更多信息，请参阅 Amazon Aurora 的集群端点和 Amazon Aurora 的读取器端点。

如果数据库集群的主数据库实例失败，Aurora 将自动失效转移到新的主数据库实例。它通过将现有 Aurora 副本提升为新的主数据库实例或者创建新的主数据库实例来完成该操作。如果发生了失效转移，您可以使用集群端点重新连接到新提升或新创建的主数据库实例，或者使用读取器端点重新连接到数据库集群中的 Aurora 副本之一。在失效转移期间，在将 Aurora 副本提升为新的主数据库实例之后，读取器终端节点可能会在很短的时间内将连接定向到数据库集群的新主数据库实例。

如果您设计自己的应用程序逻辑来管理与实例终端节点的连接，则可以手动或以编程方式搜索数据库集群中生成的可用数据库实例集。使用 describe-db-clusters AWS CLI 命令或 DescribeDBClusters RDS API 操作查找数据库集群和读取器端点、数据库实例、数据库实例是否为读取器及其提升层级。然后，您可以在失效转移后确认其实例类，并连接到适当的实例终端节点。

有关失效转移的更多信息，请参阅Aurora 数据库集群的容错能力。

有关 Amazon Aurora 中的高可用性的更多信息，请参阅 Amazon Aurora 的高可用性。

Amazon Aurora 的集群端点

Aurora 数据库集群的集群终端节点（或读取器终端节点）连接到该数据库集群的当前主数据库实例。此终端节点是唯一可以执行写操作（如 DDL 语句）的终端节点。因此，集群终端节点是您在首次设置集群时或集群仅包含单个数据库实例时连接到的终端节点。

每个 Aurora 数据库集群均有一个集群终端节点和一个主数据库实例。

对数据库集群上的所有写入操作使用集群终端节点，这些操作包括插入、更新、删除和 DDL 更改。您还可以对读取操作（如查询）使用集群终端节点。

集群终端节点为数据库集群的读取/写入连接提供故障转移支持。如果数据库集群的当前主数据库实例失败，Aurora 将自动故障转移到新的主数据库实例。在故障转移期间，数据库集群将继续为从新的主数据库实例到集群终端节点的请求提供服务，对服务造成的中断最少。

以下示例介绍 Aurora MySQL 数据库集群中的集群终端节点。

```
mydbcluster.cluster-c7tj4example.us-east-1.rds.amazonaws.com:3306
```

每个 Aurora 集群都有一个内置集群端点，其名称和其它属性由 Aurora 管理。您无法创建、删除或修改此类端点。

在管理集群、执行提取、转换、加载 (ETL) 操作或开发和测试应用程序时，可以使用集群终端节点。集群终端节点将连接到集群的主实例。主实例是您可以在其中创建表和索引、运行 INSERT 语句以及执行其他 DDL 和 DML 操作的唯一数据库实例。

当故障转移机制将新数据库实例提升为集群的读/写主实例时，集群终端节点指向的物理 IP 地址会发生更改。如果您使用任意形式的连接池或其他多路复用，请准备刷新或减少任何缓存的 DNS 信息的生存时间。这样做可确保您不会尝试与在故障转移后不可用或现在为只读的数据库实例建立读/写连接。

Amazon Aurora 的读取器端点

Aurora 数据库集群的读取器端点为数据库集群的只读连接提供连接平衡支持。对读取操作 (如查询) 使用读取器终端节点。通过在只读 Aurora 副本上处理这些语句，此终端节点可减少主实例上的开销。它还可以帮助集群扩展容量以处理同时进行的 SELECT 查询，扩展的容量与集群中的 Aurora 副本数成比例。每个 Aurora 数据库集群均有一个读取器终端节点。

如果集群包含一个或多个 Aurora 副本，则读取器端点将对 Aurora 副本间的每个连接请求进行连接平衡。在这种情况下，您只能在该会话中执行只读语句，例如 SELECT。如果集群仅包含主实例而不包含 Aurora 副本，则读取器终端节点将连接到主实例。在这种情况下，您可以通过终端节点执行写入操作。

以下示例介绍 Aurora MySQL 数据库集群中的读取器终端节点。

```
mydbcluster.cluster-ro-c7tj4example.us-east-1.rds.amazonaws.com:3306
```

您将读取器终端节点用于 Aurora 集群的只读连接。此端点使用连接平衡机制来帮助您的集群处理查询密集型工作负载。读取器端点是您向在集群上执行报告或其他只读操作的应用程序提供的端点。

读取器端点对到 Aurora 数据库集群中的可用 Aurora 副本的连接执行连接平衡。它不会对单独的查询进行连接平衡。如果要对每个查询进行连接均衡以分布数据库集群的读取工作负载，请为每个查询打开与读取器端点的新连接。

每个 Aurora 集群都有一个内置读取器终端节点，其名称和其他属性由 Aurora 管理。您无法创建、删除或修改此类终端节点。

如果您的集群仅包含主目标（实例或数据库分片组）而不包含 Aurora 副本，则读取器端点将连接到主实例。在这种情况下，您可以通过此终端节点执行写入操作。

> **Tip**
>
> 通过 RDS 代理，您可以为 Aurora 集群创建其他只读终端节点。这些端点执行与 Aurora 读取器端点相同的连接平衡。如果读取器实例不可用，则应用程序可以比 Aurora 读取器终端节点更快地重新连接到代理终端节点。代理终端节点还可以利用其他代理功能，例如多路复用。有关更多信息，请参阅 将读取器终端节点与 Aurora 集群结合使用。

Amazon Aurora 的实例端点

实例终端节点 连接到 Aurora 集群中的特定数据库实例。数据库集群中的每个数据库实例具有自己的唯一实例终端节点。因此，数据库集群的当前主数据库实例具有一个实例终端节点，并且数据库集群中的每个 Aurora 副本都具有一个实例终端节点。

对于可能不适合使用集群终端节点或读取器终端节点的场景，实例终端节点提供对与数据库集群连接的直接控制。例如，根据工作负载类型，客户端应用程序可能需要更精细的连接平衡。在这种情况下，您可以配置多个客户端以连接到数据库集群中的不同 Aurora 副本，以便分配读取工作负载。有关在 Aurora PostgreSQL 故障转移后使用实例终端节点提高连接速度的示例，请参阅Amazon Aurora PostgreSQL 的快速故障转移。有关在 Aurora MySQL 故障转移后使用实例端点提高连接速度的示例，请参阅 MariaDB Connector/J 故障转移支持 – 案例 Amazon Aurora。

以下示例介绍 Aurora MySQL 数据库集群中数据库实例的实例终端节点。

```
mydbinstance.c7tj4example.us-east-1.rds.amazonaws.com:3306
```

Aurora 集群中的每个数据库实例均有自己的内置实例终端节点，其名称和其他属性由 Aurora 管理。您无法创建、删除或修改此类终端节点。如果您使用 Amazon RDS，可能会熟悉实例终端节点。但是，使用 Aurora 时，与实例终端节点相比，您通常更频繁使用写入器和读取器终端节点。

在日常的 Aurora 操作中，使用实例终端节点的主要方式是诊断影响 Aurora 集群中某个特定实例的容量或性能问题。在连接到特定实例时，您可以检查其状态变量、指标等。这样做可以帮助您确定该实例与集群中其他实例的不同之处。

在高级使用案例中，您可能会以不同的方式配置某些数据库实例。在这种情况下，使用实例终端节点直接连接到更小、更大或具有与其他实例具有不同特征的实例。此外，设置故障转移优先级，以便此特殊

的数据库实例是作为主实例接管的最后选择。我们建议您在此类情况下使用自定义终端节点而不是实例终端节点。这样做可以在您向集群中添加更多数据库实例时简化连接管理和高可用性。

Amazon Aurora 的自定义端点

Aurora 集群的自定义终端节点 表示一组选定数据库实例。在连接到端点时，Aurora 会执行连接平衡并选择组中的某个实例来处理连接。您可以定义此终端节点引用的实例，并确定此终端节点的用途。

在您创建自定义终端节点之前，Aurora 数据库集群没有自定义终端节点。您可以为每个预调配的 Aurora 集群或 Aurora Serverless v2 集群创建最多 5 个自定义端点。您无法对 Aurora Serverless v1 集群使用自定义终端节点。

自定义端点根据数据库实例的只读或读/写功能以外的条件，提供平衡的数据库连接。例如，您可以定义自定义终端节点以连接到使用特定的 AWS 实例类或特定的数据库参数组的实例。然后，您可以将此自定义终端节点的相关信息告知特定的用户组。例如，您可以将内部用户定向到低容量实例以生成报告或执行临时（一次性）查询，并将生产流量定向到高容量实例。

由于连接可以转到与自定义终端节点关联的任何数据库实例，因此，建议您确保该组中的所有数据库实例共享一些类似的特性。这样做可确保性能、内存容量等对于连接到该终端节点的每个人都是一致的。

此功能适用于具有特殊类型的工作负载的高级用户，在这些工作负载下，使集群中的所有 Aurora 副本保持相同是不切实际的。利用自定义终端节点，您可以预测用于每个连接的数据库实例的容量。在使用自定义终端节点时，通常不使用该集群的读取器终端节点。

以下示例介绍 Aurora MySQL 数据库集群中数据库实例的自定义终端节点。

```
myendpoint.cluster-custom-c7tj4example.us-east-1.rds.amazonaws.com:3306
```

当集群包含具有不同容量和配置设置的数据库实例时，您可以使用自定义终端节点来简化连接管理。

以前，您可能已使用 CNAMES 机制从您自己的域设置域名服务 (DNS) 别名以便获得类似结果。通过使用自定义终端节点，可以避免在集群增大或缩小时更新 CNAME 记录。自定义终端节点还意味着，您可以使用加密的传输层安全性/安全套接字层 (TLS/SSL) 连接。

您可以拥有多组专用数据库实例，而不是为每个特定目的使用一个数据库实例并连接到其实例终端节点。在这种情况下，每个组都有自己的自定义终端节点。这样一来，在专用于报告或处理生产任务或内部查询等任务的所有实例之间，Aurora 就可以执行连接平衡。自定义端点在实例之间被动分配连接，使用 DNS 随机返回其中一个实例的 IP 地址。如果组中的某个数据库实例变为不可用，则 Aurora 会将后续自定义终端节点连接定向到与同一终端节点关联的其他数据库实例之一。

主题

- Amazon Aurora 中自定义端点的注意事项
- 创建自定义终端节点
- 查看自定义终端节点
- 编辑自定义端点
- 删除自定义终端节点
- Amazon Aurora 的自定义端点的 AWS CLI 示例

Amazon Aurora 中自定义端点的注意事项

使用以下各节来管理、指定属性，并使用自定义端点的成员资格规则。

主题

- 管理自定义终端节点
- 指定自定义终端节点的属性
- 自定义终端节点的成员资格规则

管理自定义终端节点

由于新创建的 Aurora 集群没有自定义终端节点，因此，您必须自行创建和管理这些对象。您可以使用 AWS Management Console、AWS CLI 或 Amazon RDS API 完成此操作。

> **Note**
>
> 您还必须为从快照还原的 Aurora 集群创建和管理自定义终端节点。快照中不包含自定义终端节点。您在还原后再次创建它们，并在还原的集群与原始集群位于同一区域时选择新的终端节点名称。

要从 AWS Management Console 中使用自定义终端节点，您可以导航到 Aurora 集群的详细信息页，并使用 Custom Endpoints (自定义终端节点) 部分下的控件。

要从 AWS CLI 使用自定义终端节点，您可以使用以下操作：

- create-db-cluster-endpoint

- describe-db-cluster-endpoints
- modify-db-cluster-endpoint
- delete-db-cluster-endpoint

要通过 Amazon RDS API 使用自定义终端节点，您可以使用以下函数：

- CreateDBClusterEndpoint
- DescribeDBClusterEndpoints
- ModifyDBClusterEndpoint
- DeleteDBClusterEndpoint

指定自定义终端节点的属性

自定义终端节点名称的最大长度为 63 个字符。名称格式如下：

```
endpoint_name.cluster-custom-customer_DNS_identifier.AWS_Region.rds.amazonaws.com
```

您不能为同一 AWS 区域中的多个集群重用相同的自定义端点名称。自定义 DNS 标识符是与特定 AWS 区域中您的 AWS 账户相关联的唯一标识符。

每个自定义终端节点都有一个关联的类型，该类型将确定哪些数据库实例有资格与该终端节点关联。目前，类型可以是 READER 或 ANY。对于自定义终端节点类型，应注意以下事项：

- 无法在 AWS Management Console中选择自定义端点类型。通过 AWS Management Console 创建的所有自定义终端节点都具有类型 ANY。

  您可以使用 AWS CLI 或 Amazon RDS API 设置和修改自定义端点类型。
- 只有读取器数据库实例可以是 READER 自定义端点的一部分。
- 读取器和写入器数据库实例都可以是 ANY 自定义端点的一部分。Aurora 以相同的概率将到 ANY 类型的集群终端节点的连接定向到任何关联的数据库实例。ANY 类型适用于使用任意复制拓扑的集群。
- 如果您尝试根据集群的复制配置创建类型不合适的自定义终端节点，则 Aurora 将返回错误。

自定义终端节点的成员资格规则

在将数据库实例添加到自定义终端节点或将其从自定义终端节点中删除时，与该数据库实例的任何现有连接都将保持活动状态。

您可以定义要包含在自定义终端节点中或从其中排除的数据库实例的列表。我们将这些列表分别称为静态 和排除 列表。您可以使用包含/排除机制进一步细分数据库实例组，并确保自定义终端节点集涵盖集群中的所有数据库实例。每个自定义终端节点只能包含其中一种列表类型。

在 AWS Management Console 中：

- 该选项由复选框 Attach future instances added to this cluster（附加将来添加到此集群的实例）表示。如果清除该复选框，则自定义终端节点将使用仅包含页面中指定的数据库实例的静态列表。选中此复选框后，自定义终端节点将使用排除列表。在这种情况下，自定义终端节点表示集群中的所有数据库实例（包括您将来添加的任何实例），但页面中未选中的实例除外。
- 控制台不允许您指定端点类型。使用控制台创建的任何自定义端点均属于 ANY 类型。

  因此，当数据库实例因失效转移或提升而在写入器和读取器之间变换角色时，Aurora 不会更改自定义端点的成员资格。

在 AWS CLI 和 Amazon RDS API 中：

- 您可以指定端点类型。因此，当端点类型设置为 READER 时，端点成员资格将在失效转移和提升期间自动调整。

  例如，类型为 READER 的自定义端点包含一个 Aurora 副本，此副本随后被提升为写入器数据库实例。新的写入器实例不再是自定义端点的一部分。
- 您可以在各个成员更改其角色后将其添加到列表中或从列表中删除。使用 modify-db-cluster-endpoint AWS CLI 命令或 ModifyDBClusterEndpoint API 操作。

您可以将一个数据库实例与多个自定义终端节点关联。例如，假设您将新数据库实例添加到集群，或者 Aurora 通过自动扩展机制自动添加数据库实例。在这些情况下，数据库实例将添加到它符合条件的所有自定义终端节点。要将数据库实例添加到哪些端点基于自定义端点类型 READER 或 ANY，以及为每个端点定义的任何静态或排除列表。例如，如果终端节点包含数据库实例的静态列表，则新添加的 Aurora 副本不会添加到该终端节点。相反，如果终端节点具有排除列表，则新添加的 Aurora 副本将添加到终端节点（如果它们未在排除列表中指定且其角色与自定义终端节点的类型匹配）。

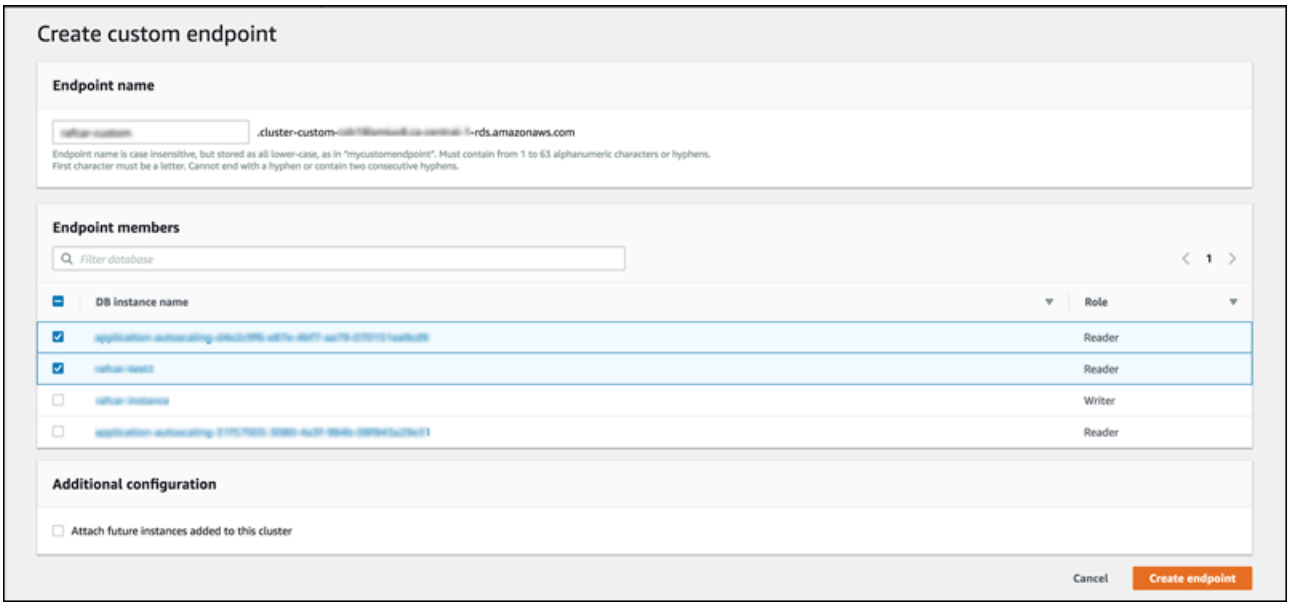
如果一个 Aurora 副本变得不可用，它仍将与任何自定义终端节点关联。例如，当它处于不正常、已停止、重新启动等状态时，它仍然是自定义终端节点的一部分。但是，您无法通过这些终端节点连接到它，直到它再次可用。

创建自定义终端节点

使用 AWS Management Console、AWS CLI 或 Amazon RDS API 创建自定义端点。

控制台

要使用 AWS Management Console创建自定义终端节点，请转到集群详细信息页，然后选择 Endpoints (终端节点) 部分中的 `Create custom endpoint` 操作。为自定义终端节点选择一个名称，该名称对于用户 ID 和区域是唯一的。要选择即使在集群扩展时也保持不变的数据库实例列表，请清除 Attach future instances added to this cluster (挂载以后添加到此集群的实例) 复选框。如果选中该复选框，在将任何新实例添加到集群时，自定义终端节点将动态添加这些实例。



无法在 AWS Management Console中选择自定义端点类型。通过 AWS Management Console创建的所有自定义端点都具有类型 ANY。

AWS CLI

要使用 AWS CLI 创建自定义终端节点，请运行 create-db-cluster-endpoint 命令。

以下命令创建附加到特定集群的自定义终端节点。最初，该终端节点与集群中的所有 Aurora 副本实例相关联。后续命令将其与集群中的一组特定数据库实例相关联。

对于 Linux、macOS 或 Unix：

```
aws rds create-db-cluster-endpoint --db-cluster-endpoint-identifier custom-endpoint-
doc-sample \
```

```
  --endpoint-type reader \
  --db-cluster-identifier cluster_id

aws rds modify-db-cluster-endpoint --db-cluster-endpoint-identifier custom-endpoint-
doc-sample \
  --static-members instance_name_1 instance_name_2
```

对于 Windows：

```
aws rds create-db-cluster-endpoint --db-cluster-endpoint-identifier custom-endpoint-
doc-sample ^
  --endpoint-type reader ^
  --db-cluster-identifier cluster_id

aws rds modify-db-cluster-endpoint --db-cluster-endpoint-identifier custom-endpoint-
doc-sample ^
  --static-members instance_name_1 instance_name_2
```

RDS API

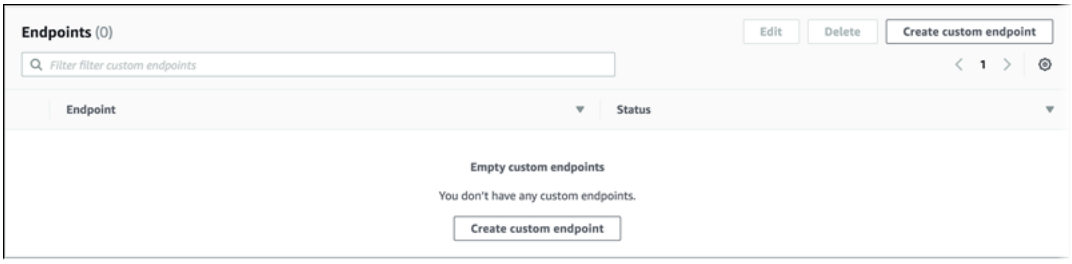
要使用 RDS API 创建自定义终端节点，请运行 CreateDBClusterEndpoint 操作。

查看自定义终端节点

控制台

要使用 AWS Management Console查看自定义终端节点，请转到集群的集群详细信息页，并查看 Endpoints (终端节点) 部分下方的内容。本部分仅包含有关自定义终端节点的信息。主要的 Details (详细信息) 部分中列出了内置终端节点的详细信息。要查看特定自定义终端节点的详细信息，请选择其名称以显示该终端节点的详细信息页。

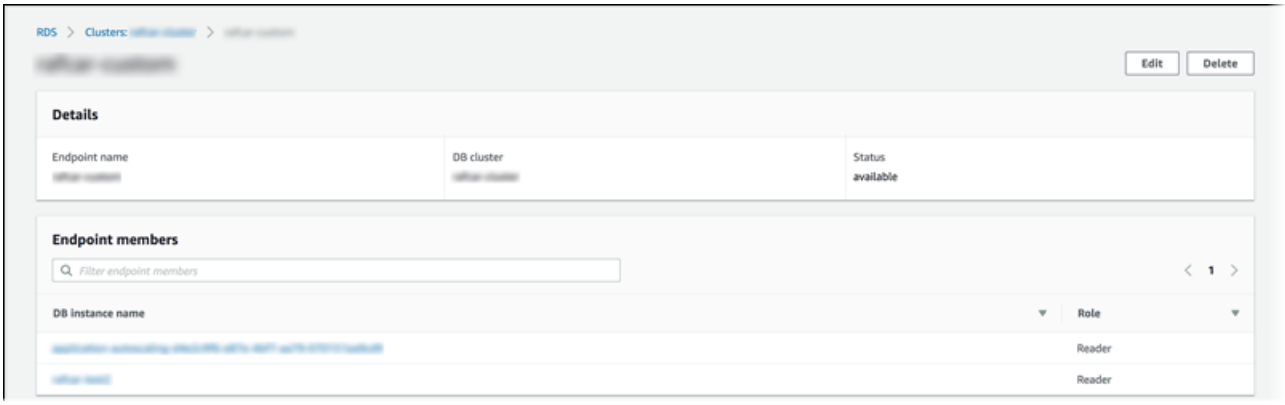
以下屏幕截图说明最初如何清空 Aurora 集群的自定义终端节点列表。



在为该集群创建一些自定义终端节点后，它们将显示在 Endpoints (终端节点) 部分下。



单击详细信息页面可显示终端节点当前所关联的数据库实例。



要查看添加到集群的新数据库实例是否也自动添加到终端节点的其他详细信息，请打开终端节点的 Edit（编辑）页面。

AWS CLI

要使用 AWS CLI 查看自定义终端节点，请运行 describe-db-cluster-endpoints 命令。

以下命令显示与指定区域中的指定集群关联的自定义终端节点。输出包括内置终端节点和任何自定义终端节点。

对于 Linux、macOS 或 Unix：

```
aws rds describe-db-cluster-endpoints --region region_name \
  --db-cluster-identifier cluster_id
```

对于 Windows：

```
aws rds describe-db-cluster-endpoints --region region_name ^
  --db-cluster-identifier cluster_id
```

下面显示了来自 describe-db-cluster-endpoints 命令的一些示例输出。EndpointType 或 WRITER 的 READER 表示集群的内置读/写终端节点和只读终端节点。EndpointType 的 CUSTOM 表示

您创建的终端节点并选择关联的数据库实例。其中一个终端节点具有非空 StaticMembers 字段，表示它与一组精确的数据库实例相关联。另一个终端节点有一个非空 ExcludedMembers 字段，表示终端节点与 ExcludedMembers 下列出的数据库实例之外 的所有数据库实例相关联。当您将新实例添加到集群时，第二种自定义终端节点会增长以容纳这些新实例。

```
{
 "DBClusterEndpoints": [
  {
   "Endpoint": "custom-endpoint-demo.cluster-c7tj4example.ca-
central-1.rds.amazonaws.com",
   "Status": "available",
   "DBClusterIdentifier": "custom-endpoint-demo",
   "EndpointType": "WRITER"
  },
  {
   "Endpoint": "custom-endpoint-demo.cluster-ro-c7tj4example.ca-
central-1.rds.amazonaws.com",
   "Status": "available",
   "DBClusterIdentifier": "custom-endpoint-demo",
   "EndpointType": "READER"
  },
  {
   "CustomEndpointType": "ANY",
   "DBClusterEndpointIdentifier": "powers-of-2",
   "ExcludedMembers": [],
   "DBClusterIdentifier": "custom-endpoint-demo",
   "Status": "available",
   "EndpointType": "CUSTOM",
   "Endpoint": "powers-of-2.cluster-custom-c7tj4example.ca-
central-1.rds.amazonaws.com",
   "StaticMembers": [
     "custom-endpoint-demo-04",
     "custom-endpoint-demo-08",
     "custom-endpoint-demo-01",
     "custom-endpoint-demo-02"
   ],
   "DBClusterEndpointResourceIdentifier": "cluster-endpoint-
W7PE3TLLFNSHXQKFU6J6NV5FHU",
   "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:111122223333:cluster-
endpoint:powers-of-2"
  },
  {
   "CustomEndpointType": "ANY",
```

```
    "DBClusterEndpointIdentifier": "eight-and-higher",
    "ExcludedMembers": [
      "custom-endpoint-demo-04",
      "custom-endpoint-demo-02",
      "custom-endpoint-demo-07",
      "custom-endpoint-demo-05",
      "custom-endpoint-demo-03",
      "custom-endpoint-demo-06",
      "custom-endpoint-demo-01"
    ],
    "DBClusterIdentifier": "custom-endpoint-demo",
    "Status": "available",
    "EndpointType": "CUSTOM",
    "Endpoint": "eight-and-higher.cluster-custom-123456789012.ca-
central-1.rds.amazonaws.com",
    "StaticMembers": [],
    "DBClusterEndpointResourceIdentifier": "cluster-endpoint-
W7PE3TLLFNSHYQKFU6J6NV5FHU",
    "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:111122223333:cluster-
endpoint:eight-and-higher"
  }
 ]
}
```

RDS API

要使用 RDS API 查看自定义终端节点，请运行 DescribeDBClusterEndpoints.html 操作。

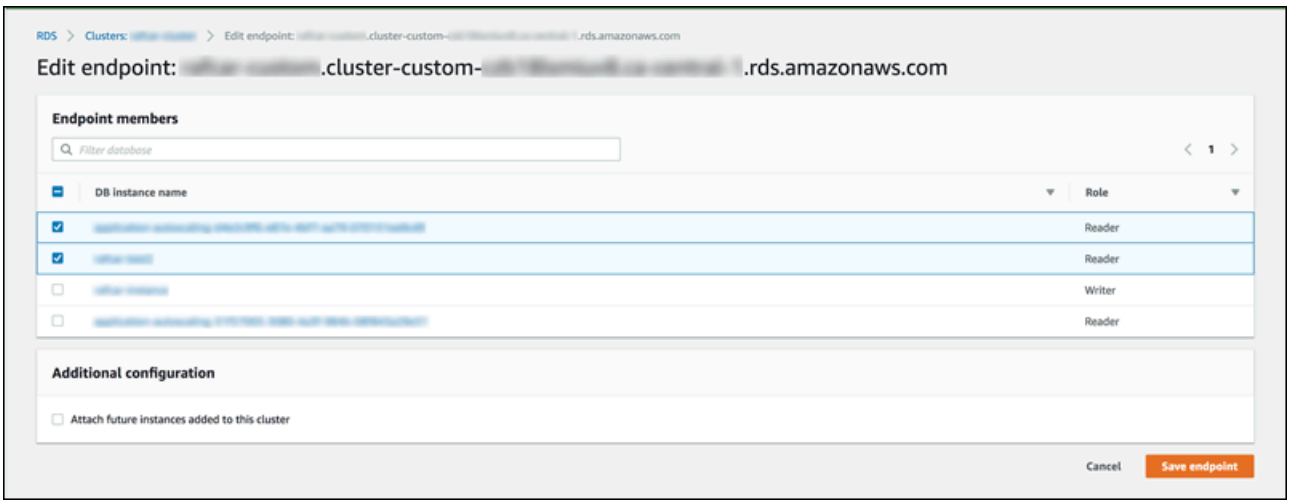
编辑自定义端点

您可以编辑自定义终端节点的属性以更改与终端节点关联的数据库实例。您也可以更改静态列表和排除列表之间的终端节点。如果您需要了解有关这些终端节点属性的更多详细信息，请参阅自定义终端节点的成员资格规则。

当编辑操作的更改正在进行时，您可以继续连接和使用自定义终端节点。

控制台

要使用 AWS Management Console编辑自定义终端节点，您可以在集群详细信息页面上选择终端节点，或显示终端节点的详细信息页面，然后选择 Edit (编辑) 操作。



AWS CLI

要使用 AWS CLI 编辑自定义终端节点，请运行 modify-db-cluster-endpoint 命令。

以下命令更改应用于自定义终端节点的数据库实例集，并可选择在静态列表或排除列表的行为之间切换。`--static-members` 和 `--excluded-members` 参数接受以空格分隔的数据库实例标识符列表。

对于 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster-endpoint --db-cluster-endpoint-identifier my-custom-endpoint
 \
  --static-members db-instance-id-1 db-instance-id-2 db-instance-id-3 \
  --region region_name

aws rds modify-db-cluster-endpoint --db-cluster-endpoint-identifier my-custom-endpoint
 \
  --excluded-members db-instance-id-4 db-instance-id-5 \
  --region region_name
```

对于 Windows：

```
aws rds modify-db-cluster-endpoint --db-cluster-endpoint-identifier my-custom-endpoint
 ^
  --static-members db-instance-id-1 db-instance-id-2 db-instance-id-3 ^
  --region region_name

aws rds modify-db-cluster-endpoint --db-cluster-endpoint-identifier my-custom-endpoint
 ^
```

```
  --excluded-members db-instance-id-4 db-instance-id-5 ^
  --region region_name
```

RDS API

要使用 RDS API 编辑自定义终端节点，请运行 ModifyDBClusterEndpoint.html 操作。

删除自定义终端节点

使用 AWS Management Console、AWS CLI 或 Amazon RDS API 删除自定义端点。

控制台

要使用 AWS Management Console删除自定义终端节点，请转到集群详细信息页面，选择相应的自定义终端节点，然后选择 Delete (删除) 操作。



AWS CLI

要使用 AWS CLI 删除自定义终端节点，请运行 delete-db-cluster-endpoint 命令。

以下命令删除自定义终端节点。您无需指定关联的集群，但必须指定区域。

对于 Linux、macOS 或 Unix：

```
aws rds delete-db-cluster-endpoint --db-cluster-endpoint-identifier custom-end-point-id
 \
  --region region_name
```

对于 Windows：

```
aws rds delete-db-cluster-endpoint --db-cluster-endpoint-identifier custom-end-point-id
 ^
  --region region_name
```

RDS API

要使用 RDS API 删除自定义终端节点，请运行 DeleteDBClusterEndpoint 操作。

Amazon Aurora 的自定义端点的 AWS CLI 示例

以下教程使用带 Unix Shell 语法的 AWS CLI 示例，来说明您如何可以定义一个具有多个“小型”数据库实例和几个“大型”数据库实例的集群，并创建自定义端点来连接到每组数据库实例。要在您自己的系统上运行类似的命令，您应该充分熟悉使用 Aurora 集群和 AWS CLI 的基础知识，以便为区域、子网组和 VPC 安全组等参数提供您自己的值。

此示例演示初始设置步骤：创建一个 Aurora 集群并向其添加数据库实例。这是一个异构集群，意味着并非所有数据库实例都具有相同的容量。大多数实例使用 AWS 实例类 `db.r4.4xlarge`，但最后两个数据库实例使用 `db.r4.16xlarge`。这些示例 `create-db-instance` 命令中的每个命令都将其输出打印到屏幕，并将 JSON 的副本保存在文件中以供以后检查。

```
aws rds create-db-cluster --db-cluster-identifier custom-endpoint-demo --engine aurora-
mysql \
     --engine-version 8.0.mysql_aurora.3.04.0 --master-username $MASTER_USER --manage-
master-user-password \
     --db-subnet-group-name $SUBNET_GROUP  --vpc-security-group-ids $VPC_SECURITY_GROUP
 \
     --region $REGION

for i in 01 02 03 04 05 06 07 08
do
  aws rds create-db-instance --db-instance-identifier custom-endpoint-demo-${i} \
     --engine aurora --db-cluster-identifier custom-endpoint-demo --db-instance-class
 db.r4.4xlarge \
     --region $REGION \
     | tee custom-endpoint-demo-${i}.json
done

for i in 09 10
do
  aws rds create-db-instance --db-instance-identifier custom-endpoint-demo-${i} \
     --engine aurora --db-cluster-identifier custom-endpoint-demo --db-instance-class
 db.r4.16xlarge \
     --region $REGION \
     | tee custom-endpoint-demo-${i}.json
done
```

较大的实例将保留以用于特殊类型的报告查询。为了使它们不太可能被提升为主实例，以下示例将其提升层更改为最低优先级。此示例指定了生成主用户密码并在 Secrets Manager 中对其进行管理的 `--manage-master-user-password` 选项。有关更多信息，请参阅 使用 Amazon Aurora 和 AWS Secrets Manager 管理密码。或者，您可以使用 `--master-password` 选项自行指定和管理密码。

```
for i in 09 10
do
  aws rds modify-db-instance --db-instance-identifier custom-endpoint-demo-${i} \
    --region $REGION --promotion-tier 15
done
```

假设您只想为资源最密集的查询使用两个“较大”实例。为此，您可以首先创建一个自定义的只读终端节点。然后，您可以添加静态成员列表，以便该终端节点仅连接到这些数据库实例。这些实例已处于最低的提升层，因此它们都不太可能被提升到主实例。如果其中一个被提升到主实例，它将无法通过该终端节点访问，因为我们指定了 READER 类型，而不是 ANY 类型。

以下示例演示了创建和修改终端节点命令，并演示了示例 JSON 输出，其中显示了自定义终端节点的初始状态和已修改状态。

```
$ aws rds create-db-cluster-endpoint --region $REGION \
    --db-cluster-identifier custom-endpoint-demo \
    --db-cluster-endpoint-identifier big-instances --endpoint-type reader
{
    "EndpointType": "CUSTOM",
    "Endpoint": "big-instances.cluster-custom-c7tj4example.ca-
central-1.rds.amazonaws.com",
    "DBClusterEndpointIdentifier": "big-instances",
    "DBClusterIdentifier": "custom-endpoint-demo",
    "StaticMembers": [],
    "DBClusterEndpointResourceIdentifier": "cluster-endpoint-
W7PE3TLLFNSHXQKFU6J6NV5FHU",
    "ExcludedMembers": [],
    "CustomEndpointType": "READER",
    "Status": "creating",
    "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:111122223333:cluster-
endpoint:big-instances"
}

$ aws rds modify-db-cluster-endpoint --db-cluster-endpoint-identifier big-instances \
    --static-members custom-endpoint-demo-09 custom-endpoint-demo-10 --region $REGION
{
    "EndpointType": "CUSTOM",
```

```
    "ExcludedMembers": [],
    "DBClusterEndpointIdentifier": "big-instances",
    "DBClusterEndpointResourceIdentifier": "cluster-endpoint-
W7PE3TLLFNSHXQKFU6J6NV5FHU",
    "CustomEndpointType": "READER",
    "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:111122223333:cluster-
endpoint:big-instances",
    "StaticMembers": [
        "custom-endpoint-demo-10",
        "custom-endpoint-demo-09"
    ],
    "Status": "modifying",
    "Endpoint": "big-instances.cluster-custom-c7tj4example.ca-
central-1.rds.amazonaws.com",
    "DBClusterIdentifier": "custom-endpoint-demo"
}
```

集群的默认 READER 终端节点可以连接到“小型”或“大型”数据库实例，这使得在集群繁忙时预测查询性能和可扩展性变得不切实际。为了在数据库实例集之间清晰地划分工作负载，您可以忽略默认的 READER 终端节点，并创建第二个自定义终端节点，该终端节点连接到所有其他数据库实例。以下示例通过创建自定义终端节点，然后添加排除列表来实现这一点。稍后添加到集群的任何其他数据库实例将自动添加到此终端节点。ANY 类型表示此终端节点总共与 8 个实例相关联：主实例和另外 7 个 Aurora 副本。如果示例使用的是 READER 类型，则自定义终端节点仅与 7 个 Aurora 副本相关联。

```
$ aws rds create-db-cluster-endpoint --region $REGION --db-cluster-identifier custom-
endpoint-demo \
    --db-cluster-endpoint-identifier small-instances --endpoint-type any
{
    "Status": "creating",
    "DBClusterEndpointIdentifier": "small-instances",
    "CustomEndpointType": "ANY",
    "EndpointType": "CUSTOM",
    "Endpoint": "small-instances.cluster-custom-c7tj4example.ca-
central-1.rds.amazonaws.com",
    "StaticMembers": [],
    "ExcludedMembers": [],
    "DBClusterIdentifier": "custom-endpoint-demo",
    "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:111122223333:cluster-
endpoint:small-instances",
    "DBClusterEndpointResourceIdentifier": "cluster-
endpoint-6RDDXQOC3AKKZT2PRD7ST37BMY"
}
```

```
$ aws rds modify-db-cluster-endpoint --db-cluster-endpoint-identifier small-instances \
    --excluded-members custom-endpoint-demo-09 custom-endpoint-demo-10 --region $REGION
{
    "DBClusterEndpointIdentifier": "small-instances",
    "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:c7tj4example:cluster-
endpoint:small-instances",
    "DBClusterEndpointResourceIdentifier": "cluster-
endpoint-6RDDXQOC3AKKZT2PRD7ST37BMY",
    "CustomEndpointType": "ANY",
    "Endpoint": "small-instances.cluster-custom-c7tj4example.ca-
central-1.rds.amazonaws.com",
    "EndpointType": "CUSTOM",
    "ExcludedMembers": [
        "custom-endpoint-demo-09",
        "custom-endpoint-demo-10"
    ],
    "StaticMembers": [],
    "DBClusterIdentifier": "custom-endpoint-demo",
    "Status": "modifying"
}
```

以下示例检查此集群的终端节点的状态。集群仍具有其原始集群终端节点（EndPointType 为 WRITER），您仍将它用于管理、ETL 和其他写入操作。它仍具有其原始 READER 终端节点，您不会使用此终端节点，因为它的每个连接都可能被定向到“小型”或“大型”数据库实例。自定义终端节点使得此行为是可预测的，连接保证使用基于您指定的终端节点的“小型”或“大型”数据库实例之一。

```
$ aws rds describe-db-cluster-endpoints --region $REGION
{
    "DBClusterEndpoints": [
        {
            "EndpointType": "WRITER",
            "Endpoint": "custom-endpoint-demo.cluster-c7tj4example.ca-
central-1.rds.amazonaws.com",
            "Status": "available",
            "DBClusterIdentifier": "custom-endpoint-demo"
        },
        {
            "EndpointType": "READER",
            "Endpoint": "custom-endpoint-demo.cluster-ro-c7tj4example.ca-
central-1.rds.amazonaws.com",
            "Status": "available",
            "DBClusterIdentifier": "custom-endpoint-demo"
```

```
        },
        {
            "Endpoint": "small-instances.cluster-custom-c7tj4example.ca-
central-1.rds.amazonaws.com",
            "CustomEndpointType": "ANY",
            "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:111122223333:cluster-
endpoint:small-instances",
            "ExcludedMembers": [
                "custom-endpoint-demo-09",
                "custom-endpoint-demo-10"
            ],
            "DBClusterEndpointResourceIdentifier": "cluster-
endpoint-6RDDXQOC3AKKZT2PRD7ST37BMY",
            "DBClusterIdentifier": "custom-endpoint-demo",
            "StaticMembers": [],
            "EndpointType": "CUSTOM",
            "DBClusterEndpointIdentifier": "small-instances",
            "Status": "modifying"
        },
        {
            "Endpoint": "big-instances.cluster-custom-c7tj4example.ca-
central-1.rds.amazonaws.com",
            "CustomEndpointType": "READER",
            "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:111122223333:cluster-
endpoint:big-instances",
            "ExcludedMembers": [],
            "DBClusterEndpointResourceIdentifier": "cluster-endpoint-
W7PE3TLLFNSHXQKFU6J6NV5FHU",
            "DBClusterIdentifier": "custom-endpoint-demo",
            "StaticMembers": [
                "custom-endpoint-demo-10",
                "custom-endpoint-demo-09"
            ],
            "EndpointType": "CUSTOM",
            "DBClusterEndpointIdentifier": "big-instances",
            "Status": "available"
        }
    ]
}
```

最后的示例演示了与自定义终端节点的连续数据库连接如何连接到 Aurora 集群中的各种数据库实例。small-instances 终端节点始终连接到 db.r4.4xlarge 数据库实例，这些实例是此集群中编号较低的主机。

```
$ mysql -h small-instances.cluster-custom-c7tj4example.ca-central-1.rds.amazonaws.com -
u $MYUSER -p
mysql> select @@aurora_server_id;
+--------------------------+
| @@aurora_server_id       |
+--------------------------+
| custom-endpoint-demo-02 |
+--------------------------+

$ mysql -h small-instances.cluster-custom-c7tj4example.ca-central-1.rds.amazonaws.com -
u $MYUSER -p
mysql> select @@aurora_server_id;
+--------------------------+
| @@aurora_server_id       |
+--------------------------+
| custom-endpoint-demo-07 |
+--------------------------+

$ mysql -h small-instances.cluster-custom-c7tj4example.ca-central-1.rds.amazonaws.com -
u $MYUSER -p
mysql> select @@aurora_server_id;
+--------------------------+
| @@aurora_server_id       |
+--------------------------+
| custom-endpoint-demo-01 |
+--------------------------+
```

big-instances 终端节点始终连接到 db.r4.16xlarge 数据库实例，这些实例是此集群中编号最高的两个主机。

```
$ mysql -h big-instances.cluster-custom-c7tj4example.ca-central-1.rds.amazonaws.com -u
 $MYUSER -p
mysql> select @@aurora_server_id;
+--------------------------+
| @@aurora_server_id       |
+--------------------------+
| custom-endpoint-demo-10 |
+--------------------------+

$ mysql -h big-instances.cluster-custom-c7tj4example.ca-central-1.rds.amazonaws.com -u
 $MYUSER -p
mysql> select @@aurora_server_id;
+--------------------------+
```

```
| @@aurora_server_id      |
+-------------------------+
| custom-endpoint-demo-09 |
+-------------------------+
```

Amazon Aurora 数据库实例类

数据库实例类确定 Amazon Aurora 数据库实例的计算和内存容量。您需要的数据库实例类取决于您的处理能力和内存要求。

数据库实例类由数据库实例类类型和大小共同组成。例如，db.r6g 是由 AWS Graviton2 处理器提供支持的内存优化数据库实例类类型。在 db.r6g 实例类类型中，db.r6g.2xlarge 是数据库实例类。该类的大小为 2xlarge。

有关实例类定价的更多信息，请参阅 Amazon RDS 定价。

有关数据库实例类类型、支持的数据库引擎、支持的 AWS 区域或数据库实例类的硬件规格的更多信息，请参阅以下各节。

主题

- 数据库实例类类型
- 数据库实例类支持的数据库引擎
- 确定 AWS 区域 中的数据库实例类支持
- 适用于 Aurora 的数据库实例类的硬件规格

数据库实例类类型

Amazon Aurora 对于以下使用案例支持数据库实例类：

- Aurora Serverless v2
- 内存优化
- 可突增性能
- 优化型读取

有关 Amazon EC2 实例类型的更多信息，请参阅 Amazon EC2 文档中的实例类型。

Aurora Serverless v2 实例类类型

提供了以下 Aurora Serverless v2 类型：

- db.serverless – Aurora Serverless v2 使用的特殊数据库实例类类型。Aurora 会随着工作负载的变化动态调整计算、内存和网络资源。有关使用情况的详细信息，请参阅 使用 Aurora Serverless v2。

内存优化型实例类类型

内存优化型 X 系列支持以下实例类：

- db.x2g - 针对内存密集型应用程序进行优化的实例类，由 AWS Graviton2 处理器提供支持。这些实例类可提供较低的每 GiB 内存成本。

  您可以修改数据库实例，以使用由 AWS Graviton2 处理器提供支持的其中一个数据库实例类。为此，请完成与修改任何其他数据库实例相同的步骤。

内存优化型 R 系列支持以下实例类类型：

- db.r8g – 由 AWS Graviton4 处理器提供支持的实例类。这些实例类非常适合运行内存密集型工作负载。与基于 AWS Graviton3 的第七代 db.r7g 实例相比，这些实例提供了更大的实例大小，其 vCPU 和内存最多可增加 3 倍。

  您可以修改数据库实例，以使用由 AWS Graviton4 处理器提供支持的其中一个数据库实例类。为此，请完成与修改任何其他数据库实例相同的步骤。
- db.r7g – 由 AWS Graviton3 处理器提供支持的实例类。这些实例类非常适合运行内存密集型工作负载。

  您可以修改数据库实例，以使用由 AWS Graviton3 处理器提供支持的其中一个数据库实例类。为此，请完成与修改任何其他数据库实例相同的步骤。
- db.r7i – 由第四代 Intel Xeon 可扩展处理器提供支持的实例类 这些实例类已通过 SAP 认证，非常适合运行内存密集型工作负载。您可以修改数据库实例，以使用由第四代 Intel Xeon 可扩展处理器提供支持的其中一个数据库实例类。为此，请完成与修改任何其他数据库实例相同的步骤。
- db.r6g – 由 AWS Graviton2 处理器提供支持的实例类。这些实例类非常适合

  您可以修改数据库实例，以使用由 AWS Graviton2 处理器提供支持的其中一个数据库实例类。为此，请完成与修改任何其他数据库实例相同的步骤。
- db.r6i – 由第三代 Intel Xeon 可扩展处理器提供支持的实例类 这些实例类已通过 SAP 认证，非常适合运行内存密集型工作负载。
- db.r5 – 针对内存密集型应用程序进行了优化的实例类。这些实例类提供了改进的联网和 Amazon Elastic Block Store（Amazon EBS）性能。它们由 AWS Nitro 系统（专用硬件和轻量级虚拟机监控程序的组合）提供支持。
- db.r4：仅 Aurora MySQL 2.x 以及 Aurora PostgreSQL 11 和 12 版本支持这些实例类。对于所有使用 db.r4 数据库实例类的 Aurora 数据库集群，建议您尽快升级到更高代次的数据库实例类。

db.r4 实例类不适用于 Aurora I/O-Optimized 集群数据库配置。

可突增性能实例类类型

提供了以下可突增性能数据库实例类类型：

- db.t4g - 由基于 Arm 的 AWS Graviton2 处理器提供支持的通用实例类。与之前的可突增性能数据库实例类相比，这些实例类提供了更好的性价比，适用于广泛的可突增通用工作负载。Amazon RDS db.t4g 实例配置为无限模式。这表示实例可以在 24 小时时段内突增到基准以上，但需额外付费。

  您可以修改数据库实例，以使用由 AWS Graviton2 处理器提供支持的其中一个数据库实例类。为此，请完成与修改任何其他数据库实例相同的步骤。
- db.t3 – 提供基准性能水平的实例类，并且可以突增到完全 CPU 使用率。db.t3 实例配置为无限模式。这些实例类提供比以前的 db.t2 实例类更多的计算容量。它们由 AWS Nitro 系统（专用硬件和轻量级管理程序的组合）提供支持。建议仅将这些实例类用于开发和测试服务器，或其他非生产服务器。
- db.t2 – 提供基准性能水平的实例类，并且可以突增到完全 CPU 使用率。db.t2 实例配置为无限模式。建议仅将这些实例类用于开发和测试服务器，或其他非生产服务器。

  db.t2 实例类不适用于 Aurora I/O-Optimized 集群数据库配置。

> **Note**
>
> 我们建议仅将 T 数据库实例类用于开发服务器、测试服务器或其他非生产服务器。有关 T 实例类的更详细建议，请参阅使用 T 实例类进行开发和测试。

对于数据库实例类硬件规格，请参阅适用于 Aurora 的数据库实例类的硬件规格。

优化型读取实例类类型

以下是可用的优化型读取实例类类型：

- db.r6gd – 由 AWS Graviton2 处理器提供支持的实例类。这些实例类非常适合运行内存密集型工作负载，并为需要高速、低延迟本地存储的应用程序提供基于本地 NVMe 的 SSD 块级存储。

- db.r6id – 由第三代 Intel Xeon 可扩展处理器提供支持的实例类。这些实例类已通过 SAP 认证，非常适合内存密集型工作负载。它们提供最多 1 TiB 的内存和高达 7.6 TB 基于 NVMe 的直连 SSD 存储。

数据库实例类支持的数据库引擎

下表显示了 Amazon Aurora 数据库引擎支持的数据库实例类。

db.serverless – 能够自动扩缩容量的 Aurora Serverless v2 实例类

实例类	Aurora MySQL	Aurora PostgreSQL
db.serverless	请参阅 支持 Aurora Serverless v2 的区域和 Aurora 数据库引擎。	请参阅 支持 Aurora Serverless v2 的区域和 Aurora 数据库引擎。

db.x2g – 由 AWS Graviton2 处理器提供支持的内存优化型实例类

实例类	Aurora MySQL	Aurora PostgreSQL
db.x2g.16xlarge	所有当前可用的版本	15.2 及更高版本、14.3 及更高版本、13.3 及更高版本、12.8 及更高版本、11.9、11.12 及更高版本
db.x2g.12xlarge	所有当前可用的版本	15.2 及更高版本、14.3 及更高版本、13.3 及更高版本、12.8 及更高版本、11.9、11.12 及更高版本
db.x2g.8xlarge	所有当前可用的版本	15.2 及更高版本、14.3 及更高版本、13.3 及更高版本、12.8 及更高版本、11.9、11.12 及更高版本
db.x2g.4xlarge	所有当前可用的版本	15.2 及更高版本、14.3 及更高版本、13.3 及更高版本、12.8 及更高版本、11.9、11.12 及更高版本

实例类	Aurora MySQL	Aurora PostgreSQL
db.x2g.2xlarge	所有当前可用的版本	15.2 及更高版本、14.3 及更高版本、13.3 及更高版本、12.8 及更高版本、11.9、11.12 及更高版本
db.x2g.xlarge	所有当前可用的版本	15.2 及更高版本、14.3 及更高版本、13.3 及更高版本、12.8 及更高版本、11.9、11.12 及更高版本
db.x2g.large	所有当前可用的版本	15.2 及更高版本、14.3 及更高版本、13.3 及更高版本、12.8 及更高版本、11.9、11.12 及更高版本

db.r6gd – 由 AWS Graviton2 处理器提供支持的优化型读取实例类

实例类	Aurora MySQL	Aurora PostgreSQL
db.r6gd.16xlarge	否	16.1 及更高版本，15.4 及更高版本，14.9 及更高版本
db.r6gd.12xlarge	否	16.1 及更高版本，15.4 及更高版本，14.9 及更高版本
db.r6gd.8xlarge	否	16.1 及更高版本，15.4 及更高版本，14.9 及更高版本
db.r6gd.4xlarge	否	16.1 及更高版本，15.4 及更高版本，14.9 及更高版本
db.r6gd.2xlarge	否	16.1 及更高版本，15.4 及更高版本，14.9 及更高版本
db.r6gd.xlarge	否	16.1 及更高版本，15.4 及更高版本，14.9 及更高版本

db.r6id – 优化型读取实例类

实例类	Aurora MySQL	Aurora PostgreSQL
db.r6id.32xlarge	否	16.1 及更高版本，15.4 及更高版本，14.9 及更高版本
db.r6id.24xlarge	否	16.1 及更高版本，15.4 及更高版本，14.9 及更高版本

db.r7g – 由 AWS Graviton3 处理器提供支持的内存优化型实例类

实例类	Aurora MySQL	Aurora PostgreSQL
db.r7g.16xlarge	2.12.0 及更高版本、3.03.1 及更高版本	15.2 及更高版本，14.7 及更高版本，13.10 及更高版本
db.r7g.12xlarge	2.12.0 及更高版本、3.03.1 及更高版本	15.2 及更高版本，14.7 及更高版本，13.10 及更高版本
db.r7g.8xlarge	2.12.0 及更高版本、3.03.1 及更高版本	15.2 及更高版本，14.7 及更高版本，13.10 及更高版本
db.r7g.4xlarge	2.12.0 及更高版本、3.03.1 及更高版本	15.2 及更高版本，14.7 及更高版本，13.10 及更高版本
db.r7g.2xlarge	2.12.0 及更高版本、3.03.1 及更高版本	15.2 及更高版本，14.7 及更高版本，13.10 及更高版本
db.r7g.xlarge	2.12.0 及更高版本、3.03.1 及更高版本	15.2 及更高版本，14.7 及更高版本，13.10 及更高版本
db.r7g.large	2.12.0 及更高版本、3.03.1 及更高版本	15.2 及更高版本，14.7 及更高版本，13.10 及更高版本

db.r6g – 由 AWS Graviton2 处理器提供支持的内存优化型实例类

实例类	Aurora MySQL	Aurora PostgreSQL
db.r6g.16xlarge	所有当前可用的版本	15.2 及更高版本、14.3 及更高版本、13.3 及更高版本、12.8 及更高版本、11.9、11.12 及更高版本
db.r6g.12xlarge	所有当前可用的版本	15.2 及更高版本、14.3 及更高版本、13.3 及更高版本、12.8 及更高版本、11.9、11.12 及更高版本
db.r6g.8xlarge	所有当前可用的版本	15.2 及更高版本、14.3 及更高版本、13.3 及更高版本、12.8 及更高版本、11.9、11.12 及更高版本
db.r6g.4xlarge	所有当前可用的版本	15.2 及更高版本、14.3 及更高版本、13.3 及更高版本、12.8 及更高版本、11.9、11.12 及更高版本
db.r6g.2xlarge	所有当前可用的版本	15.2 及更高版本、14.3 及更高版本、13.3 及更高版本、12.8 及更高版本、11.9、11.12 及更高版本
db.r6g.xlarge	所有当前可用的版本	15.2 及更高版本、14.3 及更高版本、13.3 及更高版本、12.8 及更高版本、11.9、11.12 及更高版本
db.r6g.large	所有当前可用的版本	15.2 及更高版本、14.3 及更高版本、13.3 及更高版本、12.8 及更高版本、11.9、11.12 及更高版本

db.r6i – 内存优化型实例类

实例类	Aurora MySQL	Aurora PostgreSQL
db.r6i.32xlarge	所有当前可用的版本	15.2 及更高版本、14.3 及更高版本、13.5 及更高版本、12.9 及更高版本
db.r6i.24xlarge	所有当前可用的版本	15.2 及更高版本、14.3 及更高版本、13.5 及更高版本、12.9 及更高版本
db.r6i.16xlarge	所有当前可用的版本	15.2 及更高版本、14.3 及更高版本、13.5 及更高版本、12.9 及更高版本
db.r6i.12xlarge	所有当前可用的版本	15.2 及更高版本、14.3 及更高版本、13.5 及更高版本、12.9 及更高版本
db.r6i.8xlarge	所有当前可用的版本	15.2 及更高版本、14.3 及更高版本、13.5 及更高版本、12.9 及更高版本
db.r6i.4xlarge	所有当前可用的版本	15.2 及更高版本、14.3 及更高版本、13.5 及更高版本、12.9 及更高版本
db.r6i.2xlarge	所有当前可用的版本	15.2 及更高版本、14.3 及更高版本、13.5 及更高版本、12.9 及更高版本
db.r6i.xlarge	所有当前可用的版本	15.2 及更高版本、14.3 及更高版本、13.5 及更高版本、12.9 及更高版本
db.r6i.large	所有当前可用的版本	15.2 及更高版本、14.3 及更高版本、13.5 及更高版本、12.9 及更高版本

db.r5 – 内存优化型实例类

实例类	Aurora MySQL	Aurora PostgreSQL
db.r5.24xlarge	所有当前可用的版本	所有当前可用的版本
db.r5.16xlarge	所有当前可用的版本	所有当前可用的版本
db.r5.12xlarge	所有当前可用的版本	所有当前可用的版本
db.r5.8xlarge	所有当前可用的版本	所有当前可用的版本
db.r5.4xlarge	所有当前可用的版本	所有当前可用的版本
db.r5.2xlarge	所有当前可用的版本	所有当前可用的版本
db.r5.xlarge	所有当前可用的版本	所有当前可用的版本
db.r5.large	所有当前可用的版本	所有当前可用的版本

db.r4 – 内存优化型实例类

实例类	Aurora MySQL	Aurora PostgreSQL
db.r4.16xlarge	所有 2.x 版本；3.x 版本不支持	否
db.r4.8xlarge	所有 2.x 版本；3.x 版本不支持	否
db.r4.4xlarge	所有 2.x 版本；3.x 版本不支持	否
db.r4.2xlarge	所有 2.x 版本；3.x 版本不支持	否
db.r4.xlarge	所有 2.x 版本；3.x 版本不支持	否
db.r4.large	所有 2.x 版本；3.x 版本不支持	否

db.t4g - 由 AWS Graviton2 处理器提供支持的可突增性能实例类

实例类	Aurora MySQL	Aurora PostgreSQL
db.r6g.2xlarge	否	否
db.t4g.xlarge	否	否
db.t4g.large	所有当前可用的版本	15.2 及更高版本、14.3 及更高版本、13.3 及更高版本、12.7 及更高版本、11.12 及更高版本
db.t4g.medium	所有当前可用的版本	15.2 及更高版本、14.3 及更高版本、13.3 及更高版本、12.7 及更高版本、11.12 及更高版本
db.t4g.small	否	否

db.t3 – 具爆发能力的实例类

实例类	Aurora MySQL	Aurora PostgreSQL
db.t3.2xlarge	否	否
db.t3.xlarge	否	否
db.t3.large	所有当前可用的版本	15.2 及更高版本、14.3 及更高版本、13.3 及更高版本、12.7 及更高版本、11.12 及更高版本
db.t3.medium	所有当前可用的版本	15.2 及更高版本、14.3 及更高版本、13.3 及更高版本、12.7 及更高版本、11.12 及更高版本
db.t3.small	所有 2.x 版本；3.x 版本不支持	否
db.t3.micro	否	否

db.t2 – 具爆发能力的实例类

实例类	Aurora MySQL	Aurora PostgreSQL
db.t2.medium	所有 2.x 版本；3.x 版本不支持	否
db.t2.small	所有 2.x 版本；3.x 版本不支持	否

确定 AWS 区域 中的数据库实例类支持

要确定特定 AWS 区域中每个数据库引擎支持的数据库实例类，您可以采用以下几种方法之一。您可以使用 AWS Management Console、Amazon RDS 定价页面或 AWS CLI 命令 describe-orderable-db-instance-options。

> **Note**
>
> 当您使用 AWS Management Console 执行操作时，它会自动显示特定数据库引擎、数据库引擎版本和 AWS 区域 支持的数据库实例类。您可以执行的操作示例包括创建和修改数据库实例。

目录

- 使用 Amazon RDS 定价页面确定 AWS 区域 中的数据库实例类支持
- 使用 AWS CLI 确定 AWS 区域 中的数据库实例类支持
  - 列出 AWS 区域 中特定数据库引擎版本支持的数据库实例类
  - 列出支持 AWS 区域 中特定数据库实例类的数据库引擎版本

使用 Amazon RDS 定价页面确定 AWS 区域 中的数据库实例类支持

您可以使用 Amazon Aurora 定价页面来确定特定 AWS 区域中每个数据库引擎支持的数据库实例类。

使用定价页面确定区域中每个引擎支持的数据库实例类

1. 转至 Amazon Aurora 定价。
2. 在 AWS 定价计算器部分中选择 Amazon Aurora 引擎。
3. 在选择区域中，选择一个 AWS 区域。

4. 在集群配置选项中，选择一个配置选项。
5. 使用兼容实例部分查看支持的数据库实例类。
6. （可选）在计算器中选择其它选项，然后选择保存并查看摘要或保存并添加服务。

使用 AWS CLI 确定 AWS 区域 中的数据库实例类支持

您可以使用 AWS CLI 来确定 AWS 区域 中的特定数据库引擎和数据库引擎版本支持哪些数据库实例类。

要使用下面的 AWS CLI 示例，请为数据库引擎、数据库引擎版本、数据库实例类和 AWS 区域 输入有效值。下表显示了有效的数据库引擎值。

引擎名称	CLI 命令中的引擎值	有关版本的更多信息
MySQL 5.7 兼容和 8.0 兼容的 Aurora	`aurora-mysql`	《Aurora MySQL 发布说明》中的 Amazon Aurora MySQL 版本 2 的数据库引擎更新和 Amazon Aurora MySQL 版本 3 的数据库引擎更新
Aurora PostgreSQL	`aurora-postgresql`	Aurora PostgreSQL 发布说明

有关 AWS 区域 名称的信息，请参阅 AWS 地区。

以下示例演示了如何使用 describe-orderable-db-instance-options AWS CLI 命令确定 AWS 区域 中的数据库实例类支持。

主题

- 列出 AWS 区域 中特定数据库引擎版本支持的数据库实例类
- 列出支持 AWS 区域 中特定数据库实例类的数据库引擎版本

列出 AWS 区域 中特定数据库引擎版本支持的数据库实例类

要列出 AWS 区域 中特定数据库引擎版本支持的数据库实例类，请运行以下命令。

对于 Linux、macOS 或 Unix：

```
aws rds describe-orderable-db-instance-options --engine engine --engine-version version
 \
    --query "OrderableDBInstanceOptions[].
{DBInstanceClass:DBInstanceClass,SupportedEngineModes:SupportedEngineModes[0]}" \
    --output table \
    --region region
```

对于 Windows：

```
aws rds describe-orderable-db-instance-options --engine engine --engine-version version
 ^
    --query "OrderableDBInstanceOptions[].
{DBInstanceClass:DBInstanceClass,SupportedEngineModes:SupportedEngineModes[0]}" ^
    --output table ^
    --region region
```

输出还显示了每个数据库实例类支持的引擎模式。

例如，以下命令列出了美国东部（弗吉尼亚州北部）的 Aurora PostgreSQL 数据库引擎 13.6 版支持的数据库实例类。

对于 Linux、macOS 或 Unix：

```
aws rds describe-orderable-db-instance-options --engine aurora-postgresql --engine-
version 15.3 \
    --query "OrderableDBInstanceOptions[].
{DBInstanceClass:DBInstanceClass,SupportedEngineModes:SupportedEngineModes[0]}" \
    --output table \
    --region us-east-1
```

对于 Windows：

```
aws rds describe-orderable-db-instance-options --engine aurora-postgresql --engine-
version 15.3 ^
    --query "OrderableDBInstanceOptions[].
{DBInstanceClass:DBInstanceClass,SupportedEngineModes:SupportedEngineModes[0]}"  ^
    --output table ^
    --region us-east-1
```

列出支持 AWS 区域 中特定数据库实例类的数据库引擎版本

要列出支持 AWS 区域 中特定数据库实例类的数据库引擎版本，请运行以下命令。

对于 Linux、macOS 或 Unix：

```
aws rds describe-orderable-db-instance-options --engine engine --db-instance-
class DB_instance_class \
    --query "OrderableDBInstanceOptions[].
{EngineVersion:EngineVersion,SupportedEngineModes:SupportedEngineModes[0]}" \
    --output table \
    --region region
```

对于 Windows：

```
aws rds describe-orderable-db-instance-options --engine engine --db-instance-
class DB_instance_class ^
    --query "OrderableDBInstanceOptions[].
{EngineVersion:EngineVersion,SupportedEngineModes:SupportedEngineModes[0]}" ^
    --output table ^
    --region region
```

输出还显示了每个数据库引擎版本支持的引擎模式。

例如，以下命令列出了 US East (N. Virginia) 中支持 db.r5.large 数据库实例类的 Aurora PostgreSQL 数据库引擎的数据库引擎版本。

对于 Linux、macOS 或 Unix：

```
aws rds describe-orderable-db-instance-options --engine aurora-postgresql --db-
instance-class db.r7g.large \
    --query "OrderableDBInstanceOptions[].
{EngineVersion:EngineVersion,SupportedEngineModes:SupportedEngineModes[0]}" \
    --output table \
    --region us-east-1
```

对于 Windows：

```
aws rds describe-orderable-db-instance-options --engine aurora-postgresql --db-
instance-class db.r7g.large ^
    --query "OrderableDBInstanceOptions[].
{EngineVersion:EngineVersion,SupportedEngineModes:SupportedEngineModes[0]}" ^
    --output table ^
    --region us-east-1
```

适用于 Aurora 的数据库实例类的硬件规格

在本节的表中，可以找到有关适用于 Aurora 的 Amazon RDS 数据库实例类的硬件详细信息。

有关每个数据库实例类的 Aurora 数据库引擎支持的信息，请参阅数据库实例类支持的数据库引擎。

主题

- 适用于 Aurora 的数据库实例类的硬件术语
- 内存优化型实例类的硬件规格
- 可突增性能实例类的硬件规格

适用于 Aurora 的数据库实例类的硬件术语

以下术语用于描述数据库实例类的硬件规格。

vCPU

虚拟中央处理器（CPU）的数量。虚拟 CPU 是可用于比较数据库实例类的容量单位。您不再购买或租用特定的处理器并用上数月或数年，而是以小时为单位租用容量。我们的目标是在实际基础硬件的限制内提供特定数量的一致 CPU 容量。

ECU

Amazon EC2 实例的整数处理能力的相对度量。为了便于开发人员比较不同实例类的 CPU 容量，我们定义了一个 Amazon EC2 计算单位。分配给特定实例的 CPU 量是以这些 EC2 计算单位来表示的。一个 ECU 目前提供的 CPU 容量相当于 1.0–1.2 GHz 2007 Opteron 或 2007 Xeon 处理器。

内存 (GiB)

为数据库实例分配的 RAM (GiB)。内存与 vCPU 之间通常具有一致的比率。以 db.r4 实例类为例，其具有类似于 db.r5 实例类的内存 vCPU 比。但是，对于大多数使用案例，db.r5 实例类会提供比 db.r4 实例类更好、更一致的性能。

最大 EBS 带宽（Mbps）

以 MB/s 为单位的最大 EBS 带宽。除以 8 可获得预期吞吐量（MB/s）。

> **Note**
>
> 此数字是指数据库实例内用于本地存储的 I/O 带宽。它不适用于与 Aurora 集群卷之间的通信。

网络带宽

与其他数据库实例类有关的网络速度。

有关使用 Amazon CloudWatch 指标监控 Aurora 数据库实例吞吐量的信息，请参阅 使用 Amazon CloudWatch 指标评估 Aurora MySQL 数据库实例使用情况和 使用 CloudWatch 指标评估 Aurora PostgreSQL 数据库实例使用情况。

内存优化型实例类的硬件规格

下表显示了内存优化型实例类的计算、内存、存储和带宽规格。

db.x2g – 具有 AWS Graviton2 处理器的内存优化型实例类

实例类	vCPU	ECU	内存（GiB）	实例存储（GiB）	最大 EBS 带宽（Mbps）	网络带宽（Gbps）
db.x2g.16xlarge	64	—	1024	仅 EBS 优化	19000	25
db.x2g.12xlarge	48	—	768	仅 EBS 优化	14,250	20
db.x2g.8xlarge	32	—	512	仅 EBS 优化	9500	12
db.x2g.4xlarge	16	—	256	仅 EBS 优化	4750	最多 10
db.x2g.2xlarge	8	—	128	仅 EBS 优化	最多 4,750	最多 10
db.x2g.xlarge	4	—	64	仅 EBS 优化	最多 4,750	最多 10
db.x2g.large	2	—	32	仅 EBS 优化	最多 4,750	最多 10

db.r8g – 由 AWS Graviton4 处理器提供支持的内存优化型实例类

实例类	vCPU	ECU	内存（GiB）	实例存储（GiB）	最大 EBS 带宽（Mbps）	网络带宽（Gbps）
db.r8g.48xlarge	192	—	1536	仅 EBS 优化	40000	50
db.r8g.24xlarge	96	—	768	仅 EBS 优化	30000	40

实例类	vCPU	ECU	内存（GiB）	实例存储（GiB）	最大 EBS 带宽（Mbps）	网络带宽（Gbps)
db.r8g.16xlarge	64	—	512	仅 EBS 优化	20000	30
db.r8g.12xlarge	48	—	384	仅 EBS 优化	15000	22.5
db.r8g.8xlarge	32	—	256	仅 EBS 优化	10000	15
db.r8g.4xlarge	16	—	128	仅 EBS 优化	最高 10,000	最多 15
db.r8g.2xlarge	8	—	64	仅 EBS 优化	最高 10,000	最多 15
db.r8g.xlarge	4	—	32	仅 EBS 优化	最高 10,000	最多 12.5
db.r8g.large	2	—	16	仅 EBS 优化	最高 10,000	最多 12.5

db.r7i – 由第四代 Intel Xeon 可扩展处理器提供支持的内存优化型实例类

实例类	vCPU	ECU	内存（GiB）	实例存储（GiB）	最大 EBS 带宽（Mbps）	网络带宽（Gbps)
db.r7i.48xlarge	192	—	1536	仅 EBS 优化	40000	50
db.r7i.24xlarge	96	—	768	仅 EBS 优化	30000	37.5
db.r7i.16xlarge	64	—	512	仅 EBS 优化	20000	25
db.r7i.12xlarge	48	—	384	仅 EBS 优化	15000	18.75
db.r7i.8xlarge	32	—	256	仅 EBS 优化	10000	12.5
db.r7i.4xlarge	16	—	128	仅 EBS 优化	最高 10,000	最多 12.5
db.r7i.2xlarge	8	—	64	仅 EBS 优化	最高 10,000	最多 12.5
db.r7i.xlarge	4	—	32	仅 EBS 优化	最高 10,000	最多 12.5
db.r7i.large	2	—	16	仅 EBS 优化	最高 10,000	最多 12.5

db.r7g – 具有 AWS Graviton3 处理器的内存优化型实例类

实例类	vCPU	ECU	内存（GiB）	实例存储（GiB）	最大 EBS 带宽（Mbps）	网络带宽（Gbps）
db.r7g.16xlarge	64	—	512	仅 EBS 优化	20000	30
db.r7g.12xlarge	48	—	384	仅 EBS 优化	15000	22.5
db.r7g.8xlarge	32	—	256	仅 EBS 优化	10000	15
db.r7g.4xlarge	16	—	128	仅 EBS 优化	最高 10,000	最多 15
db.r7g.2xlarge	8	—	64	仅 EBS 优化	最高 10,000	最多 15
db.r7g.xlarge	4	—	32	仅 EBS 优化	最高 10,000	最多 12.5
db.r7g.large	2	—	16	仅 EBS 优化	最高 10,000	最多 12.5

db.r6id – 具有第三代 Intel Xeon 可扩展处理器和 SSD 存储的内存优化型实例类

实例类	vCPU	ECU	内存（GiB）	实例存储（GiB）	最大 EBS 带宽（Mbps）	网络带宽（Gbps）
db.r6id.32xlarge	128	—	1024	4x1900 NVMe SSD	40000	50
db.r6id.24xlarge	96	—	768	4x1425 NVMe SSD	30000	37.5

db.r6gd – 具有 AWS Graviton2 处理器和 SSD 存储的内存优化型实例类

实例类	vCPU	ECU	内存（GiB）	实例存储（GiB）	最大 EBS 带宽（Mbps）	网络带宽（Gbps）
db.r6gd.16xlarge	64	—	512	2 x 1900 NVMe SSD	19000	25

实例类	vCPU	ECU	内存（GiB）	实例存储（GiB）	最大 EBS 带宽（Mbps）	网络带宽（Gbps)
db.r6gd.12xlarge	48	—	384	2 x 1425 NVMe SSD	13,500	20
db.r6gd.8xlarge	32	—	256	1 x 1900 NVMe SSD	9,000	12
db.r6gd.4xlarge	16	—	128	1 x 950 NVMe SSD	4750	最多 10
db.r6gd.2xlarge	8	—	64	1 x 474 NVMe SSD	最多 4,750	最多 10
db.r6gd.xlarge	4	—	32	1 x 237 NVMe SSD	最多 4,750	最多 10

db.r6g – 具有 AWS Graviton2 处理器的内存优化型实例类

实例类	vCPU	ECU	内存（GiB）	实例存储（GiB）	最大 EBS 带宽（Mbps）	网络带宽（Gbps)
db.r6g.16xlarge	64	—	512	仅 EBS 优化	19000	25
db.r6g.12xlarge	48	—	384	仅 EBS 优化	13,500	20
db.r6g.8xlarge	32	—	256	仅 EBS 优化	9,000	12
db.r6g.4xlarge	16	—	128	仅 EBS 优化	4750	最多 10
db.r6g.2xlarge	8	—	64	仅 EBS 优化	最多 4,750	最多 10
db.r6g.xlarge	4	—	32	仅 EBS 优化	最多 4,750	最多 10
db.r6g.large	2	—	16	仅 EBS 优化	最多 4,750	最多 10

db.r6i – 具有第三代 Intel Xeon 可扩展处理器的内存优化型实例类

实例类	vCPU	ECU	内存（GiB）	实例存储（GiB）	最大 EBS 带宽（Mbps）	网络带宽（Gbps）
db.r6i.32xlarge	128	—	1024	仅 EBS 优化	40000	50
db.r6i.24xlarge	96	—	768	仅 EBS 优化	30000	37.5
db.r6i.16xlarge	64	—	512	仅 EBS 优化	20000	25
db.r6i.12xlarge	48	—	384	仅 EBS 优化	15000	18.75
db.r6i.8xlarge	32	—	256	仅 EBS 优化	10000	12.5
db.r6i.4xlarge	16	—	128	仅 EBS 优化	最高 10,000	最多 12.5
db.r6i.2xlarge	8	—	64	仅 EBS 优化	最高 10,000	最多 12.5
db.r6i.xlarge	4	—	32	仅 EBS 优化	最高 10,000	最多 12.5
db.r6i.large	2	—	16	仅 EBS 优化	最高 10,000	最多 12.5

db.r5 – 内存优化型实例类

实例类	vCPU	ECU	内存（GiB）	实例存储（GiB）	最大 EBS 带宽（Mbps）	网络带宽（Gbps）
db.r5.24xlarge	96	347	768	仅 EBS 优化	19000	25
db.r5.16xlarge	64	264	512	仅 EBS 优化	13600	20
db.r5.12xlarge	48	173	384	仅 EBS 优化	9500	12
db.r5.8xlarge	32	132	256	仅 EBS 优化	6800	10
db.r5.4xlarge	16	71	128	仅 EBS 优化	4750	最多 10
db.r5.2xlarge	8	38	64	仅 EBS 优化	最多 4,750	最多 10
db.r5.xlarge	4	19	32	仅 EBS 优化	最多 4,750	最多 10

实例类	vCPU	ECU	内存（GiB）	实例存储（GiB）	最大 EBS 带宽（Mbps）	网络带宽（Gbps）
db.r5.large	2	10	16	仅 EBS 优化	最多 4,750	最多 10

db.r4 – 具有 Intel Xeon 可扩展处理器的内存优化型实例类

实例类	vCPU	ECU	内存（GiB）	实例存储（GiB）	最大 EBS 带宽（Mbps）	网络带宽（Gbps）
db.r4.16xlarge	64	195	488	仅 EBS 优化	14,000	25
db.r4.8xlarge	32	99	244	仅 EBS 优化	7,000	10
db.r4.4xlarge	16	53	122	仅 EBS 优化	3,500	最多 10
db.r4.2xlarge	8	27	61	仅 EBS 优化	1,700	最多 10
db.r4.xlarge	4	13.5	30.5	仅 EBS 优化	850	最多 10
db.r4.large	2	7	15.25	仅 EBS 优化	425	最多 10

可突增性能实例类的硬件规格

下表显示了可突增性能实例类的计算、内存、存储和带宽规格。

db.t4g - 由 AWS Graviton2 处理器提供支持的可突增性能实例类

实例类	vCPU	ECU	内存（GiB）	实例存储（GiB）	最大 EBS 带宽（Mbps）	网络带宽（Gbps）
db.t4g.large	2	—	8	仅 EBS 优化	最多 2,780	最多 5
db.t4g.medium	2	—	4	仅 EBS 优化	最多 2,085	最多 5

db.t3 – 具爆发能力的实例类

实例类	vCPU	ECU	内存（GiB）	实例存储（GiB）	最大 EBS 带宽（Mbps）	网络带宽（Gbps）
db.t3.large	2	变量	8	仅 EBS 优化	最多 2,048	最多 5
db.t3.medium	2	变量	4	仅 EBS 优化	最多 1,536	最多 5
db.t3.small	2	变量	2	仅 EBS 优化	最多 1,536	最多 5

db.t2 – 具爆发能力的实例类

实例类	vCPU	ECU	内存（GiB）	实例存储（GiB）	最大 EBS 带宽（Mbps）	网络带宽（Gbps）
db.t2.medium	2	变量	4	仅限于 EBS	—	中
db.t2.small	1	变量	2	仅限于 EBS	—	低

Amazon Aurora 存储

接下来，您可以了解 Aurora 存储子系统。Aurora 使用分布式的共享存储架构，这是影响 Aurora 集群的性能、可扩展性和可靠性的一个重要因素。

主题

- Amazon Aurora 存储概述
- 集群卷包含的内容
- Amazon Aurora 数据库集群的存储配置
- Aurora 存储如何自动调整大小
- Aurora 数据存储的计费方式

Amazon Aurora 存储概述

Aurora 数据存储在集群卷 中，该集群卷是使用固态驱动器 (SSD) 的单个虚拟卷。集群卷由跨一个 AWS 区域中的三个可用区的数据副本组成。由于数据会自动跨可用区复制，因此，数据具有高持久性，同时数据丢失的可能性很小。此复制还确保您的数据库在失效转移期间更可用。这样做是因为数据

副本已存在于其他可用区中并且继续响应对数据库集群中数据库实例的数据请求。复制的数量与集群中的数据库实例数量无关。

Aurora 对非持久性临时文件使用单独的本地存储。这包括用于在查询处理过程对大型数据集进行排序或用于索引构建等用途的文件。有关更多信息，请参阅Aurora MySQL 的临时存储限制和Aurora PostgreSQL 的临时存储限制。

集群卷包含的内容

Aurora 集群卷包含所有用户数据、架构对象和内部元数据，例如系统表和二进制日志。例如，Aurora 在集群卷中存储一个 Aurora 集群的所有表、索引、二进制大对象 (BLOB)、存储过程等。

Aurora 共享存储架构使您的数据独立于集群中的数据库实例。例如，您可以快速添加数据库实例，因为 Aurora 不会创建表数据的新副本。相反，数据库实例连接到已包含所有数据的共享卷。您可以从集群中删除数据库实例，而无需从集群中删除任何基础数据。仅当您删除整个集群时，Aurora 才会删除数据。

Amazon Aurora 数据库集群的存储配置

Amazon Aurora 有两种数据库集群存储配置：

- Aurora I/O-Optimized – 提高了 I/O 密集型应用程序的性价比和可预测性。您只需为数据库集群的使用量和存储付费，而无需为读取和写入 I/O 操作支付额外费用。

  当您的 I/O 支出占 Aurora 数据库总支出的 25% 或更多时，Aurora I/O-Optimized 是最佳选择。

  当您使用支持 Aurora I/O-Optimized 集群配置的数据库引擎版本创建或修改数据库集群时，可以选择 Aurora I/O-Optimized。您可以随时从 Aurora I/O-Optimized 切换到 Aurora Standard。
- Aurora Standard – 为许多 I/O 使用率适中的应用程序提供经济实惠的定价。除了数据库集群的使用量和存储外，您还需要为每 100 万个 I/O 操作请求支付标准费率。

  当您的 I/O 支出低于 Aurora 数据库总支出的 25% 时，Aurora Standard 是最佳选择。

  您可以每 30 天从 Aurora Standard 切换到 Aurora I/O-Optimized 一次。当您针对非基于 NVMe 的数据库实例在 Aurora Standard 和 Aurora I/O-Optimized 存储选项之间切换时，不会出现停机。但是，对于基于 NVMe 的数据库实例，当您在 Aurora I/O-Optimized 和 Aurora Standard 存储选项之间切换时，需要重新启动数据库引擎，而这可能导致短暂的停机。

有关 AWS 区域和版本支持的信息，请参阅支持集群存储配置的区域和 Aurora 数据库引擎。

有关 Amazon Aurora 存储配置定价的更多信息，请参阅 Amazon Aurora 定价。

有关在创建数据库集群时选择存储配置的信息，请参阅创建数据库集群。有关修改数据库集群的存储配置的信息，请参阅Amazon Aurora 设置。

Aurora 存储如何自动调整大小

Aurora 集群卷随着数据库中数据量的增加自动增大。有关每个引擎版本的最大 Aurora 集群卷大小的信息，请参阅 Amazon Aurora 大小限制。这种自动存储扩展与高性能和高分布式存储子系统相结合。当您的主要目标是可靠性和高可用性时，这些使 Aurora 成为您重要的企业数据的理想选择。

要显示卷状态，请参阅 显示 Aurora MySQL 数据库集群的卷状态 或 显示 Aurora PostgreSQL 数据库集群的卷状态。有关平衡存储成本与其他优先级的方法，存储扩展 介绍了如何在 CloudWatch 中监控 Amazon Aurora 指标 `AuroraVolumeBytesLeftTotal` 和 `VolumeBytesUsed`。

删除 Aurora 数据后，将释放为该数据分配的空间。删除数据的示例包括删除或截断表。这种自动减少存储使用量有助于您显著减少存储费用。

> **Note**
>
> 此处讨论的存储限制和动态调整大小行为适用于存储在集群卷中的持久性表和其他数据。
> 对于 Aurora PostgreSQL，临时表数据存储在本地数据库实例中。
> 对于 Aurora MySQL 版本 2，在默认情况下，临时表数据存储在写入器实例的集群卷中，读取器实例的临时表数据存储在本地存储中。有关更多信息，请参阅 磁盘上的临时表的存储引擎。
> 对于 Aurora MySQL 版本 3，临时表数据存储在本地数据库实例或集群卷中。有关更多信息，请参阅 Aurora MySQL 版本 3 中的新临时表行为。
> 驻留在本地存储中的临时表的最大大小受数据库实例的最大本地存储大小限制。本地存储大小取决于您使用的实例类。有关更多信息，请参阅Aurora MySQL 的临时存储限制和Aurora PostgreSQL 的临时存储限制。

某些存储功能（如集群卷的最大大小和删除数据时自动调整大小）取决于集群的 Aurora 版本。有关更多信息，请参阅 存储扩展。您还可以了解如何避免存储问题，以及如何监控集群中分配的存储和可用空间。

Aurora 数据存储的计费方式

即使对于特定的引擎版本，Aurora 集群卷最大可增大到 256 TiB，您也只需为在 Aurora 集群卷中使用的空间付费。在早期的 Aurora 版本中，集群卷可以重复使用在删除数据时释放的空间，但分配的存储

空间绝不会减少。现在，当删除 Aurora 数据（例如，通过删除表或数据库）时，总体分配的空间将减少与之相当的数量。因此，您可以通过删除不再需要的表、索引、数据库等来减少存储费用。

> **Tip**
>
> 对于没有动态调整大小功能的早期版本，重置集群的存储使用量涉及到执行逻辑转储和还原到新集群。对于大量的数据，该操作可能需要很长时间。如果您遇到这种情况，请考虑将集群升级到支持动态卷大小调整的版本。

有关哪些 Aurora 版本支持动态调整大小以及如何通过监控集群的存储使用量来显著减少存储费用的信息，请参阅 存储扩展。有关 Aurora 备份存储账单的信息，请参阅 了解 Amazon Aurora 备份存储使用量。有关 Aurora 数据存储的定价信息，请参阅 Amazon RDS for Aurora 定价。

Amazon Aurora 可靠性

Aurora 的设计具有可靠、持久和容错的特点。您可通过执行一些操作（例如，添加 Aurora 副本并将这些副本置于不同的可用区中）来构建 Aurora 数据库集群以提高可用性，Aurora 还包括一些自动化功能，这些功能可使其成为一种可靠的数据库解决方案。

主题

- 存储自动修复
- 自动恢复页面缓存
- 从计划外重启中恢复

存储自动修复

由于 Aurora 将多个数据副本保留在三个可用区中，因此大大降低了因磁盘故障导致数据丢失的可能性。Aurora 会自动检测集群卷包含的磁盘卷中的故障。如果磁盘卷的某个区段发生故障，Aurora 会立即修复该区段。在 Aurora 修复磁盘区段时，它使用集群卷包含的其他卷中的数据以确保已修复区段中的数据最新。因此，Aurora 将避免数据丢失，并减少了执行时间点还原以从磁盘故障恢复的需求。

自动恢复页面缓存

在 Aurora 中，每个数据库实例的页面缓存将通过与数据库不同的单独进程进行管理，这将允许页面缓存独立于数据库进行自动恢复。（页面缓存在 Aurora MySQL 上也称为 InnoDB 缓冲池，在 Aurora PostgreSQL 上也称为缓冲区缓存。）

万一发生数据库故障，页面缓存将保留在内存中，这样，当数据库重新启动时，页面缓存中的当前数据页将保持“热”状态。这样，初始查询便无需执行读取 I/O 操作来“预热”页面缓存，从而提高性能。

对于 Aurora MySQL，重新启动和失效转移时的页面缓存行为如下：

- 可以重启写入器实例，而无需重启读取器实例。
  - 如果读取器实例在写入器实例重新启动时没有重新启动，则它们不会丢失其页面缓存。
  - 如果读取器实例在写入器实例重新启动时重新启动，则它们确实会丢失其页面缓存。
- 当读取器实例重新启动时，写入器和读取器实例上的页面缓存都会保留。
- 当数据库集群发生失效转移时，效果与写入器实例重新启动时类似。在新的写入器实例（以前是读取器实例）上，页面缓存仍然存在，但是在读取器实例（以前是写入器实例）上，页面缓存将不存在。

对于 Aurora PostgreSQL，您可以使用集群缓存管理来保留指定读取器实例（在失效转移后会成为写入器实例）的页面缓存。有关更多信息，请参阅 通过 Aurora PostgreSQL 的集群缓存管理提供故障转移后的快速恢复。

从计划外重启中恢复

Aurora 设计为几乎可以立即从计划外重启中恢复，并在不使用二进制日志的情况下继续为您的应用程序数据提供服务。Aurora 在并行线程上异步执行恢复，以便在发生计划外重启后使数据库能够打开并立即恢复使用。

有关更多信息，请参阅Aurora 数据库集群的容错能力 和进行优化以缩短数据库重启时间。

下面是二进制日志记录与 Aurora MySQL 上的计划外重启恢复的注意事项：

- 直接在 Aurora 上启用二进制日志记录会影响计划外重启后的恢复时间，因为它强制数据库实例执行二进制日志恢复。
- 所用二进制日志记录的类型影响日志记录的大小和效率。对于相同数量的数据库活动，某些格式会比二进制日志中的其他格式记录更多信息。`binlog_format` 参数的以下设置会产生不同的日志数据量：
  - `ROW` – 最多的日志数据
  - `STATEMENT` – 最少的日志数据
  - `MIXED` – 中等数量的日志数据，通常提供最佳的数据完整性和性能组合

  二进制日志数据量影响恢复时间。二进制日志中记录的数据越多，数据库实例在恢复过程中就必须处理更多数据，这会增加恢复时间。

- 要通过二进制日志记录减少计算开销并缩短恢复时间，您可以使用增强型二进制日志。增强型二进制日志可将数据库恢复时间缩短多达 99%。有关更多信息，请参阅 为 Aurora MySQL 设置增强型二进制日志。
- Aurora 不需要二进制日志来复制数据库集群中的数据或执行时间点恢复（PITR）。
- 如果您不需要外部复制的二进制日志 (或外部二进制日志流)，我们建议您将 `binlog_format` 参数设置为 `OFF` 以禁用二进制日志记录。这样做可以减少恢复时间。

有关 Aurora 二进制日志记录和复制的更多信息，请参阅使用 Amazon Aurora 进行复制。有关不同 MySQL 复制类型的含义的更多信息，请参阅 MySQL 文档中的基于语句和基于行的复制的优点和缺点。

Amazon Aurora 安全性

Amazon Aurora 的安全性在三个级别上进行管理：

- 要控制可对 Aurora 数据库集群和数据库实例执行 Amazon RDS 托管操作的人员，请使用 AWS Identity and Access Management (IAM)。使用 IAM 凭证连接到AWS时，您的AWS账户必须具有授予执行 Amazon RDS 管理操作所需的权限的 IAM 策略。有关更多信息，请参阅 Amazon Aurora 的 Identity and Access Management。

  如果要使用 IAM 访问 Amazon RDS 控制台，则必须先使用您的用户凭证登录到 AWS Management Console，然后转至 https://console.aws.amazon.com/rds 上的 Amazon RDS 控制台。
- 必须基于 Amazon VPC 服务在 Virtual Private Cloud (VPC) 中创建 Aurora 数据库集群。要控制哪些设备和 Amazon EC2 实例能够建立与 VPC 中 Aurora 数据库集群的数据库实例的终端节点和端口的连接，请使用 VPC 安全组。您可以使用传输层安全性 (TLS)/安全套接字层 (SSL) 建立这些终端节点和端口连接。此外，公司的防火墙规则也可以控制公司中运行的哪些设备可以建立与数据库实例的连接。有关 VPC 的更多信息，请参阅Amazon VPC 和 Amazon Aurora。
- 要对 Amazon Aurora 数据库集群的登录名和权限进行身份验证，可单独或组合采用以下各种方式。
  - 您可以采用与单独 MySQL 或 PostgreSQL 数据库实例相同的方式。

    对单独 MySQL 或 PostgreSQL 数据库实例的登录名和权限进行身份验证的方法（例如，使用 SQL 命令或修改数据库表）也适用于 Aurora。有关更多信息，请参阅 使用 Amazon Aurora MySQL 实现高安全性 或 使用 Amazon Aurora PostgreSQL 实现高安全性。
  - 您可以使用 IAM 数据库身份验证。

如果采用 IAM 数据库身份验证，您可使用用户或 IAM 角色以及身份验证令牌对您的 Aurora 数据库集群进行身份验证。身份验证令牌是使用签名版本 4 签名流程生成的唯一值。通过使用 IAM 数据库身份验证，您可以使用相同的凭证来控制对 AWS 资源和数据库的访问。有关更多信息，请参阅 IAM 数据库身份验证。

- 您可以对 Aurora PostgreSQL 和 Aurora MySQL 使用 Kerberos 身份验证。

  在用户连接到 Aurora PostgreSQL 和 Aurora MySQL 数据库集群时，您可以使用 Kerberos 对用户进行身份验证。在这种情况下，数据库集群与 AWS Directory Service for Microsoft Active Directory 配合使用来启用 Kerberos 身份验证。AWS Directory Service for Microsoft Active Directory 也称为 AWS Managed Microsoft AD。将所有凭证保存在同一目录中可以节省您的时间和精力。您具有一个集中位置用于存储和管理多个数据库集群的凭证。使用目录还可以改善您的整体安全概要。有关更多信息，请参阅在 Aurora PostgreSQL 中使用 Kerberos 身份验证 和对 Aurora MySQL 使用 Kerberos 身份验证。

有关配置安全性的信息，请参阅Amazon Aurora 中的安全性。

将 SSL 与 Aurora 数据库集群配合使用

Amazon Aurora 数据库集群支持从使用与 Amazon RDS 数据库实例相同的过程和公有密钥的应用程序中建立安全套接字层 (SSL) 连接。有关更多信息，请参阅使用 Amazon Aurora MySQL 实现高安全性、使用 Amazon Aurora PostgreSQL 实现高安全性或将 TLS/SSL 与 Aurora Serverless v1 结合使用。

Amazon Aurora 的高可用性

Amazon Aurora 体系架构涉及存储和计算的分离。Aurora 包括一些适用于数据库集群中数据的高可用性功能。即使集群中的部分或全部数据库实例变得不可用，数据也会保持安全。其他高可用性功能适用于数据库实例。这些功能有助于确保一个或多个数据库实例准备就绪，以处理来自应用程序的数据库请求。

主题

- Aurora 数据的高可用性
- Aurora 数据库实例的高可用性
- 使用 Aurora Global Database 跨 AWS 区域的高可用性
- Aurora 数据库集群的容错能力

- Amazon RDS 代理的高可用性

Aurora 数据的高可用性

Aurora 跨一个 AWS 区域中的多个可用区将数据副本存储在数据库集群中。无论数据库集群中的数据库实例是否跨多个可用区，Aurora 都会存储这些副本。有关 Aurora 的更多信息，请参阅管理 Amazon Aurora 数据库集群。

在将数据写入到主数据库实例时，Aurora 将数据跨可用区同步复制到与集群卷关联的 6 个存储节点。这样做可以提供数据冗余，消除 I/O 冻结，以及在系统备份期间将延迟峰值降到最低。在计划内的系统维护期间，运行高性能的数据库实例可以提高可用性，并帮助保护数据库以防发生故障和可用区中断。有关可用区的更多信息，请参阅区域及可用区。

Aurora 数据库实例的高可用性

在创建主（写入器）实例后，可以创建多达 15 个只读 Aurora 副本。Aurora 副本也称为读取器实例。Aurora 副本使用异步复制来支持高可用性，不会影响主实例的性能。

在日常操作期间，您可以通过使用读取器实例处理 SELECT 查询来减轻读取密集型应用程序的部分工作负载。当问题影响到主实例时，其中一个读取器实例将作为主实例进行接管。此机制称为失效转移。许多 Aurora 功能适用于失效转移机制。例如，Aurora 检测数据库问题并在必要时自动激活失效转移机制。Aurora 还具有可缩短失效转移完成时间的功能。这样做可以显著减少数据库在失效转移期间无法写入的时间。

Aurora 旨在实现尽快恢复，最快的恢复途径通常是重启或失效转移到同一个数据库实例。与失效转移相比，重启速度更快，开销也更少。

想要使用相同的连接字符串，即使在失效转移提升了新的主实例时也保持不变，请连接到集群端点。集群端点 始终表示集群中的当前主实例。有关集群端点的更多信息，请参阅 Amazon Aurora 端点连接。

> **Tip**
>
> 在每个 AWS 区域内，可用区（AZ）表示相互不同的位置，以便在发生中断时提供隔离。建议您将数据库集群中的主实例和读取器实例分配到多个可用区，以提高数据库集群的可用性。这样，影响整个可用区的问题不会导致您的集群中断。
> 您可以通过在创建集群时进行简单选择来设置多可用区数据库集群。您可以使用 AWS Management Console、AWS CLI 或 Amazon RDS API。您还可以通过添加新的读取器数据库实例并指定不同的可用区，将现有 Aurora 数据库集群转换为多可用区数据库集群。

使用 Aurora Global Database 跨 AWS 区域的高可用性

为了跨多个 AWS 区域实现高可用性，您可以设置 Aurora Global Database。每个 Aurora Global Database 均跨越多个 AWS 区域，可在 AWS 区域中实现低延迟的全局读取以及从停机中进行灾难恢复。Aurora 异步将主 AWS 区域中的所有数据和更新复制到各个辅助区域。有关更多信息，请参阅 使用 Amazon Aurora Global Database。

Aurora 数据库集群的容错能力

Aurora 数据库集群设计为具有容错能力。集群卷跨一个 AWS 区域中的多个可用区 (AZ)，每个可用区均包含一个集群卷数据副本。该功能意味着您的数据库集群可容忍可用区的故障，而不发生任何数据丢失，只是会短暂中断服务。

如果数据库集群中的主实例失败，Aurora 将通过两种方式之一来自动失效转移到新的主实例：

- 将现有的 Aurora 副本提升为新的主实例
- 创建新的主实例

如果数据库集群具有一个或多个 Aurora 副本，则 Aurora 副本将在故障事件期间被提升为主实例。故障事件将导致短暂中断，其间的读取和写入操作将失败并引发异常。不过，服务通常会在 60 秒内（经常在 30 秒内）还原。要提高数据库集群的可用性，建议您在两个或更多的不同的可用区中创建至少一个或多个 Aurora 副本。

> **Tip**
>
> 在 Aurora MySQL 中，您可以通过在集群中拥有多个读取器数据库实例来提高失效转移期间的可用性。在 Aurora MySQL 中，Aurora 只重启它失效转移到的写入器数据库实例和读取器实例。集群中的其他读取器实例在失效转移期间仍然可用，以通过与读取器端点建立连接来继续处理查询。
>
> 还通过将 RDS 代理与 Aurora 数据库集群结合使用来提高失效转移期间的可用性。有关更多信息，请参阅 Amazon RDS 代理的高可用性。

您可以通过为每个副本分配一个优先级来自定义发生故障后将 Aurora 副本提升为主实例的顺序。优先级介于 0（最高优先级）和 15（最低优先级）之间。如果主实例失败，则 Amazon RDS 会将具有最高优先级的 Aurora 副本提升为新的主实例。您可以随时修改 Aurora 副本的优先级。修改优先级不会触发失效转移。

多个 Aurora 副本可共享同一个优先级，这会产生多个提升层。如果两个或更多 Aurora 副本共享同一个优先级，则 Amazon RDS 将提升最大的副本。如果两个或多个 Aurora 副本共享同一优先级和大小，则 Amazon RDS 将提升同一提升层中的任意副本。

> **Note**
>
> 确定失效转移的目标涉及多个因素。在失效转移尝试失败五次后，将不再考虑提升层。

如果数据库集群不包含任何 Aurora 副本，将在故障事件期间在同一可用区中重新创建主实例。故障事件将导致中断，其间的读取和写入操作将失败并引发异常。创建新的主实例时将还原服务，该操作所需的时间通常在 10 分钟内。将 Aurora 副本提升为主实例要比创建新的主实例快得多。

假设集群中的主实例由于影响整个可用区的中断而不可用。在这种情况下，使新主实例联机的方式取决于您的集群是否使用多可用区配置：

- 如果您的预置或 Aurora Serverless v2 集群包含其他可用区中的任何读取器实例，则 Aurora 会使用失效转移机制将其中一个读取器实例提升为新主实例。
- 如果您的预置或 Aurora Serverless v2 集群只包含一个数据库实例，或者主实例和所有读取器实例均位于同一可用区中，请确保在另一个可用区中手动创建一个或多个新数据库实例。
- 如果您的集群使用 Aurora Serverless v1，则 Aurora 会在另一个可用区中自动创建新数据库实例。但是，此过程涉及更换主机，因此比失效转移花费的时间更长。

> **Note**
>
> Amazon Aurora 还支持对外部 MySQL 数据库或 RDS MySQL 数据库实例的复制。有关更多信息，请参阅 Aurora 与 MySQL 之间或 Aurora 与其他 Aurora 数据库集群之间的复制（二进制日志复制）。

Amazon RDS 代理的高可用性

使用 RDS 代理，您可以构建能够透明地承受数据库故障的应用程序，而无需编写复杂的故障处理代码。代理会自动将流量路由到新的数据库实例，同时保留应用程序连接。它还绕过域名系统（DNS）缓存，以将 Aurora 多可用区数据库的失效转移时间缩短多达 66%。有关更多信息，请参阅 适用于 Aurora 的 Amazon RDS 代理。

使用 Amazon Aurora 进行复制

Aurora 有几个复制选项。每个 Aurora 数据库集群在同一集群中的多个数据库实例之间都有内置复制。您还可以在设置复制时将 Aurora 集群作为源或目标集群。将数据复制到或从 Aurora 集群中复制数据时，您可以选择内置功能，例如 Aurora Global Database，也可以选择用于 MySQL 或 PostgreSQL 数据库引擎的传统复制机制。您可以根据自己的需要选择适当的选项，以获得高可用性、便利性和性能的正确组合。以下部分说明选择每种方法的方式和时间。

主题

- Aurora 副本
- 使用 Aurora MySQL 进行复制
- 使用 Aurora PostgreSQL 进行复制

Aurora 副本

在 Aurora 预置数据库集群中创建第二个、第三个以及更多数据库实例时，Aurora 自动设置从写入器数据库实例到所有其他数据库实例的复制。这些其他数据库实例是只读实例，称为 Aurora 副本。讨论如何在集群中组合写入器和读取器数据库实例时，我们还将其称为读取器实例。

Aurora 副本有两个主要用途。您可以向他们发出查询以扩展应用程序的读取操作。为此，您通常需要连接到集群的读取器端点。这样，Aurora 可以将只读连接的负载分散到集群中尽可能多的 Aurora 副本之间。Aurora 副本还有助于提高可用性。如果集群中的写入器实例变为不可用，则 Aurora 会自动提升其中一个读取器实例以取代它作为新的写入器。

每个 Aurora 数据库集群最多可以包含 15 个 Aurora 副本。Aurora 副本可以分配到数据库集群在AWS区域中所跨的多个可用区。

数据库集群中的数据有自己的高可用性和可靠性功能，与集群中的数据库实例无关。如果您不熟悉 Aurora 存储功能，请参阅 Amazon Aurora 存储概述。数据库集群卷实际上由该数据库集群的多个数据副本组成。数据库集群中的主实例和 Aurora 副本都将集群卷中的数据作为单个逻辑卷查看。

因此，所有 Aurora 副本均返回相同的查询结果数据，且副本滞后时间非常短。此滞后通常远远少于主实例写入更新后的 100 毫秒。副本滞后因数据库更改速率而异。也就是说，在对数据库执行大量写入操作期间，您可能发现副本滞后时间变长。

> **Note**
>
> 在以下 Aurora PostgreSQL 版本中，当 Aurora Replica 与写入器数据库实例失去通信超过 60 秒时，它会自动重启：
>
> - 14.6 及更低版本
> - 13.9 及更低版本
> - 12.13 及更低版本
> - 所有 Aurora PostgreSQL 11 版本
>
> 有了读取可用性功能后，Aurora 副本不会自动重启。有关读取可用性功能及其适用版本的更多信息，请参阅提高 Aurora 副本的读取可用性。

Aurora 副本十分适用于读取扩展，因为它们完全专用于集群卷上的读取操作。写入操作由主实例进行管理。由于集群卷是在数据库集群中的所有数据库实例间共享的，因此无需其他操作即可复制每个 Aurora 副本的数据副本。

要提高可用性，可以使用 Aurora 副本作为失效转移目标。也就是说，如果主实例失败，Aurora 副本将提升为主实例。提升过程只造成短暂的中断，在此期间，对主实例发出的读写请求将失败，并且会出现异常。

通过失效转移提升 Aurora 副本要比重新创建主实例快得多。如果 Aurora 数据库集群不包含任何 Aurora 副本，则数据库集群在数据库实例从故障中恢复时不可用。

发生失效转移时，可能会重启某些 Aurora 副本，具体取决于数据库引擎版本。例如，在 Aurora MySQL 中，Aurora 在失效转移期间仅重启写入器数据库实例和失效转移目标。有关不同 Aurora 数据库引擎版本的重启行为的更多信息，请参阅重启 Amazon Aurora 数据库集群或 Amazon Aurora 数据库实例。有关重启或失效转移时页面缓存会发生什么情况的信息，请参阅自动恢复页面缓存。

对于高可用性场景，建议您创建一个或多个 Aurora 副本。您的 Aurora 数据库集群应该与主实例具有相同的数据库实例类，并且位于不同可用区中。有关将 Aurora 副本作为失效转移目标的更多信息，请参阅 Aurora 数据库集群的容错能力。

您无法为未加密的 Aurora 数据库集群创建已加密的 Aurora 副本。您无法为加密的 Aurora 数据库集群创建未加密的 Aurora 副本。

> **Tip**
>
> 您可以使用 Aurora 集群内的 Aurora 副本作为唯一复制形式，以保持数据高可用性。您还可以将内置 Aurora 复制与其他类型的复制结合使用。这样做可以帮助进一步提高您的数据的高可用性和地理分布水平。

有关如何创建 Aurora 副本的详细信息，请参阅将 Aurora 副本添加到数据库集群。

使用 Aurora MySQL 进行复制

除了 Aurora 副本以外，您还可以通过以下选项与 Aurora MySQL 进行复制：

- 不同AWS区域中的 Aurora MySQL 数据库集群。
  - 您可以使用 Aurora Global Database 跨多个区域复制数据。有关详细信息，请参阅 使用 Aurora Global Database 跨 AWS 区域的高可用性
  - 利用 MySQL 二进制日志 (binlog) 复制，您可以在不同的 AWS 区域中创建 Aurora MySQL 数据库集群的 Aurora 只读副本。通过这种方式，每个集群最多可以创建五个只读副本，且每个副本位于不同区域。
- 相同区域中的两个 Aurora MySQL 数据库集群（通过使用 MySQL 二进制日志(binlog) 复制实现）。
- 一个作为数据源的 RDS for MySQL 数据库实例，一个 Aurora MySQL 数据库集群（通过创建 RDS for MySQL 数据库实例的 Aurora 只读副本实现）。通常，此方法用于迁移到 Aurora MySQL，而非持续复制。

有关与 Aurora MySQL 进行复制的更多信息，请参阅使用 Amazon Aurora MySQL 进行复制。

使用 Aurora PostgreSQL 进行复制

除了 Aurora 副本之外，您还可以通过以下选项使用 Aurora PostgreSQL 进行复制：

- 使用 Aurora Global Database 时，在 1 个区域中有 1 个主 Aurora 数据库集群，在不同区域中最多有 10 个只读辅助数据库集群。Aurora PostgreSQL 不支持跨区域 Aurora 副本。不过，您可以使用 Aurora Global Database 将 Aurora PostgreSQL 数据库集群的读取功能扩展到多个 AWS 区域，从而实现可用性目标。有关更多信息，请参阅 使用 Amazon Aurora Global Database。
- 相同区域中的两个 Aurora PostgreSQL 数据库集群（通过使用 PostgreSQL 的逻辑复制功能实现）。

- 一个作为数据源的 RDS for PostgreSQL 数据库实例和一个 Aurora PostgreSQL 数据库集群（通过创建 RDS for PostgreSQL 数据库实例的 Aurora 只读副本实现）。通常，此方法用于迁移到 Aurora PostgreSQL，而非持续复制。

有关与 Aurora PostgreSQL 进行复制的更多信息，请参阅使用 Amazon Aurora PostgreSQL 进行复制。

Aurora 的数据库实例计费

Amazon Aurora 集群中的 Amazon RDS 预置实例根据以下组件进行计费：

- 数据库实例小时数（按小时）– 根据数据库实例的数据库实例类（例如，db.t2.small 或 db.m4.large）。定价以每小时为单位列出，但账单向下计算至秒，并以十进制形式显示时间。RDS 使用量以 1 秒的增量进行计费，时长最少 10 分钟。有关更多信息，请参阅 Amazon Aurora 数据库实例类。
- 存储（每月的 GB 数）– 您为数据库实例预配置的存储容量。如果您在计费期内扩展了预置的存储容量，则应按相应比例付费。有关更多信息，请参阅 Amazon Aurora 存储。
- 输入/输出（I/O）请求（每 1 百万个请求）– 您在计费周期内仅针对 Aurora Standard 数据库集群配置发出的存储 I/O 请求总数。

  有关 Amazon Aurora I/O 计费的更多信息，请参阅Amazon Aurora 数据库集群的存储配置。
- 备份存储（每月的 GiB 数）– 备份存储是指与自动数据库备份和拍摄的有效数据库快照相关联的存储。延长备份保留期或增加快照创建数量，将增加数据库所消耗的备份存储。按秒计费不适用于备份存储（按 GB/月计量）。

  有关更多信息，请参阅“备份和还原 Amazon Aurora 数据库集群”。
- 数据传输（GB 数）– 从 Internet 和其他AWS区域传入和传出数据库实例的数据。有关有用的示例，请参阅 AWS 博客文章 Exploring Data Transfer Costs for AWS Managed Databases。

Amazon RDS 提供了以下让您根据需求优化成本的购买选项：

- On-Demand Instances（按需实例）– 按使用的数据库实例小时数付费。定价以每小时为单位列出，但账单向下计算至秒，并以十进制形式显示时间。RDS 使用量现在以 1 秒的增量进行计费，时长最少 10 分钟。

- Reserved instances（预留实例）– 将数据库实例预留一年或三年，进而获得比按需数据库实例定价高很多的折扣。通过预留实例用量，您可以在 1 个小时内启动、删除、开始或停止多个实例并获得所有实例的预留实例优势。
- Aurora Serverless v2 – Aurora Serverless v2 提供按需容量，其中计费单位为 Aurora 容量单位 (ACU) 时数，而不是数据库实例时数。Aurora Serverless v2 容量在您指定的范围内增加和减少，具体取决于数据库的负载。您可以配置一个所有容量为 Aurora Serverless v2 的集群。或者，您可以配置 Aurora Serverless v2 与按需或预留预调配实例的组合。有关 Aurora Serverless v2 ACU 的工作原理的信息，请参阅 Aurora Serverless v2 的工作原理。
- Aurora PostgreSQL Limitless Database – Aurora PostgreSQL Limitless Database 是一种自动化水平扩缩功能，可扩展到超出单个数据库实例的写入吞吐量和存储限制。Limitless Database 能够将工作负载分布在多个 Aurora 写入器实例上，同时保持作为单个数据库的易操作性。Limitless Database 可提供按需容量，其中计费单位为数据库分片组中的 Aurora 容量单位（ACU）小时数。有关 Limitless Database ACU 工作原理的信息，请参阅创建使用 Aurora PostgreSQL Limitless Database 的数据库集群。

有关 Aurora 定价信息，请参阅 Aurora 定价页面。

主题

- Aurora 的按需数据库实例
- Amazon Aurora 的预留数据库实例

Aurora 的按需数据库实例

Amazon RDS 按需数据库实例根据数据库实例的类（例如，db.t3.small 或 db.m5.large）来计费。有关 Amazon RDS 定价信息，请参阅 Amazon RDS 产品页。

数据库实例的账单周期从该数据库实例可用时开始计算。定价以每小时为单位列出，但账单向下计算至秒，并以十进制形式显示时间。Amazon RDS 使用量以一秒的增量进行计费，时长最少 10 分钟。如果可计费配置发生更改，例如扩展计算或存储容量，则最低收取 10 分钟的费用。计费将一直继续到数据库实例终止，当您删除数据库实例或数据库实例出现故障时即终止。

如果不再希望您的数据库实例被收取费用，必须将其停止或删除，以免产生更多应计费数据库实例小时数。有关计费的数据库实例状态的更多信息，请参阅查看 Aurora 集群中的数据库实例状态。

已停止的数据库实例

当数据库实例停止时，您将为预配置存储付费，包括预置 IOPS。您还将为备份存储付费，包括在指定保留时间内用于手动快照和自动备份的存储。您无需为数据库实例小时数付费。

多可用区数据库实例

多可用区设置会在不同可用区中，自动预置和维护一个同步备用副本，用来增强数据的耐久性和可用性。由于更多的资源和更高的可用性，多可用区部署的价格要高于单可用区部署，而且由于备用实例和关联资源的增加，多可用区部署的成本可能约为两倍。

关于多可用区定价，请考虑以下重要细节：

- 计算成本：对于主实例和备用实例，均按照每数据库实例小时数来计费。
- 存储成本：对于主实例和备用实例，均按照所预置的每月 GB 数量收费。
- 数据传输费用：主实例和备用实例之间的复制已包含在费用中，但根据您的使用情况，可能会收取其他数据传输费用。

要根据您的具体应用场景和 AWS 区域准确估算每月费用，您可以使用 AWS 定价计算器。此工具让您可以输入配置详细信息，然后为您提供全面的成本明细。

> **Note**
>
> 定价可能随时可能会发生变化。有关最新的信息，请参阅 Amazon RDS 定价页面。

Amazon Aurora 的预留数据库实例

通过使用预留数据库实例，您可以将数据库实例预留一年或三年。相比按需数据库实例定价，数据库预留实例可以提供大幅折扣。预留数据库实例不是物理实例，而是对账户中使用的特定按需数据库实例所应用的账单折扣。用于预留数据库实例的折扣与实例类型和 AWS 区域相关联。

使用预留数据库实例的一般过程如下：首先，获取有关可用预留数据库实例产品的信息，然后购买预留数据库实例产品，最后获取有关您的现有预留数据库实例的信息。

有关购买预留数据库实例和查看预留数据库实例的账单的信息，请参阅以下各节。

- 购买 Amazon Aurora 的预留数据库实例
- 查看 Amazon Aurora 的预留数据库实例的账单

预留数据库实例概述

如果购买了 Amazon RDS 中的预留数据库实例，将承诺在预留数据库实例的持续时间内为您提供特定数据库实例类型的折扣费率。要使用 Amazon RDS 预留数据库实例，应创建新的数据库实例，就像您为按需实例创建数据库实例一样。

您创建的新数据库实例必须与预留的数据库实例具有相同的规范，具体为以下方面：

- AWS 区域
- 数据库引擎（数据库引擎的版本号不需要匹配。）
- 数据库实例类型

如果新数据库实例的规格与您的账户的现有预留数据库实例匹配，则会按照为预留数据库实例提供的折扣费率向您收费。否则，将以按需费率对数据库实例进行收费。

您可以修改用作预留数据库实例的数据库实例。如果修改在预留数据库实例的规范范围内，则部分或全部折扣仍适用于修改后的数据库实例。如果修改超出规范范围（例如更改实例类），则不再适用折扣。有关更多信息，请参阅 大小灵活的预留数据库实例。

主题

- 产品类型
- Aurora 数据库集群配置灵活性
- 大小灵活的预留数据库实例
- Aurora 预留数据库实例计费示例

- 删除预留数据库实例

有关预留数据库实例的更多信息（包括定价），请参阅 Amazon RDS 预留实例。

产品类型

预留数据库实例有三种类型（无预付费用、预付部分费用和预付全部费用），使您可以基于预期使用情况优化 Amazon RDS 成本。

> **Note**
>
> 并非所有 RDS 实例类都支持所有预留实例产品类型。例如，某些实例类可能不提供“无预付费用”选项。要确认可用性，请在 AWS Management Console中查看预留实例产品或使用 `describe-reserved-db-instances-offerings` AWS CLI 命令。

无费用预付

该选项无需预付款即可访问预留数据库实例。无论使用情况如何，您的“无费用预付”预留数据库实例都将按照期限内的小时数，采用打折小时费率进行计费，无需任何预付款。该选项仅以一年期预留形式提供。

预付部分费用

该选项需要预付部分预留数据库实例费用。期限内剩余的小时数无论使用情况如何，都将按照打折小时费率计费。该选项替换了以前的高使用率选项。

预付全部费用

所有款项于期限开始时支付，无论使用了多少小时数，剩余期限不会再产生其他任何费用。

如果使用整合账单，则将组织中的所有账户视为一个账户。这意味着，组织中的所有账户都可以享受任何其他账户购买的预留数据库实例的小时成本优惠。有关整合账单的更多信息，请参阅 AWS 账单和成本管理用户指南中的 Amazon RDS 预留数据库实例。

Aurora 数据库集群配置灵活性

您可以将 Aurora 预留数据库实例用于以下两种数据库集群配置：

- Aurora I/O-Optimized – 您只需为数据库集群的使用量和存储付费，而无需为读取和写入 I/O 操作支付额外费用。

- Aurora Standard – 除了数据库集群的使用量和存储外，您还需要为每 100 万个 I/O 操作请求支付标准费率。

Aurora 会自动考虑这些配置之间的价格差异。Aurora I/O-Optimized 每小时消耗的标准化单位比 Aurora Standard 多 30%。

有关 Aurora 集群存储配置的更多信息，请参阅Amazon Aurora 数据库集群的存储配置。有关 Aurora 集群存储配置定价的更多信息，请参阅 Amazon Aurora 定价。

大小灵活的预留数据库实例

在购买预留数据库实例时，您指定的一项内容是实例类，例如，db.r5.large。有关数据库实例类的更多信息，请参阅 Amazon Aurora 数据库实例类。

如果您具有数据库实例，则需要将其扩展为更大的容量，预留数据库实例将自动应用于扩展的数据库实例。即，在所有数据库实例类大小中自动应用预留数据库实例。大小灵活的预留数据库实例可供具有相同 AWS 区域和数据库引擎的数据库实例使用。大小灵活的预留数据库实例只能在其实例类类型中扩展。例如，db.r5.large 的预留数据库实例可以应用于 db.r5.xlarge，但不能应用于 db.r6g.large，因为 db.r5 和 db.r6g 属于不同类型的实例类。

大小灵活的预留数据库实例适用于以下 Aurora 数据库引擎：

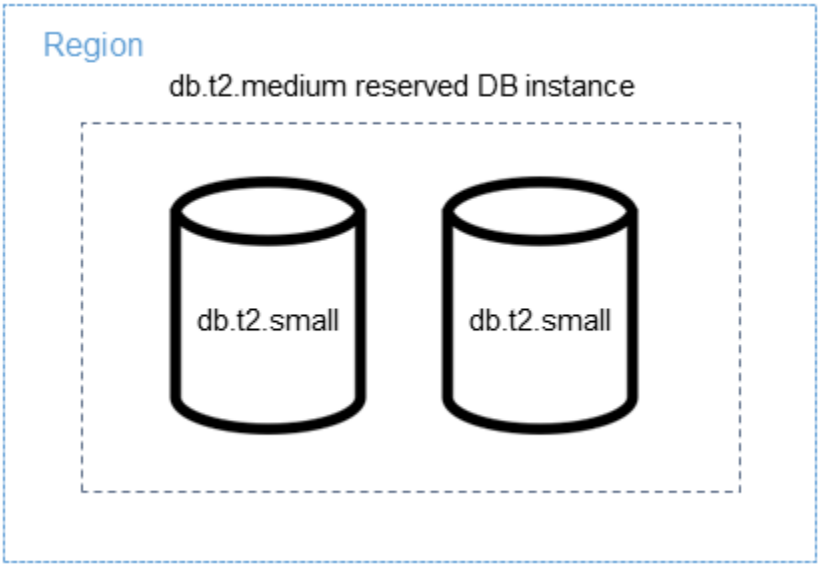
- Aurora MySQL
- Aurora PostgreSQL

您可以使用每小时标准化单位数来比较不同预留数据库实例大小的使用情况。例如，两个 db.r3.large 数据库实例的一个单位使用量相当于一个 db.r3.small 上的 8 个每小时标准化单位。下表显示了每个数据库实例大小的每小时标准化单位数。

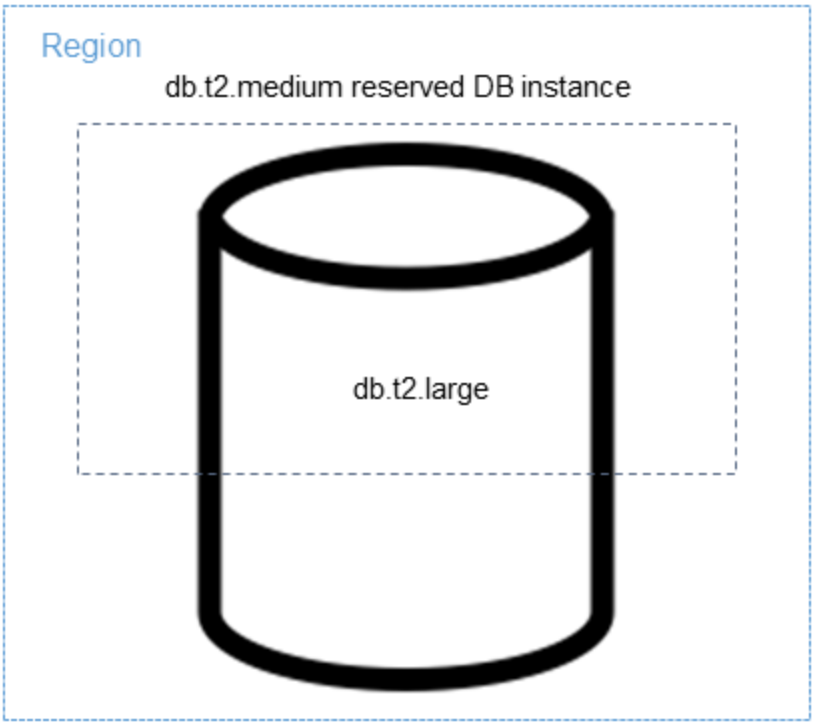
实例大小	一个数据库实例的每小时标准化单位，Aurora Standard	一个数据库实例的每小时标准化单位，Aurora I/O-Optimized	三个数据库实例（写入器和两个读取器）的标准化每小时单位，Aurora Standard	三个数据库实例（写入器和两个读取器）的标准化每小时单位，Aurora I/O-Optimized
small	1	1.3	3	3.9

实例大小	一个数据库实例的每小时标准化单位，Aurora Standard	一个数据库实例的每小时标准化单位，Aurora I/O-Optimized	三个数据库实例（写入器和两个读取器）的标准化每小时单位，Aurora Standard	三个数据库实例（写入器和两个读取器）的标准化每小时单位，Aurora I/O-Optimized
medium	2	2.6	6	7.8
large	4	5.2	12	15.6
xlarge	8	10.4	24	31.2
2xlarge	16	20.8	48	62.4
4xlarge	32	41.6	96	124.8
8xlarge	64	83.2	192	249.6
12xlarge	96	124.8	288	374.4
16xlarge	128	166.4	384	499.2
24xlarge	192	249.6	576	748.8
32xlarge	256	332.8	768	998.4

例如，假定您购买了 `db.t2.medium` 预留数据库实例，并且您的账户在同一 AWS 区域中具有两个正在运行的 `db.t2.small` 数据库实例。在这种情况下，账单优惠将完全应用于两个实例。



或者，如果您的账户在同一 AWS 区域中具有一个正在运行的 `db.t2.large` 实例，则账单优惠应用于 50% 的数据库实例使用量。



> **Note**
>
> 建议仅将 T 数据库实例类用于开发和测试服务器，或其他非生产服务器。有关 T 实例类的更多详细信息，请参阅数据库实例类类型。

Aurora 预留数据库实例计费示例

以下示例说明了同时使用 Aurora Standard 和 Aurora I/O-Optimized 数据库集群配置的 Aurora 数据库集群的预留数据库实例的定价。

使用 Aurora Standard 的示例

预留数据库实例的价格并没有为与存储、备份和 I/O 相关的成本提供折扣。以下示例说明预留数据库实例的每月总成本：

- 美国东部（弗吉尼亚州北部）的 Aurora MySQL 预留单可用区 db.r5.large 数据库实例类，每小时成本为 0.19 美元或每月 138.70 美元
- Aurora 存储，每月每 GiB 成本为 0.10 美元（假设本示例为每月 45.60 美元）
- Aurora I/O，每 100 万个请求的成本为 0.20 美元（假设本示例为每月 20 美元）
- Aurora 备份，每月每 GiB 为 0.021 美元（假设本示例为每月 30 美元）

将预留数据库实例的所有这些选项相加（138.70 + 45.60 + 20 + 30 美元），得出每月总成本为 234.30 美元。

如果您选择使用按需数据库实例而不是预留数据库实例，则美国东部（弗吉尼亚州北部）的 Aurora MySQL 单可用区 db.r5.large 数据库实例类的成本为每小时 0.29 美元或每月 217.50 美元。因此，对于按需数据库实例，将所有这些选项相加（217.50 + 45.60 + 20 + 30 美元），得出每月总成本为 313.10 美元。使用预留数据库实例，每月可节省近 79 美元。

示例：使用具有两个读取器实例的 Aurora Standard 数据库集群

要为 Aurora 数据库集群使用预留实例，只需为集群中的每个数据库实例购买一个预留实例。

扩展第一个示例，您有一个 Aurora MySQL 数据库集群，其中包含一个写入器数据库实例和两个 Aurora 副本，集群中共有三个数据库实例。两个 Aurora 副本不会产生额外的存储或备份费用。如果您购买三个 db.r5.large Aurora MySQL 预留数据库实例，则您的成本将为 234.30 美元（对于写入器数据库实例）+ 2 *（每个 Aurora 副本 138.70 美元 + 20 美元的 I/O），每月总额为 551.70 美元。

具有一个写入器数据库实例和两个 Aurora 副本的 Aurora MySQL 数据库集群的相应按需成本为 313.10 美元 + 2 * (217.50 美元 + 每个实例 20 美元输入/输出) ，每月总计为 788.10 美元。使用预留数据库实例，每月可节省 236.40 美元。

使用 Aurora I/O-Optimized 的示例

您可以将现有的 Aurora Standard 预留数据库实例重新用于 Aurora I/O-Optimized。要充分利用 Aurora I/O-Optimized 预留实例折扣的好处，您可以额外购买 30% 的与当前预留实例类似的预留实例。

下表显示了使用 Aurora I/O-Optimized 时如何估算额外预留实例数的示例。如果所需的预留实例数是小数，则可以利用预留实例提供的大小灵活性来获得整数。在这些示例中，“当前”是指您现在拥有的 Aurora Standard 预留实例。额外预留实例数是在使用 Aurora I/O-Optimized 时必须购买以保持当前预留实例折扣的 Aurora Standard 预留实例的数量。

数据库实例类	当前 Aurora Standard 预留实例数	Aurora I/O-Optimized 所需的预留实例数	需要的额外预留实例数	利用大小灵活性，需要的额外预留实例数
db.r6g.large	10	10 * 1.3 = 13	3 * db.r6g.large	3 * db.r6g.large
db.r6g.4xlarge	20	20 * 1.3 = 26	6 * db.r6g.4xlarge	6 * db.r6g.4xlarge
db.r6g.12xlarge	5	5 * 1.3 = 6.5	1.5 * db.r6g.12xlarge	db.r6g.12xlarge、r6g.4xlarge 和 r6g.2xlarge 各一个 （0.5 * db.r6g.12xlarge = 1 * db.r6g.4xlarge + 1 * db.r6g.2xlarge）
db.r6i.24xlarge	15	15 * 1.3 = 19.5	4.5 * db.r6i.24xlarge	4 * db.r6i.24xlarge + 1 * db.r6i.12xlarge

数据库实例类	当前 Aurora Standard 预留实例数	Aurora I/O-Optimized 所需的预留实例数	需要的额外预留实例数	利用大小灵活性，需要的额外预留实例数
				（0.5 * db.r6i.24xlarge = 1 * db.r6i.12xlarge）

示例：使用具有两个读取器实例的 Aurora I/O-Optimized 数据库集群

您有一个 Aurora MySQL 数据库集群，其中包含一个写入器数据库实例和两个 Aurora 副本，集群中共有三个数据库实例。它们使用 Aurora I/O-Optimized 数据库集群配置。要为此集群使用预留数据库实例，您需要购买四个属于相同数据库实例类的预留数据库实例。使用 Aurora I/O-Optimized 的三个数据库实例每小时消耗 3.9 个标准化单位，相比之下，使用 Aurora Standard 的三个数据库实例每小时消耗 3 个标准化单位。但是，您可以节省每个数据库实例的每月 I/O 成本。

> **Note**
>
> 这些示例中的价格是示例价格，可能与实际价格不符。有关 Aurora 定价信息，请参阅 Amazon Aurora 定价。

删除预留数据库实例

预留数据库实例具有一年或三年的使用期限。您无法取消预留数据库实例。不过，您可以删除预留数据库实例折扣涵盖的数据库实例。删除预留数据库实例折扣涵盖的数据库实例的过程与删除任何其他数据库实例相同。

无论您是否使用资源，都需要支付前期费用。

如果删除了预留数据库实例折扣涵盖的数据库实例，您可以启动另一个具有兼容规格的数据库实例。在这种情况下，您可以在预留期限（一年或三年）内继续享受折扣费率。

购买 Amazon Aurora 的预留数据库实例

您可以使用 AWS Management Console、AWS CLI 和 RDS API 处理预留数据库实例。

控制台

您可以使用 AWS Management Console处理预留数据库实例，如以下过程中所示。

获取有关可用预留数据库实例产品的定价和信息

1. 登录AWS Management Console并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择预留实例。
3. 选择购买预留的数据库实例。
4. 对于产品描述，请选择数据库引擎和许可类型。
5. 对于数据库实例类，请选择数据库实例类。
6. 对于部署选项，选择是需要单可用区还是多可用区数据库实例部署。

   > **Note**
   >
   > 预留 Amazon Aurora 实例的部署选项始终设置为单可用区数据库实例。但是，当您创建 Aurora 数据库集群时，原定设置部署选项是在不同的可用区（多可用区）中创建 Aurora 副本或读取器。
   > 您必须为计划使用的每个实例（包括 Aurora 副本）购买预留数据库实例。因此，对于 Aurora 上的多可用区部署，您必须购买额外的预留数据库实例。

7. 对于期限，选择要预留数据库实例的时间长度。
8. 对于产品类型，请选择产品类型。

   选择产品类型后，您可以看到定价信息。

   > **Important**
   >
   > 可以选择取消以避免购买预留数据库实例和产生任何费用。

在获得有关可用预留数据库实例产品的信息后，您可以使用该信息来购买以下过程中所示的产品。

购买预留数据库实例

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择预留实例。
3. 选择 Purchase reserved DB instance（购买预留数据库实例）。
4. 对于产品描述，请选择数据库引擎和许可类型。

5. 对于数据库实例类，请选择数据库实例类。
6. 对于多可用区部署，选择是需要单可用区还是多可用区数据库实例部署。

> **Note**
>
> 预留 Amazon Aurora 实例的部署选项始终设置为单可用区数据库实例。在从预留数据库实例中创建 Amazon Aurora 数据库集群时，该数据库集群将自动创建为多可用区。确保为您计划使用的每个数据库实例（包括 Aurora 副本）购买预留数据库实例。

7. 对于期限，选择希望预留数据库实例的时间长度。
8. 对于产品类型，请选择产品类型。

   选择产品类型后，您可以看到定价信息。
9. （可选）您可以将自己的标识符分配给购买的预留数据库实例，以帮助您跟踪这些实例。对于预留 ID，请为您的预留数据库实例键入一个标识符。
10. 选择提交。

    您的预留数据库实例已购买，然后显示在 Reserved instances（预留实例）列表中。

在购买了预留数据库实例后，您可以按以下过程中所示来获取有关预留数据库实例的信息。

获取有关 AWS 账户的预留数据库实例的信息

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择预留实例。

   您的账户的预留数据库实例随即出现。要查看有关特定预留数据库实例的详细信息，请在列表中选择该实例。然后，您可以在控制台底部的详细信息窗格中查看有关该实例的详细信息。

AWS CLI

您可以使用 AWS CLI 处理预留数据库实例，如以下示例中所示。

Example 获取可用预留数据库实例服务

要获取有关可用预留数据库实例产品的信息，请调用 AWS CLI 命令 `describe-reserved-db-instances-offerings`。

```
aws rds describe-reserved-db-instances-offerings
```

此调用返回类似于下述信息的输出：

```
OFFERING  OfferingId                            Class         Multi-AZ  Duration  Fixed
 Price  Usage Price  Description  Offering Type
OFFERING  438012d3-4052-4cc7-b2e3-8d3372e0e706  db.r3.large   y         1y
 1820.00 USD  0.368 USD    mysql        Partial  Upfront
OFFERING  649fd0c8-cf6d-47a0-bfa6-060f8e75e95f  db.r3.small   n         1y
 227.50 USD  0.046 USD    mysql        Partial  Upfront
OFFERING  123456cd-ab1c-47a0-bfa6-12345667232f  db.r3.small   n         1y
 162.00 USD   0.00 USD    mysql        All      Upfront
    Recurring Charges:   Amount  Currency  Frequency
    Recurring Charges:   0.123   USD       Hourly
OFFERING  123456cd-ab1c-37a0-bfa6-12345667232d  db.r3.large   y         1y
 700.00 USD   0.00 USD    mysql        All      Upfront
    Recurring Charges:   Amount  Currency  Frequency
    Recurring Charges:   1.25    USD       Hourly
OFFERING  123456cd-ab1c-17d0-bfa6-12345667234e  db.r3.xlarge  n         1y
 4242.00 USD   2.42 USD    mysql        No       Upfront
```

在获得有关可用预留数据库实例产品的信息后，您可以使用该信息来购买产品。

要购买预留数据库实例，请将 AWS CLI 命令 purchase-reserved-db-instances-offering 与以下参数结合使用：

- `--reserved-db-instances-offering-id` – 您要购买的产品的 ID。请参阅上述示例以获取产品 ID。
- `--reserved-db-instance-id` – 您可以将自己的标识符分配给购买的预留数据库实例，以帮助您跟踪这些实例。

Example 购买预留数据库实例

以下示例将购买 ID 为 *649fd0c8-cf6d-47a0-bfa6-060f8e75e95f* 的预留数据库实例产品，并分配标识符 *MyReservation*。

对于 Linux、macOS 或 Unix：

```
aws rds purchase-reserved-db-instances-offering \
```

```
    --reserved-db-instances-offering-id 649fd0c8-cf6d-47a0-bfa6-060f8e75e95f \
    --reserved-db-instance-id MyReservation
```

对于 Windows：

```
aws rds purchase-reserved-db-instances-offering ^
    --reserved-db-instances-offering-id 649fd0c8-cf6d-47a0-bfa6-060f8e75e95f ^
    --reserved-db-instance-id MyReservation
```

该命令返回的输出类似于下方内容：

```
RESERVATION  ReservationId      Class        Multi-AZ  Start Time
 Duration  Fixed Price  Usage Price  Count  State           Description  Offering Type
RESERVATION  MyReservation      db.r3.small  y         2011-12-19T00:30:23.247Z  1y
    455.00 USD   0.092 USD    1      payment-pending  mysql        Partial  Upfront
```

在购买了预留数据库实例后，您可以获取有关预留数据库实例的信息。

要获取有关您的 AWS 账户的预留数据库实例的信息，请按照以下示例所示调用 AWS CLI 命令 describe-reserved-db-instances。

Example 获取预留数据库实例

```
aws rds describe-reserved-db-instances
```

该命令返回的输出类似于下方内容：

```
RESERVATION  ReservationId     Class        Multi-AZ  Start Time
 Duration  Fixed Price  Usage Price  Count  State    Description  Offering Type
RESERVATION  MyReservation     db.r3.small  y         2011-12-09T23:37:44.720Z  1y
   455.00 USD   0.092 USD    1      retired  mysql        Partial  Upfront
```

RDS API

您可以使用 RDS API 处理预留数据库实例。

- 要获取有关可用预留数据库实例产品的信息，请调用 Amazon RDS API 操作 DescribeReservedDBInstancesOfferings。
- 在获得有关可用预留数据库实例产品的信息后，您可以使用该信息来购买产品。调用带以下参数的 PurchaseReservedDBInstancesOffering RDS API 操作：

- `--reserved-db-instances-offering-id` – 您要购买的产品的 ID。
- `--reserved-db-instance-id` – 您可以将自己的标识符分配给购买的预留数据库实例，以帮助您跟踪这些实例。

- 在购买了预留数据库实例后，您可以获取有关预留数据库实例的信息。调用 `DescribeReservedDBInstances` RDS API 操作。

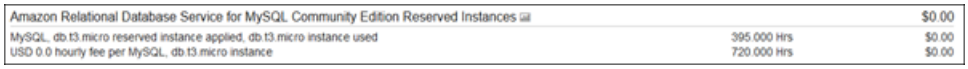
查看 Amazon Aurora 的预留数据库实例的账单

您可以在 AWS Management Console 中账单控制面板上查看预留数据库实例的账单。

要查看预留数据库实例账单

1. 登录到 AWS Management Console。
2. 从右上角的 account menu（账户菜单）中，选择 Billing Dashboard（账单控制面板）。
3. 选择控制面板右上角的 Bill Details（账单详细信息）。
4. 在 AWS Service Charges（ 服务费用）项下，展开 Relational Database Service（关系数据库服务）。
5. 展开您的预留数据库实例所在的 AWS 区域，例如 US West (Oregon) [美国西部（俄勒冈州）]。

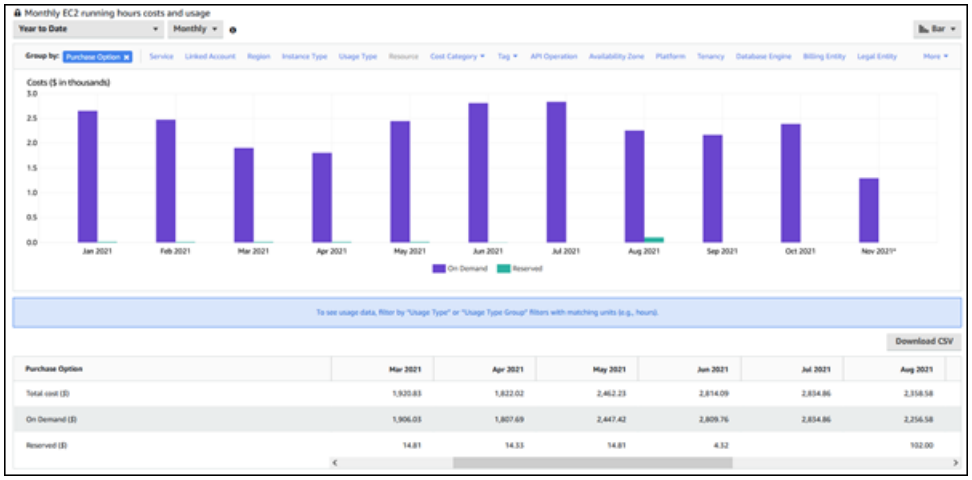
   您的预留数据库实例及其当月的每小时费用显示在 Amazon Relational Database Service for ***Database Engine*** Reserved Instances（适用于数据库引擎预留实例的 Amazon Relational Database Service）中。

   此示例中的预留数据库实例是预付全部费用购买的，因此不收取小时费用。
6. 选择 Reserved Instances（预留实例）标题旁的 Cost Explorer（条形图）图标。

   Cost Explorer 将显示 Monthly EC2 running hours costs and usage（每月 EC2 运行时间成本和使用情况）图表。
7. 清除图表右侧的 Usage Type Group（使用类型组）筛选条件。
8. 选择要检查使用成本的时间段和时间单位。

   以下示例按月显示了今年迄今按需和预留数据库实例的使用成本。



2021 年 1 月至 6 月的预留数据库实例成本是预付部分费用实例的月度费用，而 2021 年 8 月的费用是预付全部费用实例的一次性费用。

预付部分费用实例的预留实例折扣已于 2021 年 6 月到期，但该数据库实例未被删除。到期日期之后，只需以按需费率收费。

为 Amazon Aurora 设置环境

首次使用 Amazon Aurora 前，请完成以下任务。

主题

- 注册 AWS 账户
- 创建具有管理访问权限的用户
- 授权以编程方式访问
- 确定要求
- 通过创建安全组提供对 VPC 中数据库集群的访问

如果您已有 AWS 账户，知道您的 Aurora 要求并且喜欢使用 IAM 和 VPC 安全组的默认值，请向前跳到 开始使用 Amazon Aurora。

注册 AWS 账户

如果您还没有 AWS 账户，请完成以下步骤来创建一个。

注册 AWS 账户

1. 打开 https://portal.aws.amazon.com/billing/signup。
2. 按照屏幕上的说明操作。

   在注册时，将接到电话或收到短信，要求使用电话键盘输入一个验证码。

   当您注册 AWS 账户 时，系统将会创建一个 AWS 账户根用户。根用户有权访问该账户中的所有 AWS 服务和资源。作为最佳安全实践，请为用户分配管理访问权限，并且只使用根用户来执行需要根用户访问权限的任务。

注册过程完成后，AWS 会向您发送一封确认电子邮件。在任何时候，您都可以通过转至 https://aws.amazon.com/ 并选择我的账户来查看当前的账户活动并管理您的账户。

创建具有管理访问权限的用户

注册 AWS 账户 后，请保护好您的 AWS 账户根用户，启用 AWS IAM Identity Center，并创建一个管理用户，以避免使用根用户执行日常任务。

保护您的 AWS 账户根用户

1. 选择根用户并输入您的 AWS 账户 电子邮件地址，以账户拥有者身份登录 AWS Management Console。在下一页上，输入您的密码。

   要获取使用根用户登录方面的帮助，请参阅《AWS 登录 用户指南》中的 Signing in as the root user。
2. 为您的根用户启用多重身份验证（MFA）。

   有关说明，请参阅《IAM 用户指南》中的为 AWS 账户 根用户启用虚拟 MFA 设备（控制台）。

创建具有管理访问权限的用户

1. 启用 IAM Identity Center。

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的 Enabling AWS IAM Identity Center。
2. 在 IAM Identity Center 中，为用户授予管理访问权限。

   有关如何使用 IAM Identity Center 目录 作为身份源的教程，请参阅《AWS IAM Identity Center 用户指南》中的 Configure user access with the default IAM Identity Center 目录。

以具有管理访问权限的用户身份登录

- 要使用您的 IAM Identity Center 用户身份登录，请使用您在创建 IAM Identity Center 用户时发送到您的电子邮件地址的登录网址。

  要获取使用 IAM Identity Center 用户登录方面的帮助，请参阅《AWS 登录 用户指南》中的 Signing in to the AWS access portal。

将访问权限分配给其他用户

1. 在 IAM Identity Center 中，创建一个权限集，该权限集遵循应用最低权限的最佳做法。

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的 Create a permission set。
2. 将用户分配到一个组，然后为该组分配单点登录访问权限。

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的 Add groups。

授权以编程方式访问

如果用户需要在 AWS Management Console 之外与 AWS 交互，则需要编程式访问权限。授予编程式访问权限的方法取决于访问 AWS 的用户类型。

要向用户授予编程式访问权限，请选择以下选项之一。

哪个用户需要编程式访问权限？	目的	方式
人力身份 （在 IAM Identity Center 中管理的用户）	使用临时凭证签署向 AWS CLI、AWS SDK 或 AWS API 发出的编程请求。	按照您希望使用的界面的说明进行操作。 • 有关 AWS CLI 的更多信息，请参阅《AWS Command Line Interface 用户指南》中的配置 AWS CLI 以使用 AWS IAM Identity Center。 • 有关 AWS SDK、工具和 AWS API 的更多信息，请参阅《AWS SDK 和工具参考指南》中的 IAM Identity Center 身份验证。
IAM	使用临时凭证签署向 AWS CLI、AWS SDK 或 AWS API 发出的编程请求。	按照《IAM 用户指南》中将临时凭证用于 AWS 资源中的说明进行操作。
IAM	（不推荐使用） 使用长期凭证签署向 AWS CLI、AWS SDK 或 AWS API 发出的编程请求。	按照您希望使用的界面的说明进行操作。 • 有关 AWS CLI 的更多信息，请参阅《AWS Command Line Interface 用户指南》中的使用 IAM 用户凭证进行身份验证。 • 有关 AWS SDK 和工具的更多信息，请参阅《AWS SDK

哪个用户需要编程式访问权限？	目的	方式
		和工具参考指南》中的使用长期凭证进行身份验证。 • 有关 AWS API 的更多信息，请参阅《IAM 用户指南》中的管理 IAM 用户的访问密钥。

确定要求

Aurora 的基本构建基块是数据库集群。一个或多个数据库实例可以属于一个数据库集群。数据库集群提供了称为 集群终端节点的网络地址。每次您的应用程序需要访问在数据库集群中创建的数据库时，它们将连接到该数据库集群公开的集群终端节点。在创建数据库集群时指定的信息控制各种配置元素，如内存、数据库引擎和版本、网络配置、安全性以及维护时段。

在创建数据库集群和安全组之前，您必须知道数据库集群和网络需求。下面是要考虑的一些重要事项：

- 资源要求 – 应用程序或服务的内存和处理器要求是什么？ 在确定用于创建数据库集群的数据库实例类时，将会使用这些设置。有关数据库实例类的规范，请参阅 Amazon Aurora 数据库实例类。
- VPC、子网和安全组 – 您的数据库集群将在 Virtual Private Cloud (VPC) 中。必须配置安全组规则以连接到数据库集群。下面的列表说明每个 VPC 选项的规则：
  - 默认 VPC – 如果您的AWS账户在AWS区域中具有一个默认 VPC，将配置该 VPC 以支持数据库集群。如果您在创建数据库集群时指定默认 VPC：
    - 确保创建一个 VPC 安全组，该安全组对从应用程序或服务到 Aurora 数据库集群的连接进行授权。使用 VPC 控制台中的 Security Group (安全组) 选项或 AWS CLI 创建 VPC 安全组。有关信息，请参阅 步骤 3：创建 VPC 安全组。
    - 您必须指定默认数据库子网组。如果这是您在AWS区域中创建的第一个数据库集群，Amazon RDS 将在创建数据库集群时创建默认数据库子网组。
  - 用户定义的 VPC – 如果您希望在创建数据库集群时指定用户定义的 VPC：
    - 确保创建一个 VPC 安全组，该安全组对从应用程序或服务到 Aurora 数据库集群的连接进行授权。使用 VPC 控制台中的 Security Group (安全组) 选项或 AWS CLI 创建 VPC 安全组。有关信息，请参阅 步骤 3：创建 VPC 安全组。

  - 该 VPC 必须满足特定要求才能托管数据库集群，例如，至少具有两个子网，每个子网位于单独的可用区中。有关信息，请参阅 Amazon VPC 和 Amazon Aurora。
  - 您必须指定一个数据库子网组，以定义数据库集群可以使用该 VPC 中的哪些子网。有关信息，请参阅 在 VPC 中使用数据库集群 中的“数据库子网组”部分。
- 高可用性：是否需要失效转移支持？ 在 Aurora 上，多可用区部署创建一个主实例和一些 Aurora 副本。您可以将主实例和 Aurora 副本配置为位于不同的可用区以提供失效转移支持。我们建议将多可用区部署用于生产工作负载以保持高可用性。对于开发和测试用途，您可以使用非多可用区部署。有关更多信息，请参阅“Amazon Aurora 的高可用性”。
- IAM 策略：您的AWS账户是否具有相应策略来授予执行 Amazon RDS 操作所需的权限？ 如要使用 IAM 证书连接到AWS，您的 IAM 账户必须拥有 IAM 政策来授予执行 Amazon RDS 操作所需的权限。有关更多信息，请参阅“Amazon Aurora 的 Identity and Access Management”。
- 开放端口：您的数据库将要监听哪个 TCP/IP 端口？ 有些公司的防火墙可能会阻止与您的数据库引擎的默认端口进行连接。如果您的公司防火墙阻止使用默认端口，请为新数据库集群选择其他端口。请注意，在创建侦听指定端口的数据库集群后，您可以修改该数据库集群以更改端口。
- AWS区域：您希望您的数据库位于哪个AWS区域内？ 通过让数据库紧邻应用程序或 Web 服务，可以减小网络延迟。有关更多信息，请参阅“区域及可用区”。

在了解创建安全组和数据库集群所需的信息后，请继续执行下一步。

通过创建安全组提供对 VPC 中数据库集群的访问

您的数据库集群将在 VPC 中创建。安全组提供了 VPC 中的数据库集群的访问权限。它们充当关联的数据库集群的防火墙，以在集群级别控制入站和出站流量。默认情况下，将创建具有防火墙和默认安全组的数据库集群以禁止访问数据库集群。因此，您必须在安全组中添加规则以允许连接到数据库集群。使用在上一步中确定的网络和配置信息创建规则以允许访问数据库集群。

例如，如果您的应用程序将访问 VPC 中的数据库集群上的数据库，您必须添加自定义 TCP 规则以指定该应用程序用于访问数据库的端口范围和 IP 地址。如果您有位于 Amazon EC2 实例上的应用程序，则可以使用您为 Amazon EC2 实例设置的 VPC 安全组。

创建数据库集群时，您可以配置 Amazon EC2 实例与数据库集群之间的连接。有关更多信息，请参阅配置与 EC2 实例的自动网络连接。

> **Tip**
>
> 创建数据库集群时，您可以在 Amazon EC2 实例和数据库集群之间自动设置网络连接。有关更多信息，请参阅配置与 EC2 实例的自动网络连接。

有关如何将 Amazon Lightsail 中的资源连接到数据库集群的信息，请参阅 Connect Lightsail resources to AWS 服务 using VPC peering。

有关创建 VPC 以与 Aurora 结合使用的详细信息，请参阅 教程：创建 VPC 以用于数据库集群（仅限 IPv4）。有关访问数据库实例的常见场景的信息，请参阅 在 VPC 中访问数据库集群的场景。

创建 VPC 安全组

1. 登录到AWS Management Console并打开 Amazon VPC 控制台，网址：https://console.aws.amazon.com/vpc。

   > **Note**
   >
   > 确保您在 VPC 控制台中，而不是 RDS 控制台。

2. 在AWS Management Console的右上角，选择要在其中创建 VPC 安全组和数据库集群的AWS区域。在该AWS区域的 Amazon VPC 资源列表中，您应看到至少一个 VPC 和多个子网。如果您没有看到，则说明您在该 AWS 区域中没有默认 VPC。
3. 在导航窗格中，选择 Security Groups（安全组）。
4. 选择 Create security group（创建安全组）。

   此时将显示 Create security group (创建安全组) 页面。
5. 在 Basic details (基本详细信息) 中，输入 Security group name (安全组名称) 和 Description (描述)。对于 VPC，选择要在其中创建数据库集群的 VPC。
6. 在 Inbound rules (入站规则) 中，请选择 Add rule (添加规则)。

   a. 对于 Type (类型)，选择 Custom TCP (自定义TCP)。

   b. 对于 Port range (端口范围)，输入要用于数据库集群的端口值。

   c. 对于 Source (源)，选择安全组名称或键入您从中访问数据库集群的 IP 地址范围（CIDR 值）。如果您选择 My IP (我的 IP)，这会允许从浏览器中检测到的 IP 地址访问数据库集群。
7. 如果需要添加更多 IP 地址或不同的端口范围，请选择 Add rule (添加规则) 并输入规则的信息。

8. （可选）在 Outbound rules (出站规则) 中，为出站流量添加规则。默认情况下，允许所有出站流量。
9. 选择 Create security group（创建安全组）。

在创建数据库集群时，您将使用刚创建的 VPC 安全组作为数据库集群的安全组。

> **Note**
>
> 如果您使用默认 VPC，则为您创建跨越该 VPC 的所有子网的默认子网组。在创建数据库集群时，您可以选择默认 VPC 并为 DB Subnet Group (数据库子网组) 选择 default (默认)。

完成设置要求后，可以按照创建 Amazon Aurora 数据库集群中的说明使用您的要求和安全组创建数据库集群。有关开始创建使用特定数据库引擎的数据库集群的信息，请参阅 开始使用 Amazon Aurora。

开始使用 Amazon Aurora

在本节中，您可以了解如何使用 Amazon RDS 创建并连接到 Aurora 数据库集群。

以下过程是若干教程，说明了开始使用 Aurora 的基础知识。后续章节介绍了更高级的 Aurora 概念和过程，例如，不同类型的端点以及如何扩展和缩减 Aurora 集群。

> **Important**
>
> 务必先完成为 Amazon Aurora 设置环境部分中的任务，然后才能创建或连接到数据库集群。

主题

- 创建 Aurora MySQL 数据库集群并连接到该集群
- 创建 Aurora PostgreSQL 数据库集群并连接到该集群
- 教程：创建 Web 服务器和 Amazon Aurora 数据库集群

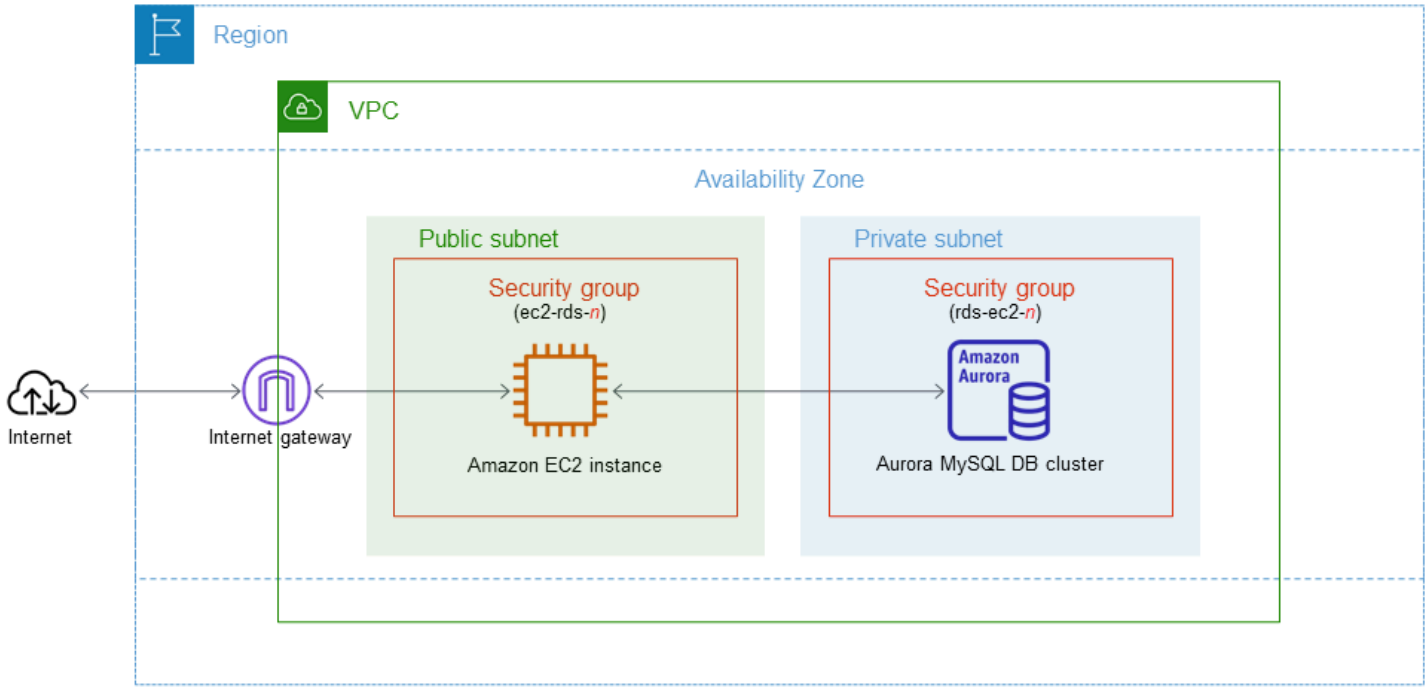
创建 Aurora MySQL 数据库集群并连接到该集群

本教程创建一个 EC2 实例和一个 Aurora MySQL 数据库集群。本教程向您展示如何使用标准 MySQL 客户端从 EC2 实例访问数据库集群。作为最佳实践，本教程在虚拟私有云（VPC）中创建私有数据库集群。在大多数情况下，同一 VPC 中的其他资源（例如 EC2 实例）可以访问数据库集群，但 VPC 之外的资源无法访问该集群。

完成本教程后，VPC 的每个可用区中都有一个公有和私有子网。在一个可用区中，EC2 实例在公有子网中，数据库实例在私有子网中。

> **Important**
>
> 创建 AWS 账户并不会收费；但是，在完成本教程过程中使用AWS资源可能会产生费用。完成本教程后，如果不再需要这些资源，可以将其删除。

下图显示了教程完成时的配置。



本教程可帮助您通过以下方法之一创建资源：

1. 使用 AWS Management Console - 步骤 1：创建 EC2 实例 和 步骤 2：创建 Aurora MySQL 数据库集群
2. 使用 AWS CloudFormation 创建数据库实例和 EC2 实例 - （可选）使用 AWS CloudFormation 创建 VPC、EC2 实例和 Aurora MySQL 集群

第一种方法使用轻松创建，通过 AWS Management Console创建私有 Aurora MySQL 数据库集群。您可以仅指定数据库引擎类型、数据库实例大小和数据库集群标识符。轻松创建为其他配置选项使用默认设置。

如果使用标准创建，则可以在创建数据库集群时指定更多配置选项。这些选项包括可用性、安全性、备份和维护设置。要创建公有数据库集群，必须使用标准创建。有关信息，请参阅the section called “创建数据库集群”。

主题

- 先决条件
- 步骤 1：创建 EC2 实例
- 步骤 2：创建 Aurora MySQL 数据库集群
- （可选）使用 AWS CloudFormation 创建 VPC、EC2 实例和 Aurora MySQL 集群
- 步骤 3：连接到 Aurora MySQL 数据库集群

- 步骤 4：删除 EC2 实例和数据库集群
- （可选）删除使用 CloudFormation 创建的 EC2 实例和数据库集群
- （可选）将您的数据库集群连接到 Lambda 函数

先决条件

在开始之前，请完成以下各节中的步骤：

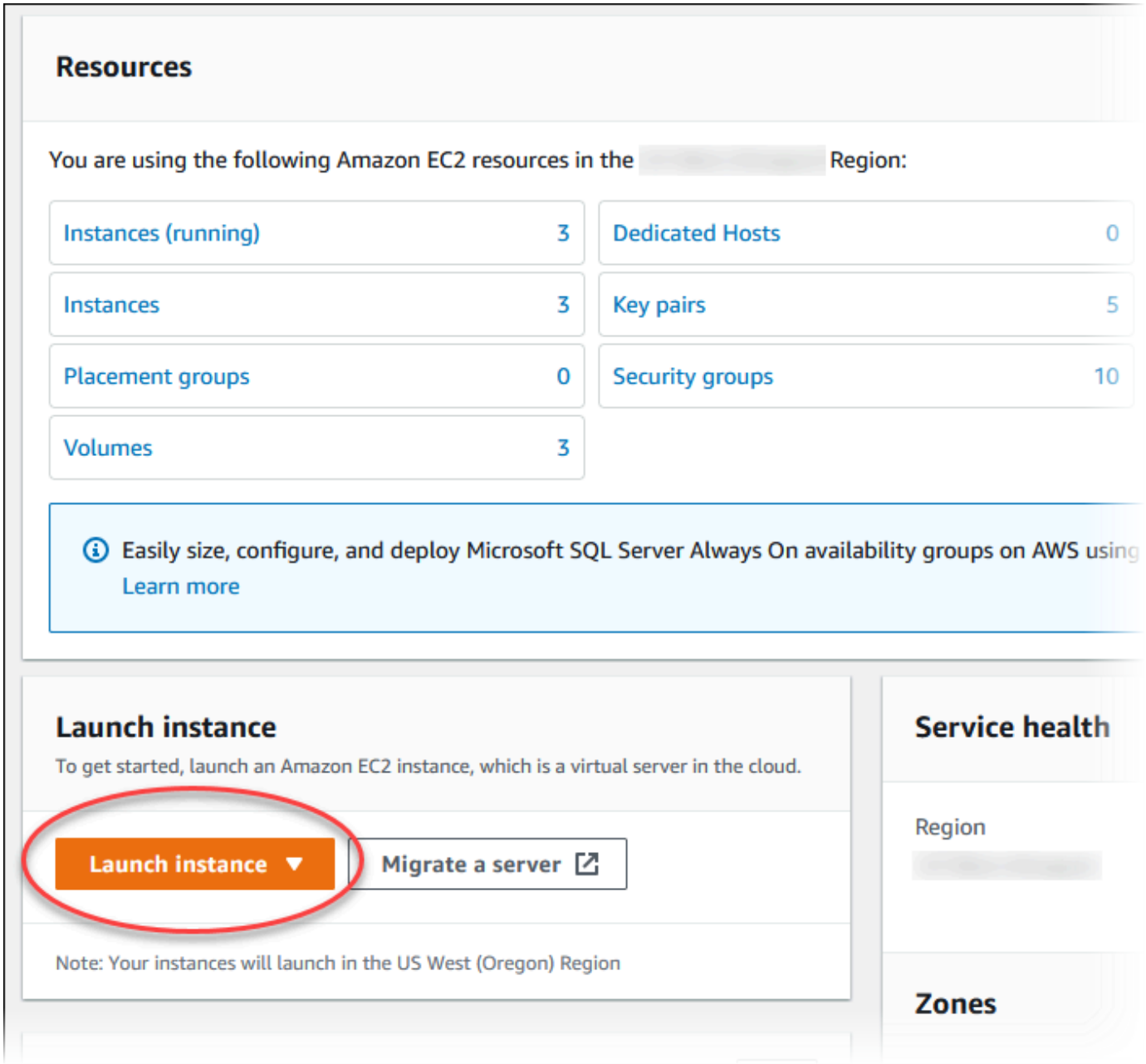
- 注册 AWS 账户
- 创建具有管理访问权限的用户

步骤 1：创建 EC2 实例

创建将用于连接到数据库的 Amazon EC2 实例。

创建 EC2 实例

1. 登录到 AWS Management Console 并打开 Amazon EC2 控制台（https://console.aws.amazon.com/ec2/）。
2. 在 AWS Management Console的右上角，选择要在其中创建 EC2 实例的 AWS 区域。
3. 选择 EC2 控制面板，然后选择启动实例，如下图所示。

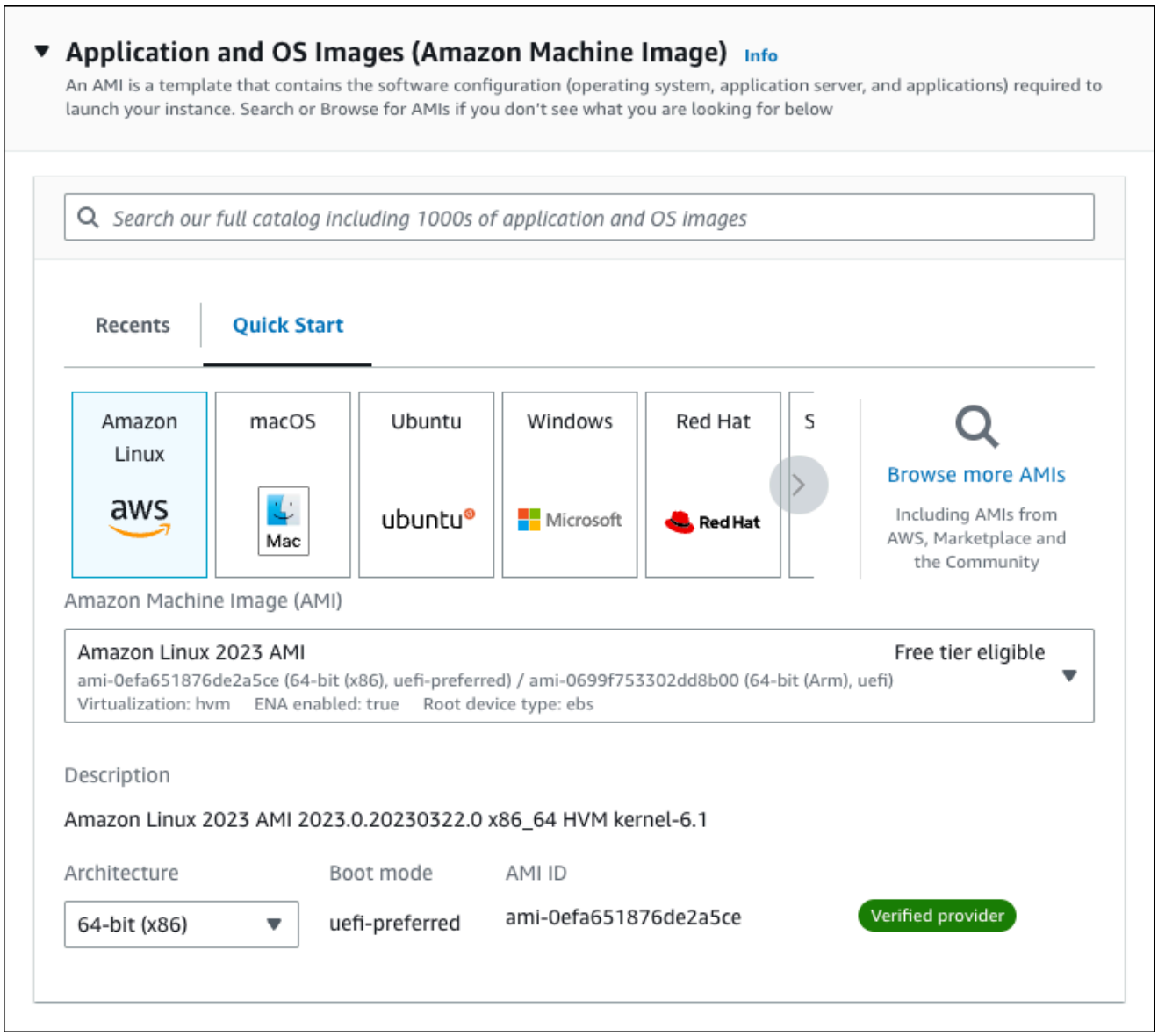


启动实例页面打开。

4. 在启动实例页面上选择以下设置。

   a. 在 Name and tags（名称和标签）下，对于 Name（名称），输入 **`ec2-database-connect`**。

   b. 在应用程序和操作系统映像（Amazon 机器映像）下，选择 Amazon Linux，然后选择 Amazon Linux 2023 AMI。对于其他选项，保留默认选择。



c. 在 Instance type（实例类型）下，选择 t2.micro。

d. 在 Key pair (login) [密钥对（登录）] 下，选择 Key pair name（密钥对名称）以使用现有密钥对。要为 Amazon EC2 实例创建新的密钥对，请选择 Create new key pair（创建新的密钥对），然后使用 Create key pair（创建密钥对）窗口来创建它。

   有关创建新的密钥对的更多信息，请参阅《Amazon EC2 用户指南》中的创建密钥对。

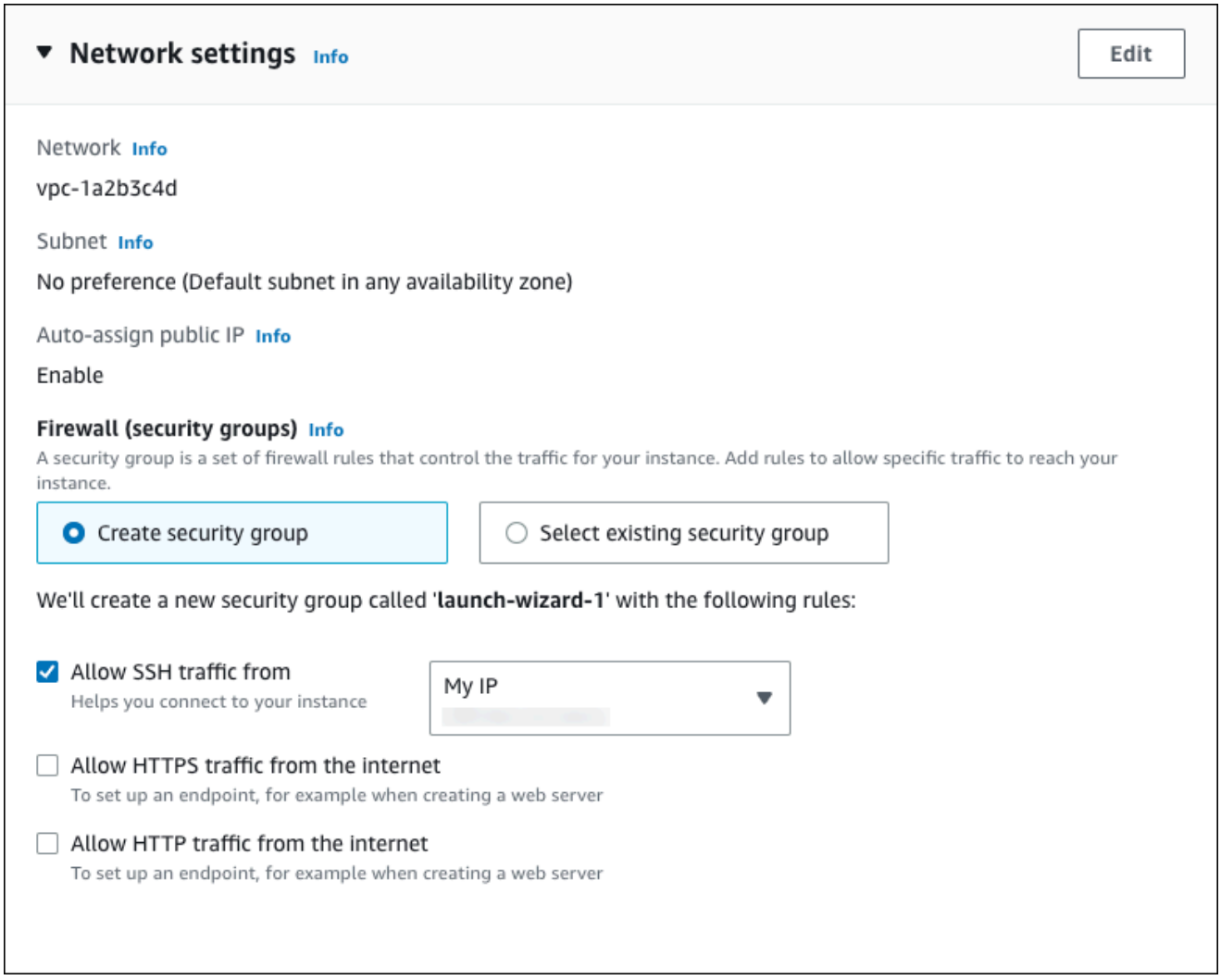
e. 对于网络设置中的允许 SSH 流量，选择 EC2 实例的 SSH 连接来源。

   如果显示的 IP 地址对于 SSH 连接是正确的，您可以选择 My IP（我的 IP）。否则，您可以确定要用来通过 Secure Shell（SSH）连接到 VPC 中的 EC2 实例的 IP 地址。要确定您的公有 IP 地址，请在新的浏览器窗口或标签页中，使用 https://checkip.amazonaws.com 上的服务。IP 地址的示例为 192.0.2.1/32。

在许多情况下，您可能通过互联网服务提供商（ISP）进行连接，或者在不使用静态 IP 地址的情况下从防火墙之后进行连接。如果是这样，请确保确定客户端计算机使用的 IP 地址范围。

> **Warning**
>
> 如果您使用 `0.0.0.0/0` 进行 SSH 访问，则所有 IP 地址可能能够使用 SSH 访问您的公有 EC2 实例。在测试环境下短时间内，此方法尚可接受，但它对于生产环境并不安全。在生产环境中，将仅向特定 IP 地址或地址范围授权使用 SSH 访问您的 EC2 实例。

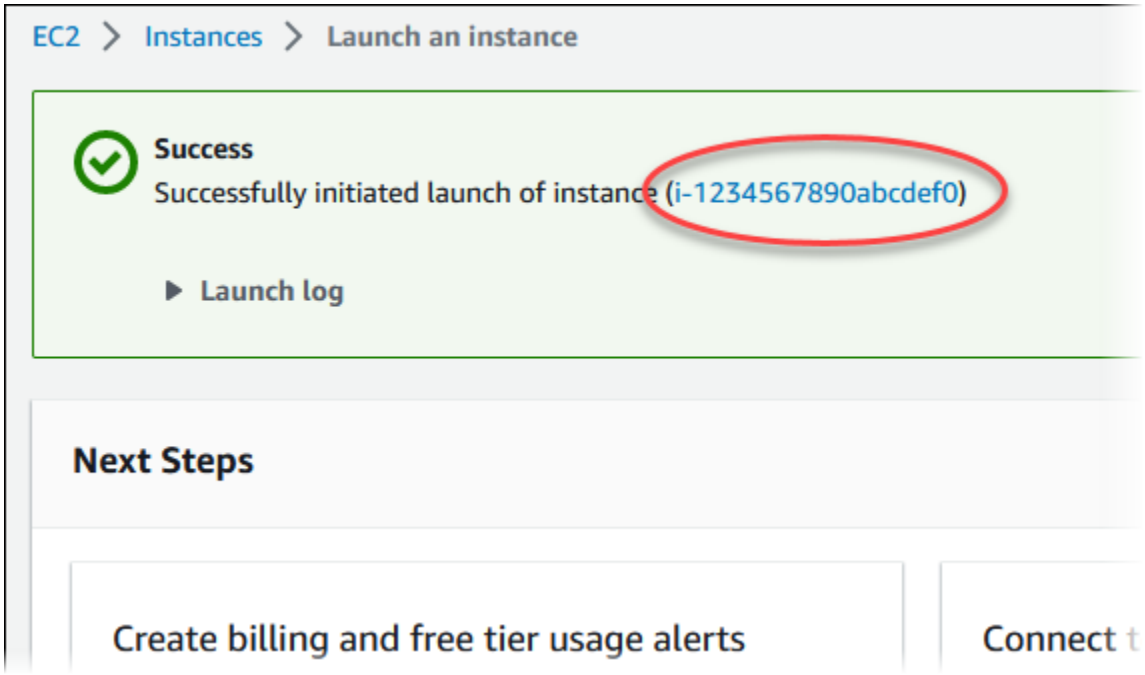
下图显示了网络设置部分的示例。



f. 对于其余部分保留默认值。

g. 查看摘要面板中您的 EC2 实例配置的摘要，当您准备好后，选择启动实例。

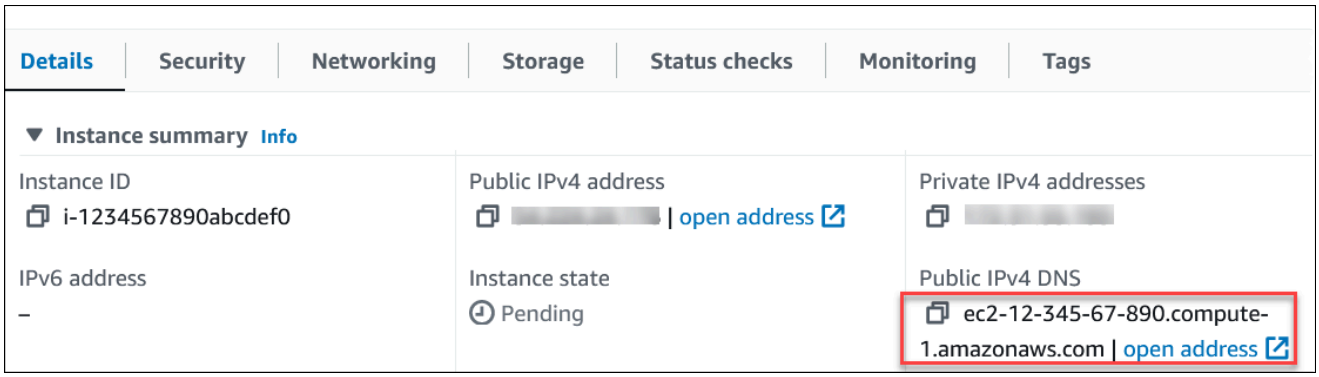
5. 在启动状态页面上，记下新 EC2 实例的标识符，例如：`i-1234567890abcdef0`。



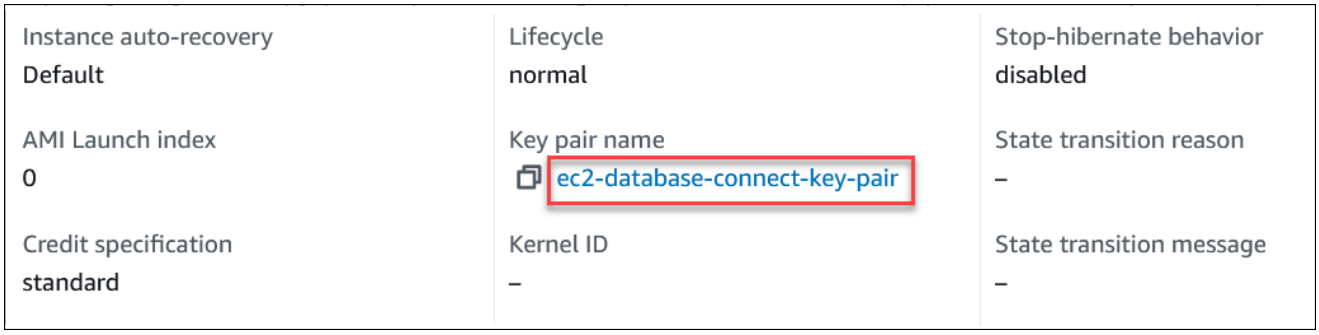
6. 选择 EC2 实例标识符以打开 EC2 实例列表，然后选择您的 EC2 实例。

7. 在详细信息选项卡中，记下使用 SSH 进行连接时所需的以下值：

   a. 在实例摘要中，记下公有 IPv4 DNS 的值。



   b. 在实例详细信息中，记下密钥对名称的值。



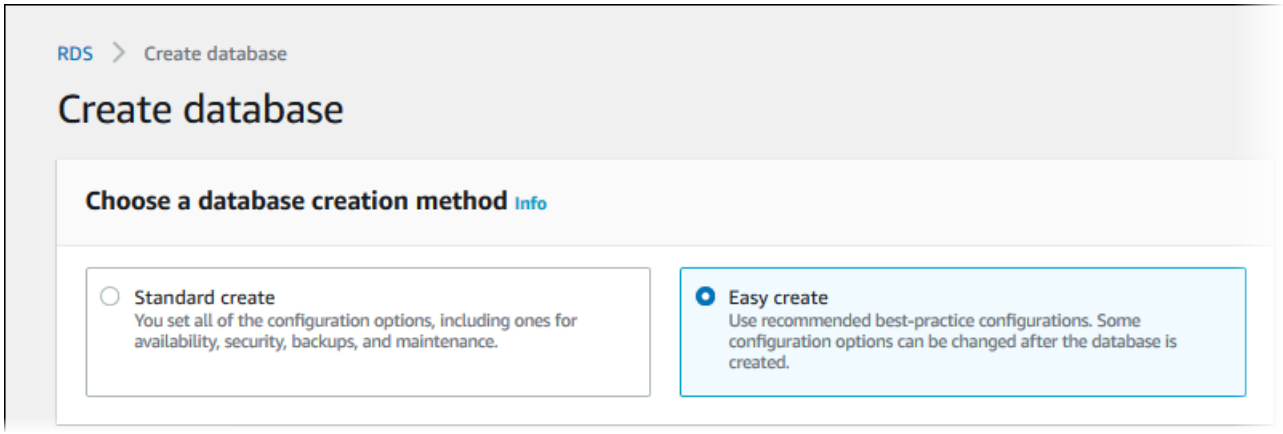
8. 等待 EC2 实例的实例状态变为正在运行，然后继续。

步骤 2：创建 Aurora MySQL 数据库集群

在该示例中，您使用轻松创建来创建一个具有 db.r6g.large 数据库实例类的 Aurora MySQL 数据库集群。

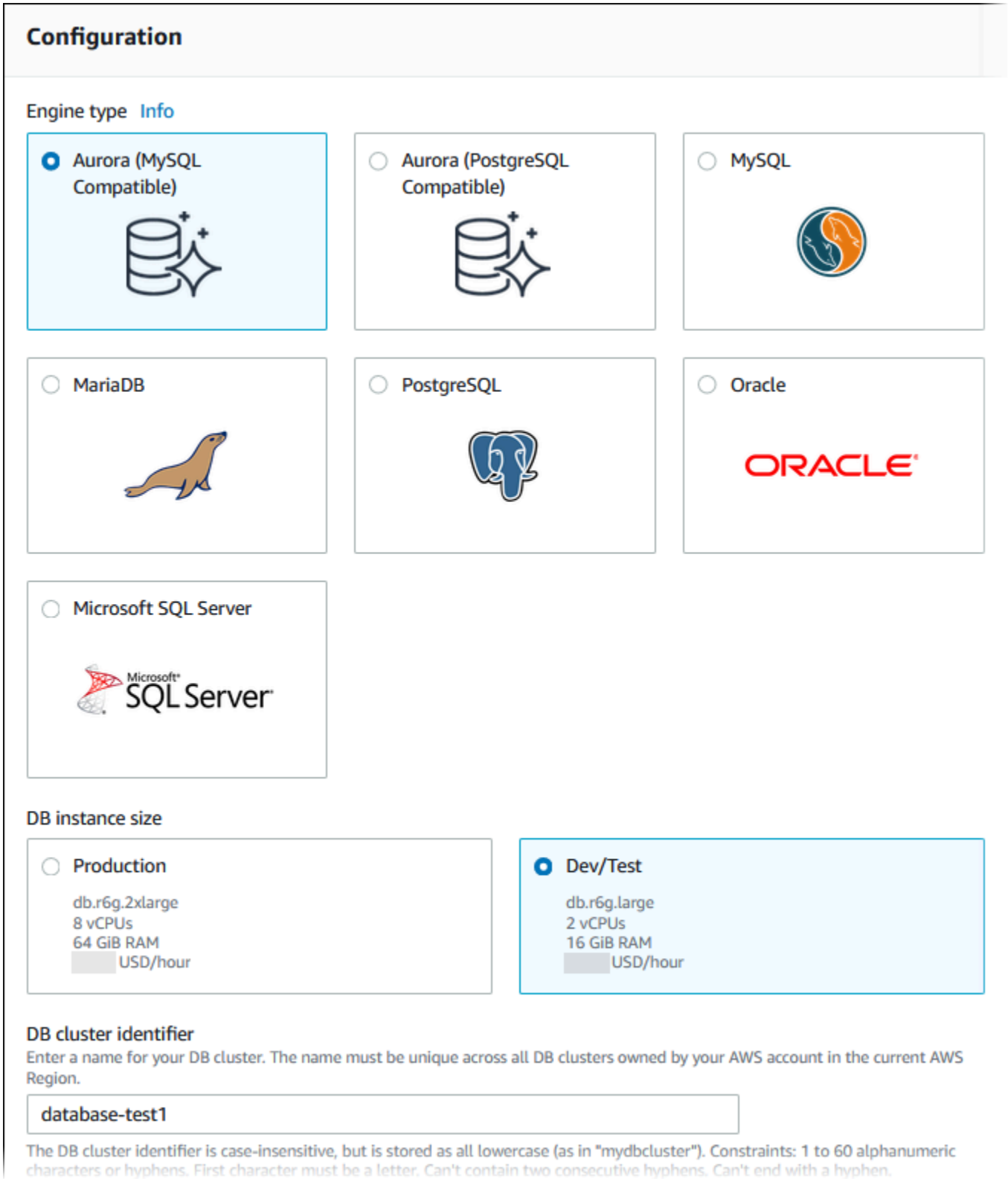
使用“轻松创建”来创建 Aurora MySQL 数据库集群

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在 Amazon RDS 控制台的右上角，选择要在其中创建数据库集群的 AWS 区域。
3. 在导航窗格中，选择 Databases (数据库)。
4. 选择 Create database (创建数据库)，并确保已选择 Easy create (轻松创建)。



5. 在配置中，对于引擎类型，选择 Aurora（MySQL 兼容）。
6. 对于数据库实例大小，选择设备/测试。
7. 对于数据库集群标识符，输入 **`database-test1`**。

   创建数据库页面应类似于以下图像。

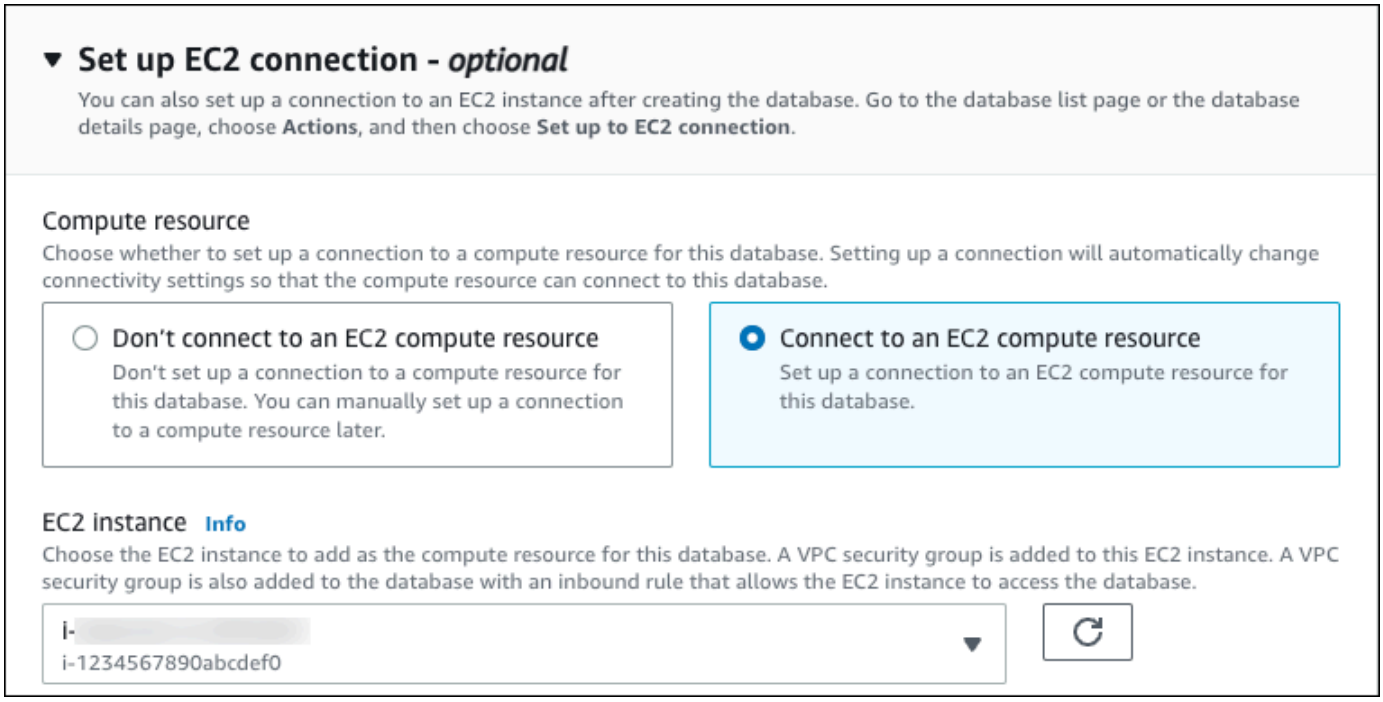


8. 对于主用户名，输入主用户的名称，或者保留默认名称。
9. 要为数据库集群使用自动生成的主密码，请选择自动生成密码。

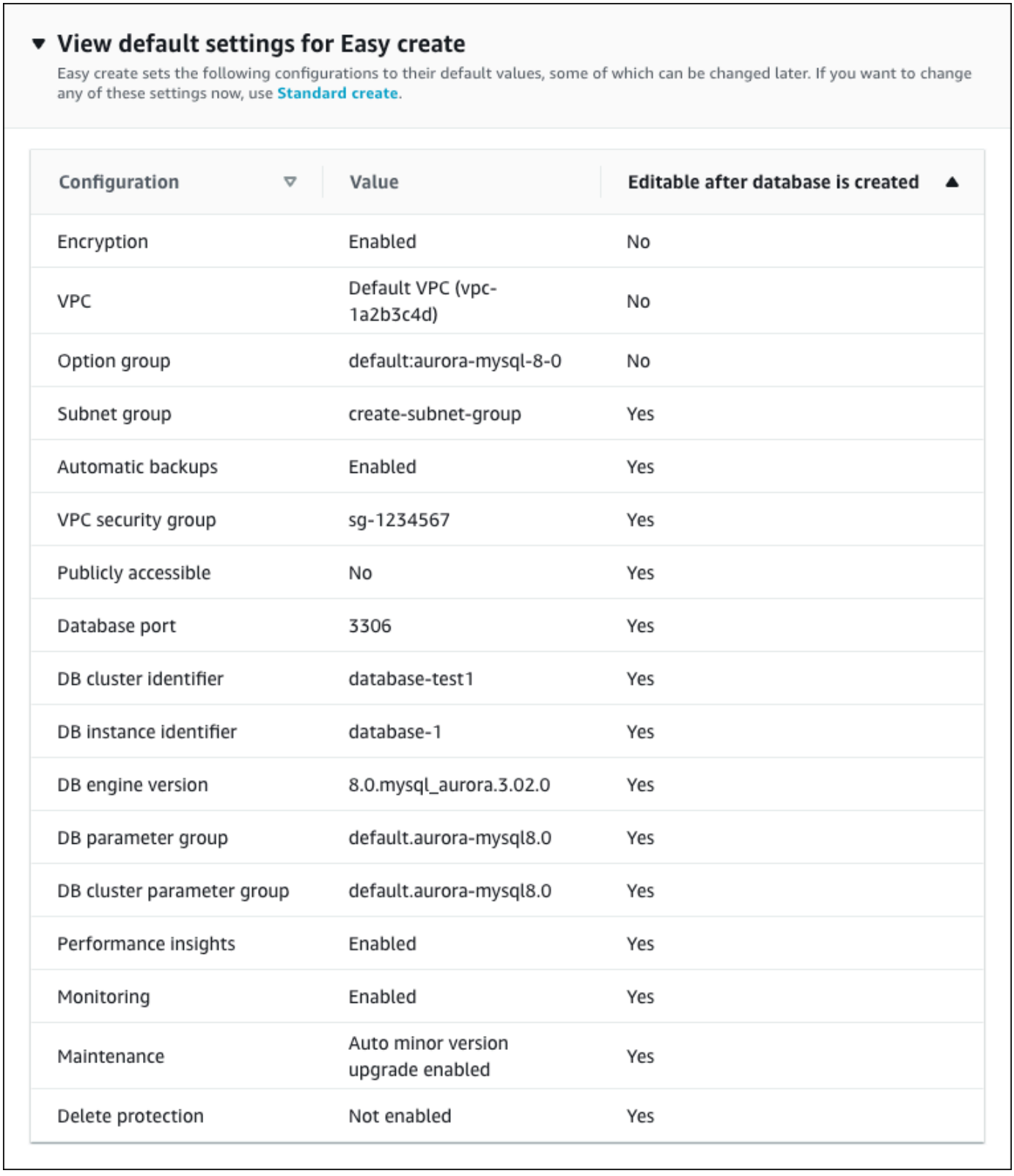
要输入主密码，请确保清除自动生成密码，然后在主密码和确认密码中输入相同的密码。

10. 要设置与您之前创建的 EC2 实例的连接，请打开设置 EC2 连接 - 可选。

    选择连接到 EC2 计算资源。选择您之前创建的 EC2 实例。



11. 打开查看轻松创建的默认设置。



View default settings for Easy create

Easy create sets the following configurations to their default values, some of which can be changed later. If you want to change any of these settings now, use **Standard create**.

Configuration	Value	Editable after database is created
Encryption	Enabled	No
VPC	Default VPC (vpc-1a2b3c4d)	No
Option group	default:aurora-mysql-8-0	No
Subnet group	create-subnet-group	Yes
Automatic backups	Enabled	Yes
VPC security group	sg-1234567	Yes
Publicly accessible	No	Yes
Database port	3306	Yes
DB cluster identifier	database-test1	Yes
DB instance identifier	database-1	Yes
DB engine version	8.0.mysql_aurora.3.02.0	Yes
DB parameter group	default.aurora-mysql8.0	Yes
DB cluster parameter group	default.aurora-mysql8.0	Yes
Performance insights	Enabled	Yes
Monitoring	Enabled	Yes
Maintenance	Auto minor version upgrade enabled	Yes
Delete protection	Not enabled	Yes

您可以检查在启用 Easy create (轻松创建) 时使用的默认设置。可在创建数据库后编辑列显示在创建数据库后可以更改的选项。

- 如果该列中的某个设置为否，而您想要不同的设置，则可以使用标准创建来创建数据库集群。
- 如果某个设置在该列中为是，而您想要不同的设置，您可以使用标准创建来创建数据库集群，也可以在创建后修改数据库集群以更改该设置。

12. 选择创建数据库。

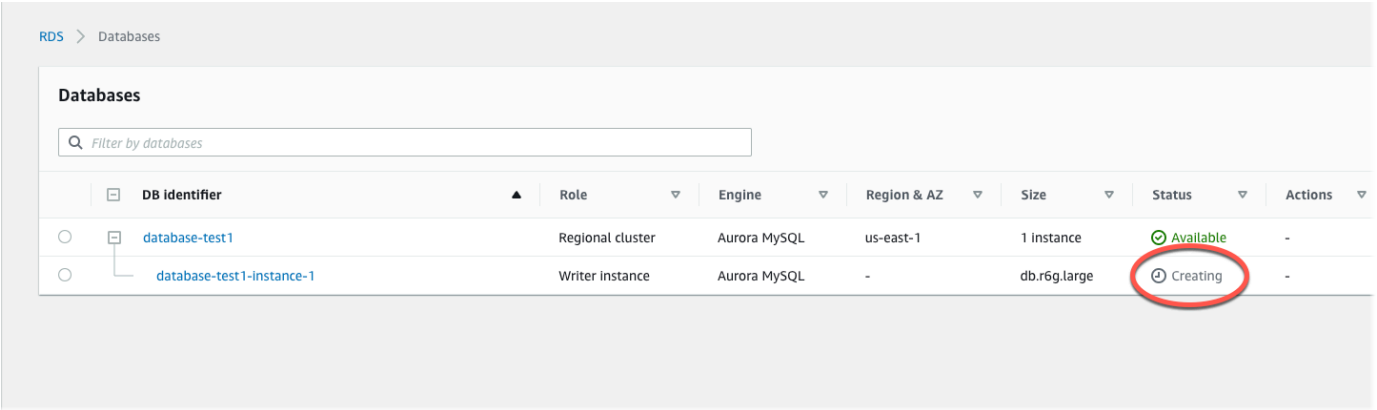
    要查看数据库集群的主用户名和密码，请选择查看凭证详细信息。

    您可以使用显示的用户名和密码，以主用户身份连接到数据库集群。

    > **Important**
    >
    > 您无法再次查看主用户密码。如果您不记录它，您可能需要更改它。
    > 如果需要在数据库集群可用后更改主用户密码，则可以修改数据库集群以执行此操作。有关修改数据库集群的更多信息，请参阅 修改 Amazon Aurora 数据库集群。

13. 在数据库列表中，选择新的 Aurora MySQL 数据库集群的名称以显示其详细信息。

    写入器实例具有正在创建状态，直到此数据库集群就绪可供使用。



    当写入器实例的状态变为可用时，您便可以连接到该数据库集群。根据数据库实例类和存储量，新数据库集群可能需要等待 20 分钟时间才可用。

（可选）使用 AWS CloudFormation 创建 VPC、EC2 实例和 Aurora MySQL 集群

您可以通过将基础设施视为代码来使用 AWS CloudFormation 配置 AWS 资源，而无需使用控制台创建 VPC、EC2 实例和 Aurora MySQL 数据库集群。为了帮助您将 AWS 资源组织成更小、更易

于管理的单元，您可以使用 AWS CloudFormation 嵌套堆栈功能。有关更多信息，请参阅在 AWS CloudFormation 控制台上创建堆栈和使用嵌套堆栈。

> **Important**
>
> AWS CloudFormation 是免费的，但 CloudFormation 创建的资源是实时的。您需要为这些资源支付标准使用费，直到您终止使用它们为止。有关更多信息，请参阅 Amazon Aurora 定价。

要使用 AWS CloudFormation 控制台创建资源，请执行以下步骤：

- 步骤 1：下载 CloudFormation 模板
- 步骤 2：使用 CloudFormation 配置资源

下载 CloudFormation 模板

CloudFormation 模板是一个 JSON 或 YAML 文本文件，其中包含有关您希望在堆栈中创建的资源的配置信息。此模板还会为您创建 VPC 和堡垒主机以及 Aurora 集群。

要下载模板文件，请打开以下链接：Aurora MySQL CloudFormation template。

在 Github 页面中，单击 Download raw file 按钮以保存模板 YAML 文件。

使用 CloudFormation 配置资源

> **Note**
>
> 在开始此过程之前，请确保您的 AWS 账户中具有 EC2 实例密钥对。有关更多信息，请参阅 Amazon EC2 密钥对和 Linux 实例。

使用 AWS CloudFormation 模板时，必须选择正确的参数以确保正确创建资源。按以下步骤操作：

1. 登录到 AWS Management Console 并打开 AWS CloudFormation 控制台 https://console.aws.amazon.com/cloudformation。
2. 选择创建堆栈。
3. 在“指定模板”部分，选择从您的计算机上传模板文件，然后选择下一步。
4. 在指定堆栈详细信息页面上，设置以下参数：
   a. 将堆栈名称设置为 AurMySQLTestStack。

b. 在参数下，通过选择两个可用区来设置可用区。

c. 在 Linux 堡垒主机配置下，在密钥名称中，选择用于登录您的 EC2 实例的密钥对。

d. 在 Linux 堡垒主机配置设置中，将允许的 IP 范围设置为您的 IP 地址。要使用 Secure Shell（SSH）连接到 VPC 中的 EC2 实例，请通过 https://checkip.amazonaws.com 上的服务确定您的公有 IP 地址。IP 地址的示例为 192.0.2.1/32。

> **Warning**
>
> 如果您使用 `0.0.0.0/0` 进行 SSH 访问，则所有 IP 地址可能能够使用 SSH 访问您的公有 EC2 实例。在测试环境下短时间内，此方法尚可接受，但它对于生产环境并不安全。在生产环境中，将仅向特定 IP 地址或地址范围授权使用 SSH 访问您的 EC2 实例。

e. 在数据库常规配置下，将数据库实例类设置为 db.r6g.large。

f. 将数据库名称设置为 **`database-test1`**。

g. 在数据库主用户名中，输入主用户的名称。

h. 在本教程中，将使用 Secrets Manager 管理数据库主用户密码设置为 `false`。

i. 在数据库密码中，设置所选密码。请记住此密码以便在教程中的后续步骤中使用。

j. 将多可用区部署设置为 `false`。

k. 将所有其他设置保留为默认值。单击下一步继续。

5. 在配置堆栈选项页面中，保留所有默认选项。单击下一步继续。

6. 在查看堆栈页面中，在检查数据库和 Linux 堡垒主机选项后，选择提交。

堆栈创建过程完成后，查看名为 BastionStack 和 AMSNS 的堆栈，记下连接到数据库所需的信息。有关更多信息，请参阅在 AWS Management Console 上查看 AWS CloudFormation 堆栈数据和资源。

步骤 3：连接到 Aurora MySQL 数据库集群

您可以使用任何标准 SQL 客户端应用程序连接到数据库集群。在该示例中，您使用 mysql 命令行客户端连接到 Aurora MySQL 数据库集群。

连接到 Aurora MySQL 数据库集群

1. 找到您的数据库集群的写入器实例的端点（DNS 名称）和端口号。

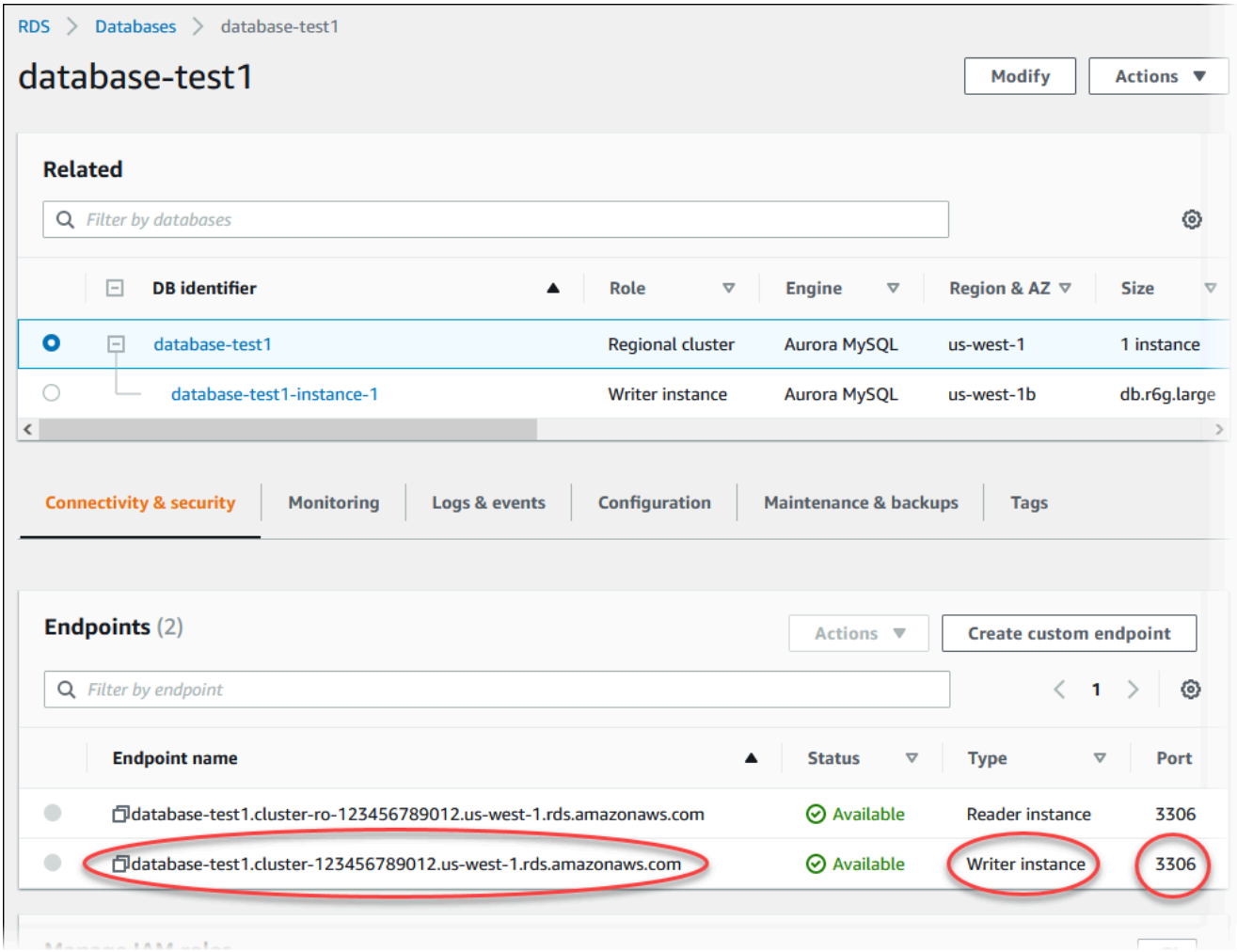
   a. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。

b. 在 Amazon RDS 控制台的右上角，选择数据库集群的 AWS 区域。

c. 在导航窗格中，选择数据库。

d. 选择 Aurora MySQL 数据库集群名称以显示其详细信息。

e. 在连接和安全性选项卡上，复制写入器实例的端点。另请注意端口号。您需要端点和端口号才能连接到数据库集群。



2. 按照《Amazon EC2 用户指南》的连接到您的 Linux 实例中的步骤，连接到您之前创建的 EC2 实例。

   建议您使用 SSH 连接到 EC2 实例。如果 SSH 客户端实用程序安装在 Windows、Linux 或 Mac 上，则可以使用以下命令格式连接到该实例：

   ```
   ssh -i location_of_pem_file ec2-user@ec2-instance-public-dns-name
   ```

例如，假设在 Linux 上 ec2-database-connect-key-pair.pem 存储在 /dir1 中，而 EC2 实例的公有 IPv4 DNS 为 ec2-12-345-678-90.compute-1.amazonaws.com。那么，SSH 命令将如下所示：

```
ssh -i /dir1/ec2-database-connect-key-pair.pem ec2-
user@ec2-12-345-678-90.compute-1.amazonaws.com
```

3. 通过更新 EC2 实例上的软件，获取最新的错误修复和安全更新。为此，请使用以下命令。

> **Note**
>
> -y 选项安装更新时不提示确认。要在安装前检查更新，请忽略该选项。

```
sudo dnf update -y
```

4. 要在 Amazon Linux 2023 上安装 MariaDB 中的 mysql 命令行客户端，请运行以下命令：

```
sudo dnf install mariadb105
```

5. 连接到 Aurora MySQL 数据库集群。例如，输入以下命令。此操作可让您使用 MySQL 客户端连接到 Aurora MySQL 数据库集群。

   将 *endpoint* 替换为写入器实例的端点，并将 *admin* 替换为您使用的主用户名。提示输入密码时，提供所使用的主密码。

```
mysql -h endpoint -P 3306 -u admin -p
```

   在输入用户的密码后，您应该会看到类似于以下内容的输出。

```
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 217
Server version: 8.0.23 Source distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MySQL [(none)]>
```

有关连接到 Aurora MySQL 数据库集群的更多信息，请参阅连接到 Amazon Aurora MySQL 数据库集群。如果您无法连接到数据库集群，请参阅无法连接到 Amazon RDS 数据库实例。

出于安全考虑，最佳做法是使用加密连接。仅当客户端和服务器位于同一 VPC 中，并且网络受信任时，才会使用未加密的 MySQL 连接。有关使用加密连接的信息，请参阅 使用 SSL 连接到 Aurora MySQL。

6. 运行 SQL 命令。

   例如，以下 SQL 命令显示了当前日期和时间：

```
SELECT CURRENT_TIMESTAMP;
```

步骤 4：删除 EC2 实例和数据库集群

在连接到您创建的示例 EC2 实例和数据库集群并进行浏览之后，删除它们，以便不再为其付费。

如果您使用 AWS CloudFormation 创建了资源，请跳过此步骤，转至下一步。

删除 EC2 实例

1. 登录到 AWS Management Console 并打开 Amazon EC2 控制台（https://console.aws.amazon.com/ec2/）。
2. 在导航窗格中，选择 Instances (实例)。
3. 选择 EC2 实例，然后依次选择实例状态、终止实例。
4. 当系统提示您确认时，选择终止。

有关删除 EC2 实例的更多信息，请参阅《Amazon EC2 用户指南》中的终止实例。

删除数据库集群

1. 登录AWS Management Console并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 选择 Databases (数据库)，然后选择与数据库集群关联的数据库实例。
3. 对于 Actions，选择 Delete。
4. 清除是否创建最终快照？。
5. 完成确认并选择删除。

删除与数据库集群关联的所有数据库实例后，将自动删除数据库集群。

（可选）删除使用 CloudFormation 创建的 EC2 实例和数据库集群

如果您使用 AWS CloudFormation 创建了资源，请在连接并浏览 EC2 实例和数据库集群示例之后，删除 CloudFormation 堆栈，以便不再为其付费。

删除 CloudFormation 资源

1. 打开 AWS CloudFormation 控制台。
2. 在 CloudFormation 控制台的堆栈页面上，选择根堆栈（名称不是 VPCStack、BastionStack 或 AMSNS 的堆栈）。
3. 选择删除。
4. 提示进行确认时，选择删除堆栈。

有关如何删除 CloudFormation 中堆栈的信息，请参阅《AWS CloudFormation 用户指南》中的在 AWS CloudFormation 控制台上删除堆栈。

（可选）将您的数据库集群连接到 Lambda 函数

您也可以将您的 Aurora MySQL 数据库集群连接到 Lambda 无服务器计算资源。Lambda 函数支持在不预调配或管理基础设施的情况下运行代码。Lambda 函数还支持自动响应任何规模的代码执行请求，从每天十几个事件到每秒数百个事件。有关更多信息，请参阅 自动连接 Lambda 函数和 Aurora 数据库集群。

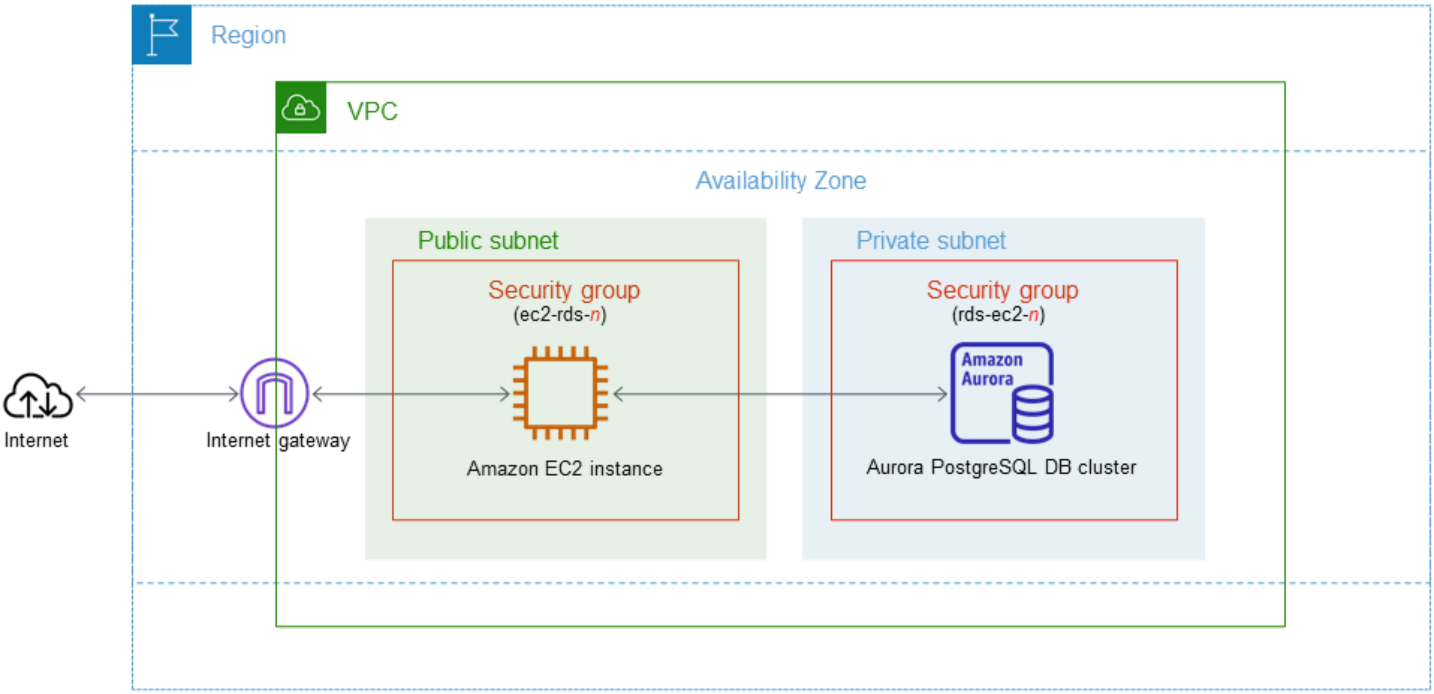
创建 Aurora PostgreSQL 数据库集群并连接到该集群

本教程创建一个 EC2 实例和一个 Aurora PostgreSQL 数据库集群。本教程向您展示如何使用标准 PostgreSQL 客户端从 EC2 实例访问数据库集群。作为最佳实践，本教程在虚拟私有云（VPC）中创建私有数据库集群。在大多数情况下，同一 VPC 中的其他资源（例如 EC2 实例）可以访问数据库集群，但 VPC 之外的资源无法访问该集群。

完成本教程后，VPC 的每个可用区中都有一个公有和私有子网。在一个可用区中，EC2 实例在公有子网中，数据库实例在私有子网中。

> **Important**
>
> 创建 AWS 账户并不会收费；但是，在完成本教程过程中使用AWS资源可能会产生费用。完成本教程后，如果不再需要这些资源，可以将其删除。

下图显示了教程完成时的配置。



本教程可帮助您通过以下方法之一创建资源：

1. 使用 AWS Management Console - 步骤 1：创建 EC2 实例 和 步骤 2：创建 Aurora PostgreSQL 数据库集群
2. 使用 AWS CloudFormation 创建数据库实例和 EC2 实例 - （可选）使用 AWS CloudFormation 创建 VPC、EC2 实例和 Aurora PostgreSQL 集群

第一种方法使用轻松创建，通过 AWS Management Console 创建私有 Aurora PostgreSQL 数据库集群。您可以仅指定数据库引擎类型、数据库实例大小和数据库集群标识符。轻松创建为其他配置选项使用默认设置。

如果使用标准创建，则可以在创建数据库集群时指定更多配置选项。这些选项包括可用性、安全性、备份和维护设置。要创建公有数据库集群，必须使用标准创建。有关信息，请参阅the section called “创建数据库集群”。

主题

- 先决条件
- 步骤 1：创建 EC2 实例
- 步骤 2：创建 Aurora PostgreSQL 数据库集群
- （可选）使用 AWS CloudFormation 创建 VPC、EC2 实例和 Aurora PostgreSQL 集群
- 步骤 3：连接到 Aurora PostgreSQL 数据库集群
- 步骤 4：删除 EC2 实例和数据库集群
- （可选）删除使用 CloudFormation 创建的 EC2 实例和数据库集群
- （可选）将您的数据库集群连接到 Lambda 函数

先决条件

在开始之前，请完成以下各节中的步骤：

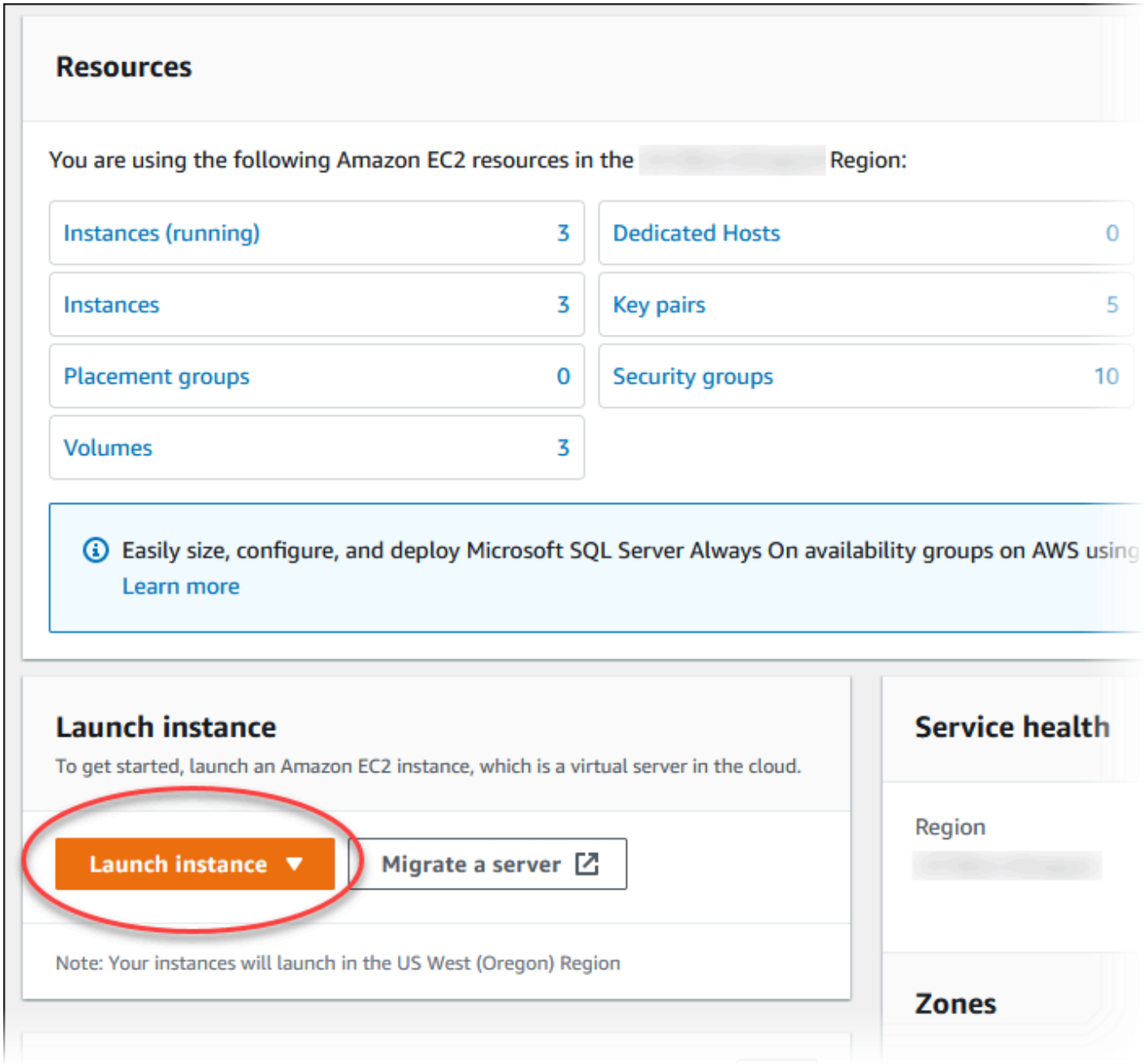
- 注册 AWS 账户
- 创建具有管理访问权限的用户

步骤 1：创建 EC2 实例

创建将用于连接到数据库的 Amazon EC2 实例。

创建 EC2 实例

1. 登录到 AWS Management Console 并打开 Amazon EC2 控制台（https://console.aws.amazon.com/ec2/）。
2. 在 AWS Management Console的右上角，选择要在其中创建 EC2 实例的 AWS 区域。
3. 选择 EC2 控制面板，然后选择启动实例，如下图所示。

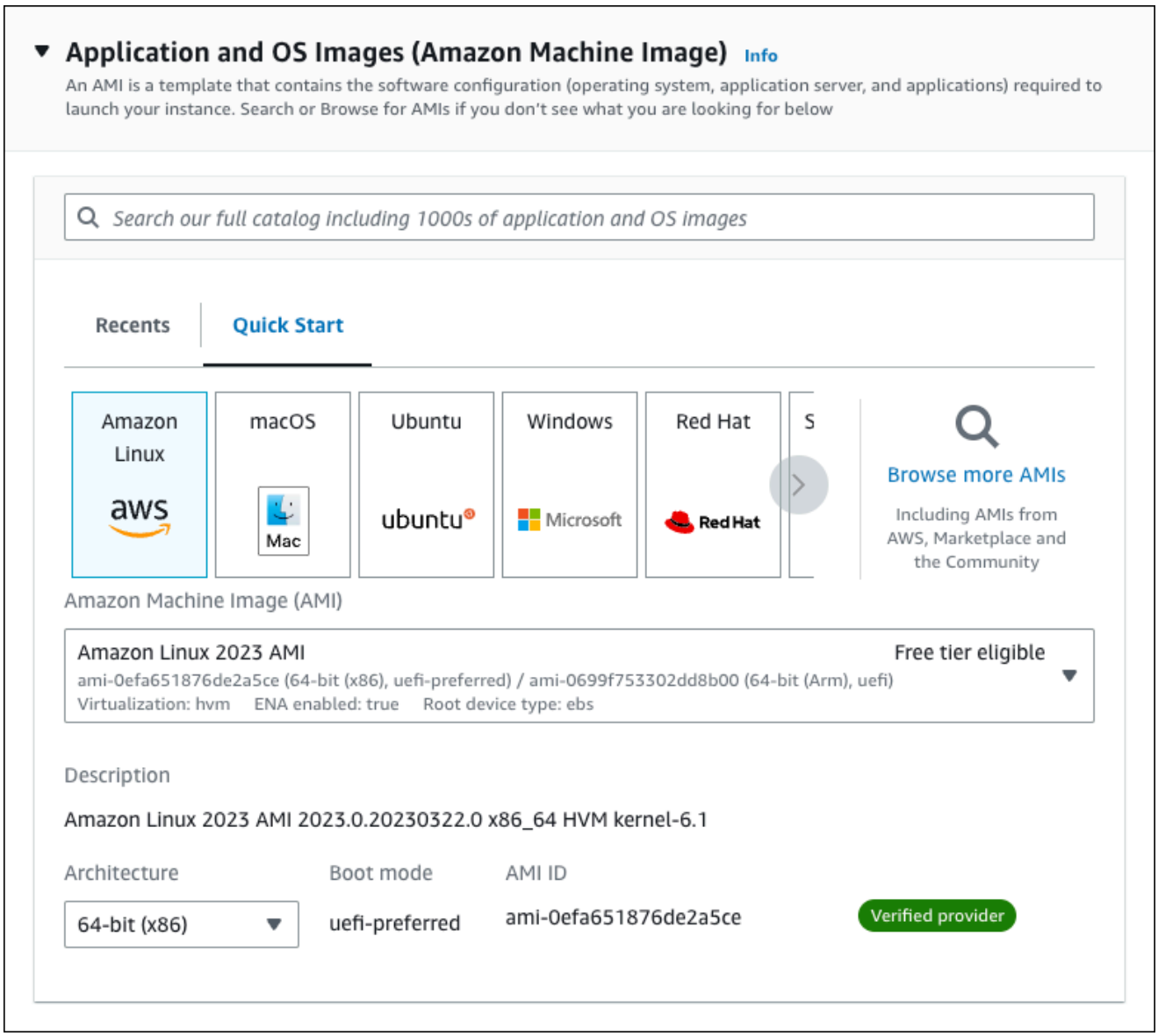


启动实例页面打开。

4. 在启动实例页面上选择以下设置。

   a. 在 Name and tags（名称和标签）下，对于 Name（名称），输入 **`ec2-database-connect`**。

   b. 在应用程序和操作系统映像（Amazon 机器映像）下，选择 Amazon Linux，然后选择 Amazon Linux 2023 AMI。对于其他选项，保留默认选择。



c. 在 Instance type（实例类型）下，选择 t2.micro。

d. 在 Key pair (login) [密钥对（登录）] 下，选择 Key pair name（密钥对名称）以使用现有密钥对。要为 Amazon EC2 实例创建新的密钥对，请选择 Create new key pair（创建新的密钥对），然后使用 Create key pair（创建密钥对）窗口来创建它。

   有关创建新的密钥对的更多信息，请参阅《Amazon EC2 用户指南》中的创建密钥对。

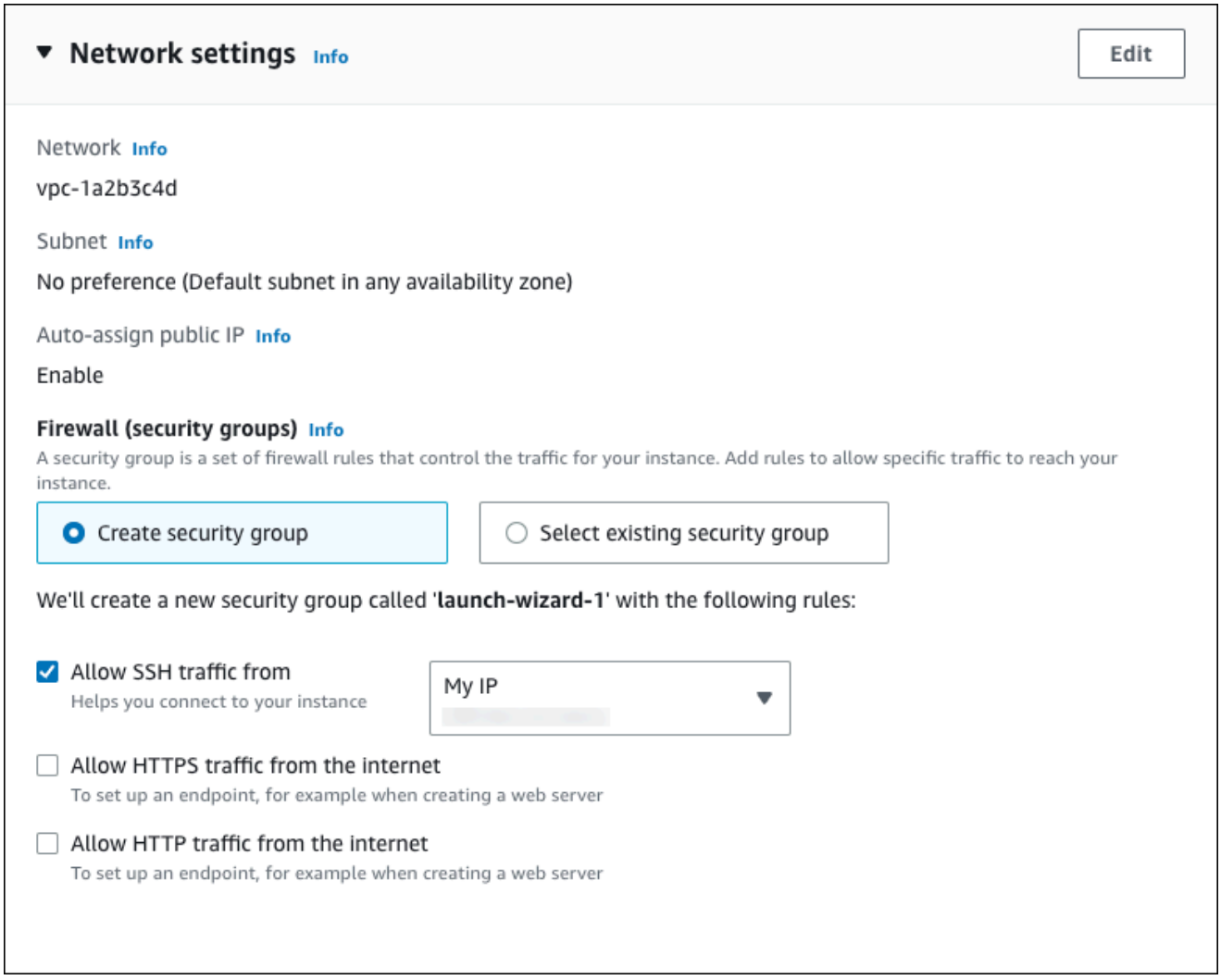
e. 对于网络设置中的允许 SSH 流量，选择 EC2 实例的 SSH 连接来源。

   如果显示的 IP 地址对于 SSH 连接是正确的，您可以选择 My IP（我的 IP）。否则，您可以确定要用来通过 Secure Shell（SSH）连接到 VPC 中的 EC2 实例的 IP 地址。要确定您的公有 IP 地址，请在新的浏览器窗口或标签页中，使用 https://checkip.amazonaws.com 上的服务。IP 地址的示例为 192.0.2.1/32。

在许多情况下，您可能通过互联网服务提供商（ISP）进行连接，或者在不使用静态 IP 地址的情况下从防火墙之后进行连接。如果是这样，请确保确定客户端计算机使用的 IP 地址范围。

> **Warning**
>
> 如果您使用 `0.0.0.0/0` 进行 SSH 访问，则所有 IP 地址可能能够使用 SSH 访问您的公有 EC2 实例。在测试环境下短时间内，此方法尚可接受，但它对于生产环境并不安全。在生产环境中，将仅向特定 IP 地址或地址范围授权使用 SSH 访问您的 EC2 实例。

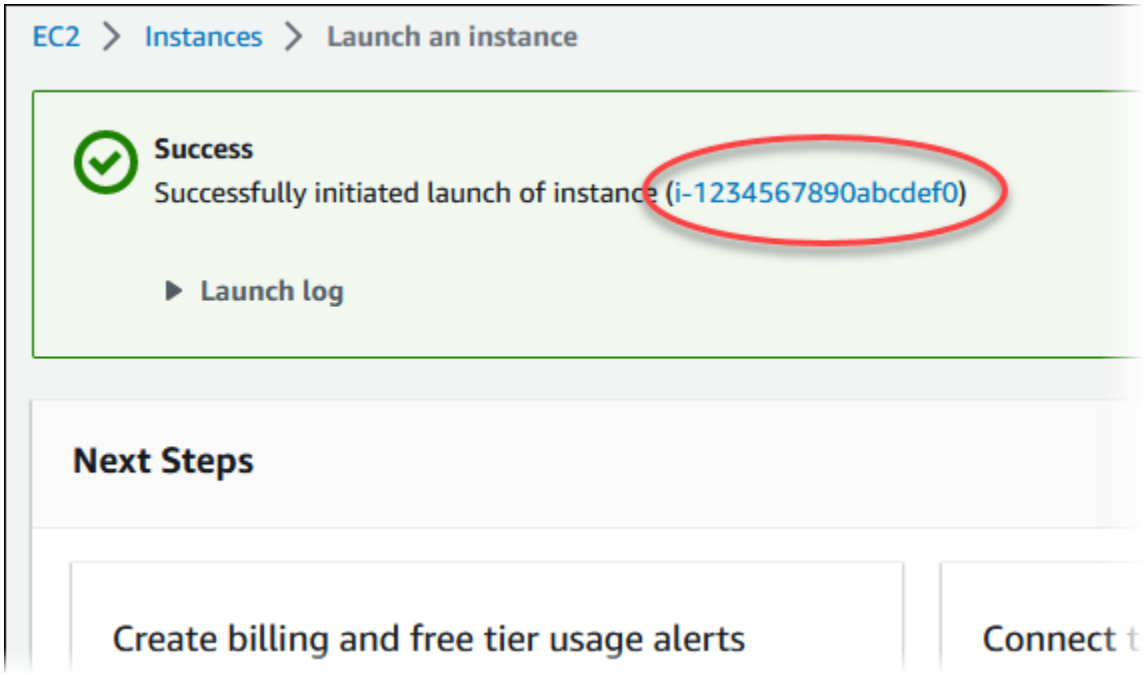
下图显示了网络设置部分的示例。



f. 对于其余部分保留默认值。

g. 查看摘要面板中您的 EC2 实例配置的摘要，当您准备好后，选择启动实例。

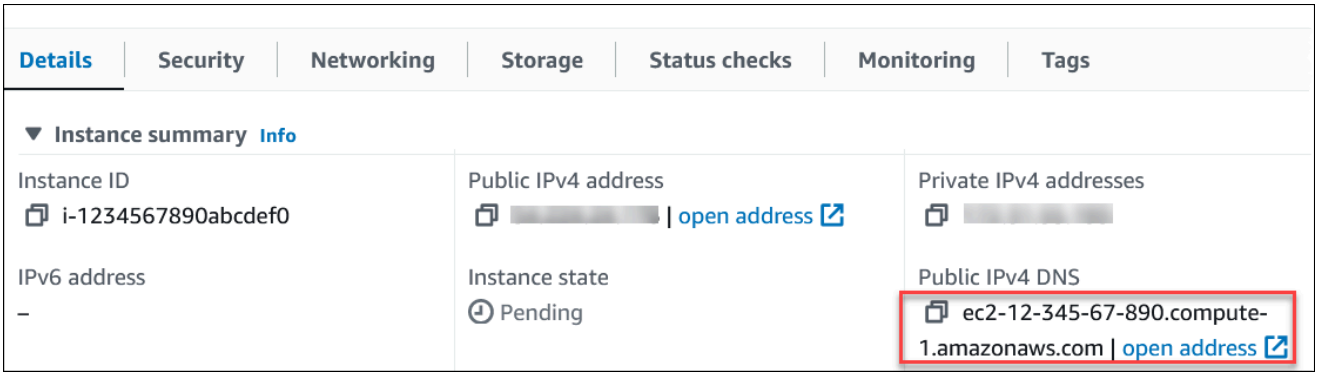
5. 在启动状态页面上，记下新 EC2 实例的标识符，例如：`i-1234567890abcdef0`。



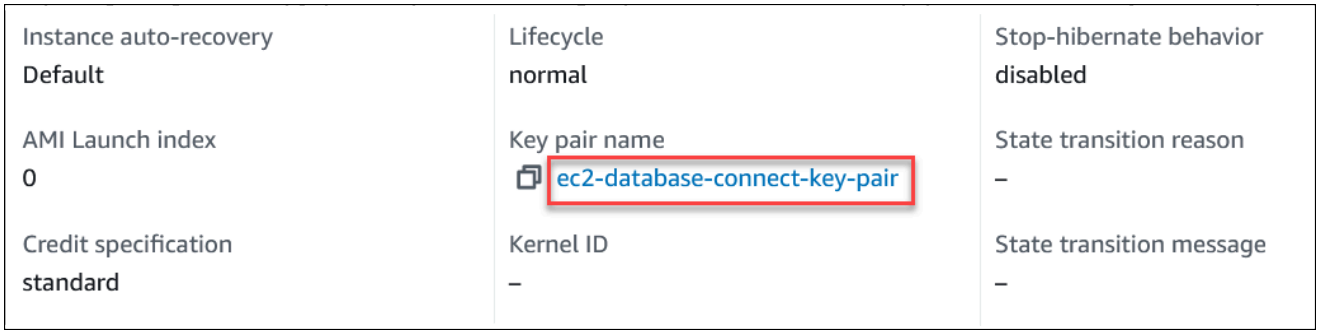
6. 选择 EC2 实例标识符以打开 EC2 实例列表，然后选择您的 EC2 实例。

7. 在详细信息选项卡中，记下使用 SSH 进行连接时所需的以下值：

   a. 在实例摘要中，记下公有 IPv4 DNS 的值。



   b. 在实例详细信息中，记下密钥对名称的值。



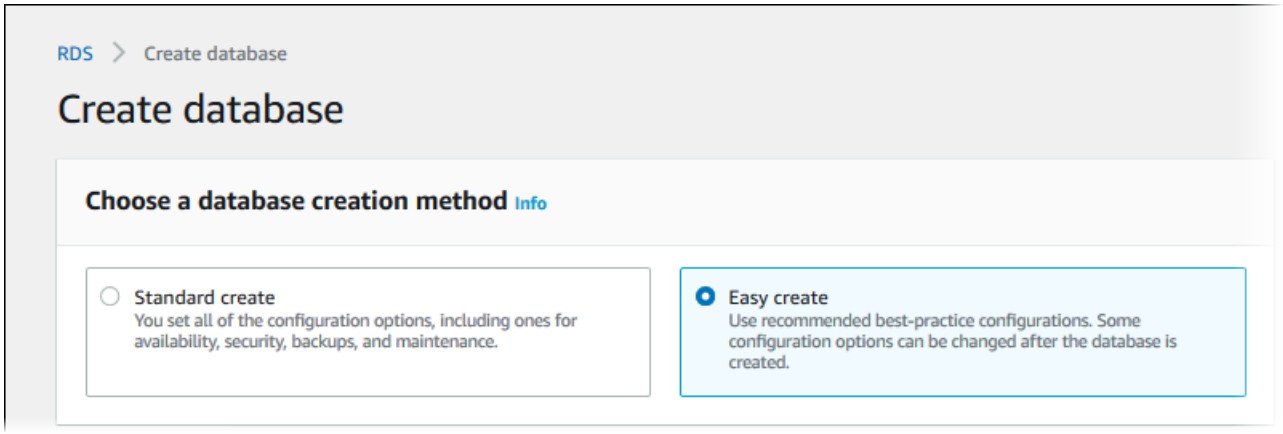
8. 等待 EC2 实例的实例状态变为正在运行，然后继续。

步骤 2：创建 Aurora PostgreSQL 数据库集群

在该示例中，您使用轻松创建来创建一个具有 db.t4g.large 数据库实例类的 Aurora PostgreSQL 数据库集群。

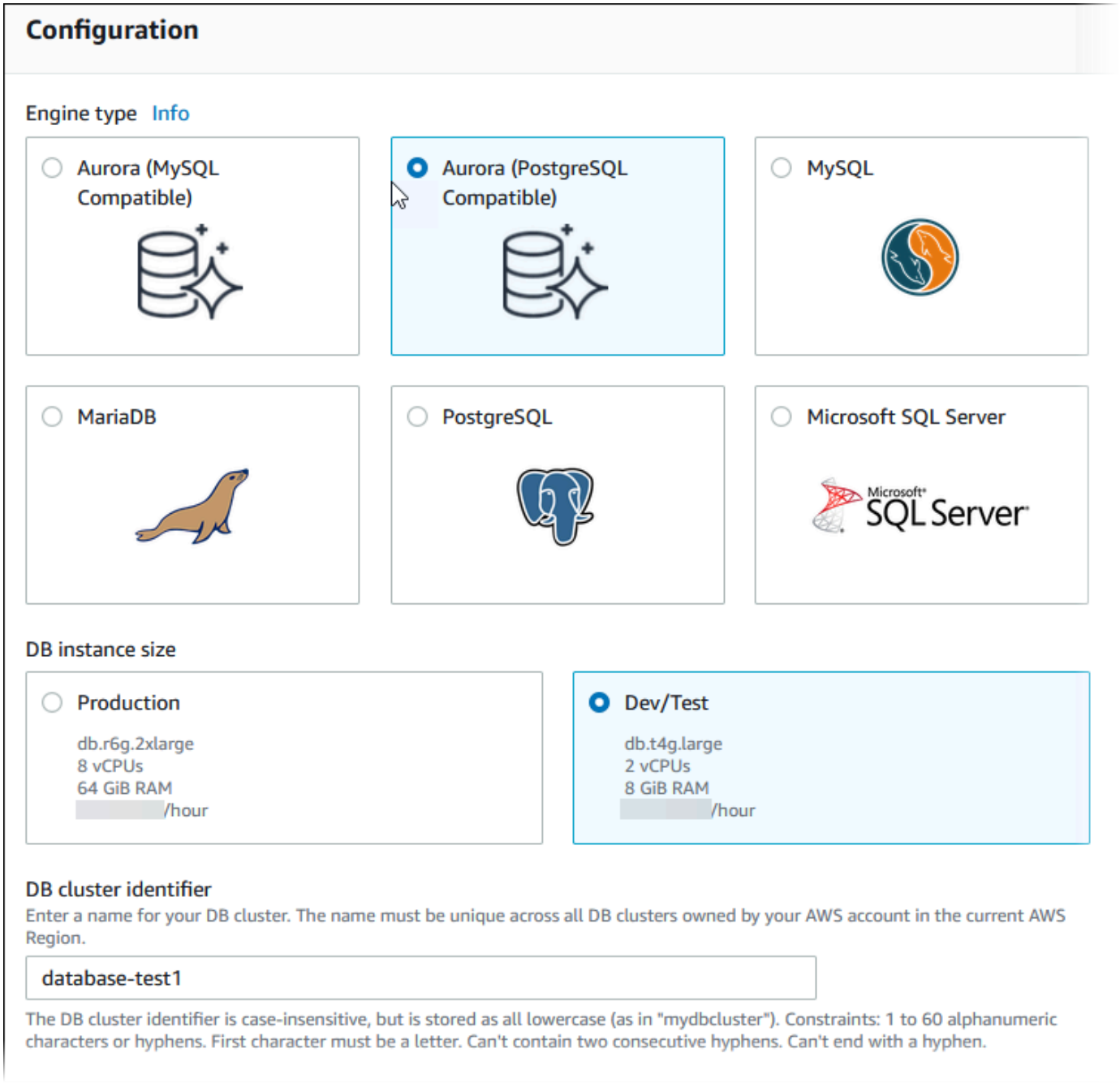
使用“轻松创建”来创建 Aurora PostgreSQL 数据库集群

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在 Amazon RDS 控制台的右上角，选择要在其中创建数据库集群的 AWS 区域。
3. 在导航窗格中，选择数据库。
4. 选择创建数据库，并确保已选择轻松创建。



5. 在配置中，对于引擎类型，选择 Aurora（PostgreSQL 兼容）。
6. 对于数据库实例大小，选择设备/测试。
7. 对于数据库集群标识符，输入 **`database-test1`**。

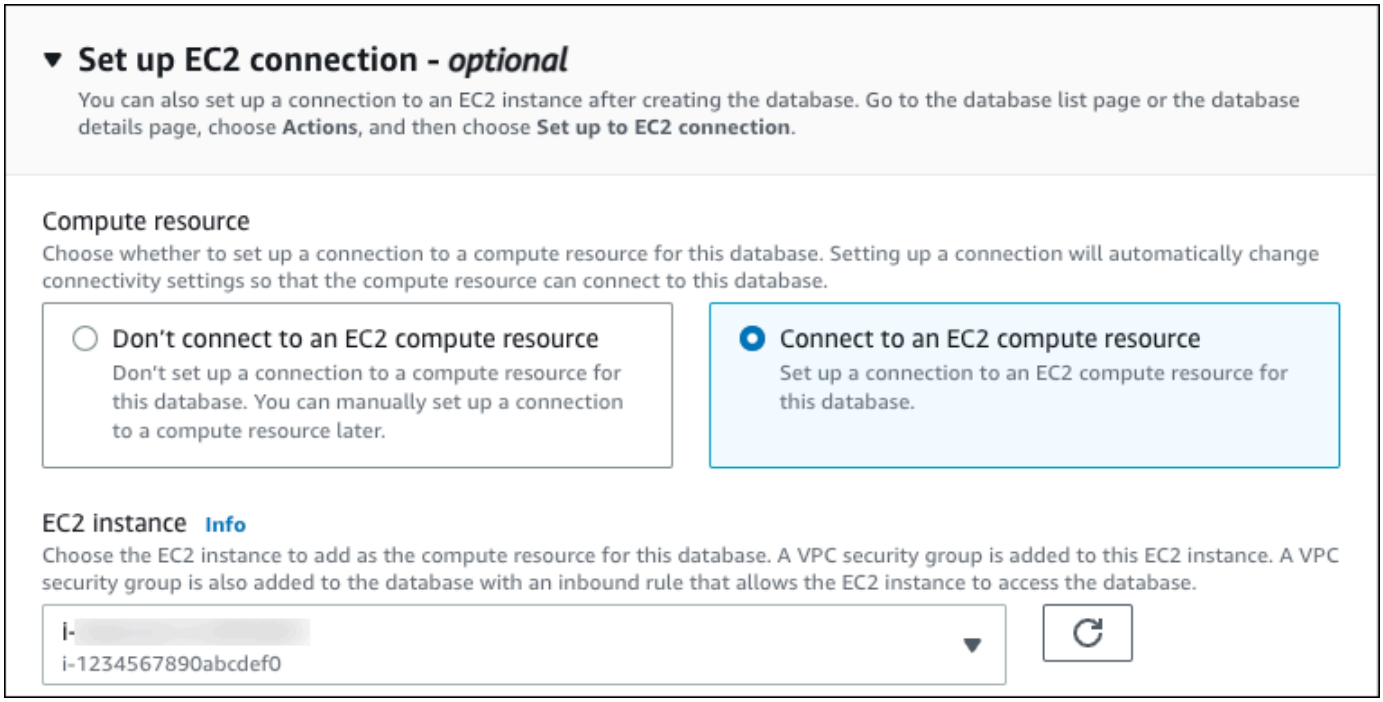
   创建数据库页面应类似于以下图像。



8. 对于主用户名，输入用户的名称，或者保留默认名称（**`postgres`**）。
9. 要为数据库集群使用自动生成的主密码，请选择自动生成密码。

   要输入主密码，请确保清除自动生成密码，然后在主密码和确认密码中输入相同的密码。
10. 要设置与您之前创建的 EC2 实例的连接，请打开设置 EC2 连接 - 可选。

    选择连接到 EC2 计算资源。选择您之前创建的 EC2 实例。



11. 打开查看轻松创建的默认设置。

View default settings for Easy create

Easy create sets the following configurations to their default values, some of which can be changed later. If you want to change any of these settings now, use **Standard create**.

Configuration	Value	Editable after database is created
Encryption	Enabled	No
VPC	Default VPC (vpc-1a2b3c4d)	No
Option group	default:aurora-postgresql-13	No
Subnet group	create-subnet-group	Yes
Automatic backups	Enabled	Yes
VPC security group	sg-1234567	Yes
Publicly accessible	No	Yes
Database port	5432	Yes
DB cluster identifier	database-test1	Yes
DB instance identifier	database-1	Yes
DB engine version	13.6	Yes
DB parameter group	default.aurora-postgresql13	Yes
DB cluster parameter group	default.aurora-postgresql13	Yes
Performance insights	Enabled	Yes
Monitoring	Enabled	Yes
Maintenance	Auto minor version upgrade enabled	Yes
Delete protection	Not enabled	Yes

您可以检查在启用 Easy create (轻松创建) 时使用的默认设置。可在创建数据库后编辑列显示在创建数据库后可以更改的选项。

- 如果该列中的某个设置为否，而您想要不同的设置，则可以使用标准创建来创建数据库集群。
- 如果某个设置在该列中为是，而您想要不同的设置，您可以使用标准创建来创建数据库集群，也可以在创建后修改数据库集群以更改该设置。

12. 选择创建数据库。

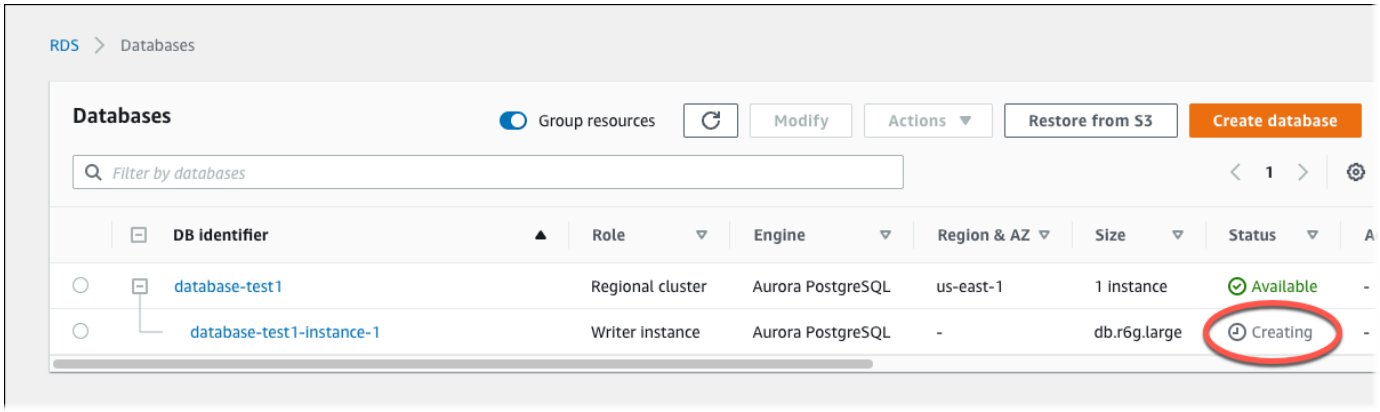
    要查看数据库集群的主用户名和密码，请选择查看凭证详细信息。

    您可以使用显示的用户名和密码，以主用户身份连接到数据库集群。

    > **Important**
    >
    > 您无法再次查看主用户密码。如果您不记录它，您可能需要更改它。
    > 如果需要在数据库集群可用后更改主用户密码，则可以修改数据库集群以执行此操作。有关修改数据库集群的更多信息，请参阅 修改 Amazon Aurora 数据库集群。

13. 在数据库列表中，选择新的 Aurora PostgreSQL 数据库集群的名称以显示其详细信息。

    写入器实例具有正在创建状态，直到此数据库集群就绪可供使用。



当写入器实例的状态变为可用时，您便可以连接到该数据库集群。根据数据库实例类和存储量，新数据库集群可能需要等待 20 分钟时间才可用。

（可选）使用 AWS CloudFormation 创建 VPC、EC2 实例和 Aurora PostgreSQL 集群

您可以通过将基础设施视为代码来使用 AWS CloudFormation 配置 AWS 资源，而无需使用控制台创建 VPC、EC2 实例和 Aurora PostgreSQL 数据库集群。为了帮助您将 AWS 资源组织成更小、更易于管理的单元，您可以使用 AWS CloudFormation 嵌套堆栈功能。有关更多信息，请参阅在 AWS CloudFormation 控制台上创建堆栈和使用嵌套堆栈。

> **Important**
>
> AWS CloudFormation 是免费的，但 CloudFormation 创建的资源是实时的。您需要为这些资源支付标准使用费，直到您终止使用它们为止。有关更多信息，请参阅 Amazon Aurora 定价。

要使用 AWS CloudFormation 控制台创建资源，请执行以下步骤：

- 步骤 1：下载 CloudFormation 模板
- 步骤 2：使用 CloudFormation 配置资源

下载 CloudFormation 模板

CloudFormation 模板是一个 JSON 或 YAML 文本文件，其中包含有关您希望在堆栈中创建的资源的配置信息。此模板还会为您创建 VPC 和堡垒主机以及 Aurora 集群。

要下载模板文件，请打开以下链接：Aurora PostgreSQL CloudFormation template。

在 Github 页面中，单击 Download raw file 按钮以保存模板 YAML 文件。

使用 CloudFormation 配置资源

> **Note**
>
> 在开始此过程之前，请确保您的 AWS 账户中具有 EC2 实例密钥对。有关更多信息，请参阅 Amazon EC2 密钥对和 Linux 实例。

使用 AWS CloudFormation 模板时，必须选择正确的参数以确保正确创建资源。按以下步骤操作：

1. 登录到 AWS Management Console 并打开 AWS CloudFormation 控制台 https://console.aws.amazon.com/cloudformation。
2. 选择创建堆栈。
3. 在“指定模板”部分，选择从您的计算机上传模板文件，然后选择下一步。
4. 在指定堆栈详细信息页面上，设置以下参数：
   a. 将堆栈名称设置为 AurPostgreSQLTestStack。
   b. 在参数下，通过选择两个可用区来设置可用区。
   c. 在 Linux 堡垒主机配置下，在密钥名称中，选择用于登录您的 EC2 实例的密钥对。
   d. 在 Linux 堡垒主机配置设置中，将允许的 IP 范围设置为您的 IP 地址。要使用 Secure Shell（SSH）连接到 VPC 中的 EC2 实例，请通过 https://checkip.amazonaws.com 上的服务确定您的公有 IP 地址。IP 地址的示例为 192.0.2.1/32。

      > **Warning**
      > 如果您使用 `0.0.0.0/0` 进行 SSH 访问，则所有 IP 地址可能能够使用 SSH 访问您的公有 EC2 实例。在测试环境下短时间内，此方法尚可接受，但它对于生产环境并不安全。在生产环境中，将仅向特定 IP 地址或地址范围授权使用 SSH 访问您的 EC2 实例。

   e. 在数据库常规配置下，将数据库实例类设置为 db.t4g.large。
   f. 将数据库名称设置为 **`database-test1`**。
   g. 在数据库主用户名中，输入主用户的名称。
   h. 在本教程中，将使用 Secrets Manager 管理数据库主用户密码设置为 `false`。
   i. 在数据库密码中，设置所选密码。请记住此密码以便在教程中的后续步骤中使用。
   j. 将多可用区部署设置为 `false`。
   k. 将所有其他设置保留为默认值。单击下一步继续。
5. 在配置堆栈选项页面中，保留所有默认选项。单击下一步继续。
6. 在查看堆栈页面中，在检查数据库和 Linux 堡垒主机选项后，选择提交。

堆栈创建过程完成后，查看名为 BastionStack 和 APGNS 的堆栈，记下连接到数据库所需的信息。有关更多信息，请参阅在 AWS Management Console 上查看 AWS CloudFormation 堆栈数据和资源。

步骤 3：连接到 Aurora PostgreSQL 数据库集群

您可以使用任何标准 PostgreSQL 客户端应用程序连接到数据库集群。在该示例中，您使用 psql 命令行客户端连接到 Aurora PostgreSQL 数据库集群。

连接到 Aurora PostgreSQL 数据库集群

1. 找到您的数据库集群的写入器实例的端点（DNS 名称）和端口号。

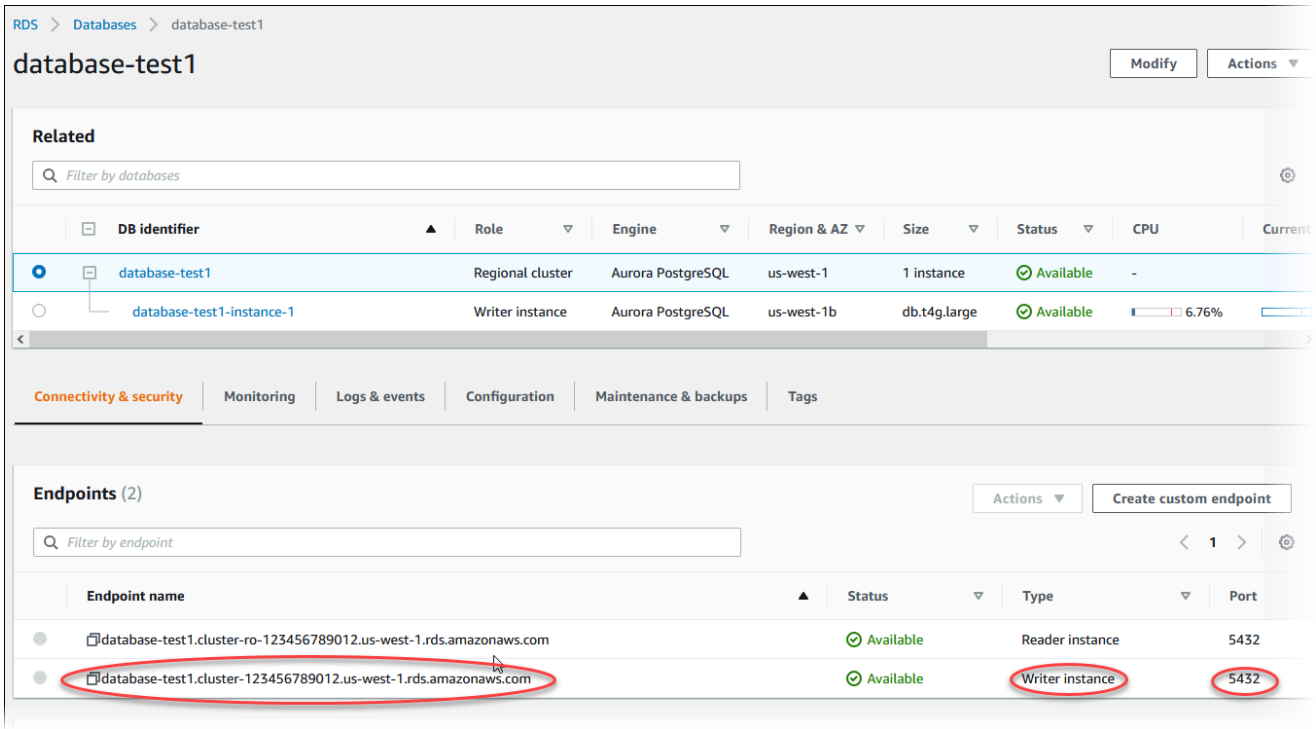
   a. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。

   b. 在 Amazon RDS 控制台的右上角，选择数据库集群的 AWS 区域。

   c. 在导航窗格中，选择数据库。

   d. 选择 Aurora PostgreSQL 数据库集群名称以显示其详细信息。

   e. 在连接和安全性选项卡上，复制写入器实例的端点。另请注意端口号。您需要端点和端口号才能连接到数据库集群。



2. 按照《Amazon EC2 用户指南》的连接到您的 Linux 实例中的步骤，连接到您之前创建的 EC2 实例。

   建议您使用 SSH 连接到 EC2 实例。如果 SSH 客户端实用程序安装在 Windows、Linux 或 Mac 上，则可以使用以下命令格式连接到该实例：

```
ssh -i location_of_pem_file ec2-user@ec2-instance-public-dns-name
```

例如，假设在 Linux 上 ec2-database-connect-key-pair.pem 存储在 /dir1 中，而 EC2 实例的公有 IPv4 DNS 为 ec2-12-345-678-90.compute-1.amazonaws.com。SSH 命令将如下所示：

```
ssh -i /dir1/ec2-database-connect-key-pair.pem ec2-
user@ec2-12-345-678-90.compute-1.amazonaws.com
```

3. 通过更新 EC2 实例上的软件，获取最新的错误修复和安全更新。为此，请使用以下命令。

> **Note**
>
> -y 选项安装更新时不提示确认。要在安装前检查更新，请忽略该选项。

```
sudo dnf update -y
```

4. 使用以下命令在 Amazon Linux 2023 上安装 PostgreSQL 中的 psql 命令行客户端：

```
sudo dnf install postgresql15
```

5. 连接到 Aurora PostgreSQL 数据库集群。例如，输入以下命令。此操作可让您使用 psql 客户端连接到 Aurora PostgreSQL 数据库集群。

   用写入器实例的端点替换 *endpoint*，用要连接到的数据库名称 --dbname 替换 *postgres*，并用您使用的主用户名替换 *postgres*。提示输入密码时，提供所使用的主密码。

```
psql --host=endpoint --port=5432 --dbname=postgres --username=postgres
```

   在输入用户的密码后，您应该会看到类似于以下内容的输出。

```
psql (14.3, server 14.6)
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256,
 compression: off)
Type "help" for help.

postgres=>
```

有关连接到 Aurora PostgreSQL 数据库集群的更多信息，请参阅连接到 Amazon Aurora PostgreSQL 数据库集群。如果您无法连接到数据库集群，请参阅无法连接到 Amazon RDS 数据库实例。

出于安全考虑，最佳做法是使用加密连接。仅当客户端和服务器位于同一 VPC 中，并且网络受信任时，才会使用未加密的 PostgreSQL 连接。有关使用加密连接的信息，请参阅 利用 SSL/TLS 保护 Aurora PostgreSQL 数据。

6. 运行 SQL 命令。

   例如，以下 SQL 命令显示了当前日期和时间：

```
SELECT CURRENT_TIMESTAMP;
```

步骤 4：删除 EC2 实例和数据库集群

在连接到您创建的示例 EC2 实例和数据库集群并进行浏览之后，删除它们，以便不再为其付费。

如果您使用 AWS CloudFormation 创建了资源，请跳过此步骤，转至下一步。

删除 EC2 实例

1. 登录到 AWS Management Console 并打开 Amazon EC2 控制台（https://console.aws.amazon.com/ec2/）。
2. 在导航窗格中，选择 Instances (实例)。
3. 选择 EC2 实例，然后依次选择实例状态、终止实例。
4. 当系统提示您确认时，选择终止。

有关删除 EC2 实例的更多信息，请参阅《Amazon EC2 用户指南》中的终止实例。

删除数据库集群

1. 登录AWS Management Console并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 选择 Databases (数据库)，然后选择与数据库集群关联的数据库实例。
3. 对于 Actions，选择 Delete。
4. 选择 Delete。

删除与数据库集群关联的所有数据库实例后，将自动删除数据库集群。

（可选）删除使用 CloudFormation 创建的 EC2 实例和数据库集群

如果您使用 AWS CloudFormation 创建了资源，请在连接并浏览 EC2 实例和数据库集群示例之后，删除 CloudFormation 堆栈，以便不再为其付费。

删除 CloudFormation 资源

1. 打开 AWS CloudFormation 控制台。
2. 在 CloudFormation 控制台的堆栈页面上，选择根堆栈（名称不是 VPCStack、BastionStack 或 APGNS 的堆栈）。
3. 选择删除。
4. 提示进行确认时，选择删除堆栈。

有关如何删除 CloudFormation 中堆栈的信息，请参阅《AWS CloudFormation 用户指南》中的在 AWS CloudFormation 控制台上删除堆栈。

（可选）将您的数据库集群连接到 Lambda 函数

您也可以将您的 Aurora PostgreSQL 数据库集群连接到 Lambda 无服务器计算资源。Lambda 函数支持在不预调配或管理基础设施的情况下运行代码。Lambda 函数还支持自动响应任何规模的代码执行请求，从每天十几个事件到每秒数百个事件。有关更多信息，请参阅 自动连接 Lambda 函数和 Aurora 数据库集群。

教程：创建 Web 服务器和 Amazon Aurora 数据库集群

本教程说明如何使用 PHP 安装 Apache Web 服务器并创建 MariaDB、MySQL 或 PostgreSQL 数据库。Web 服务器在使用 Amazon Linux 2023 的 Amazon EC2 实例上运行，您可以在 Aurora MySQL 或 Aurora PostgreSQL 数据库集群之间进行选择。Amazon EC2 实例和数据库集群均在基于 Amazon VPC 服务的 Virtual Private Cloud (VPC) 中运行。

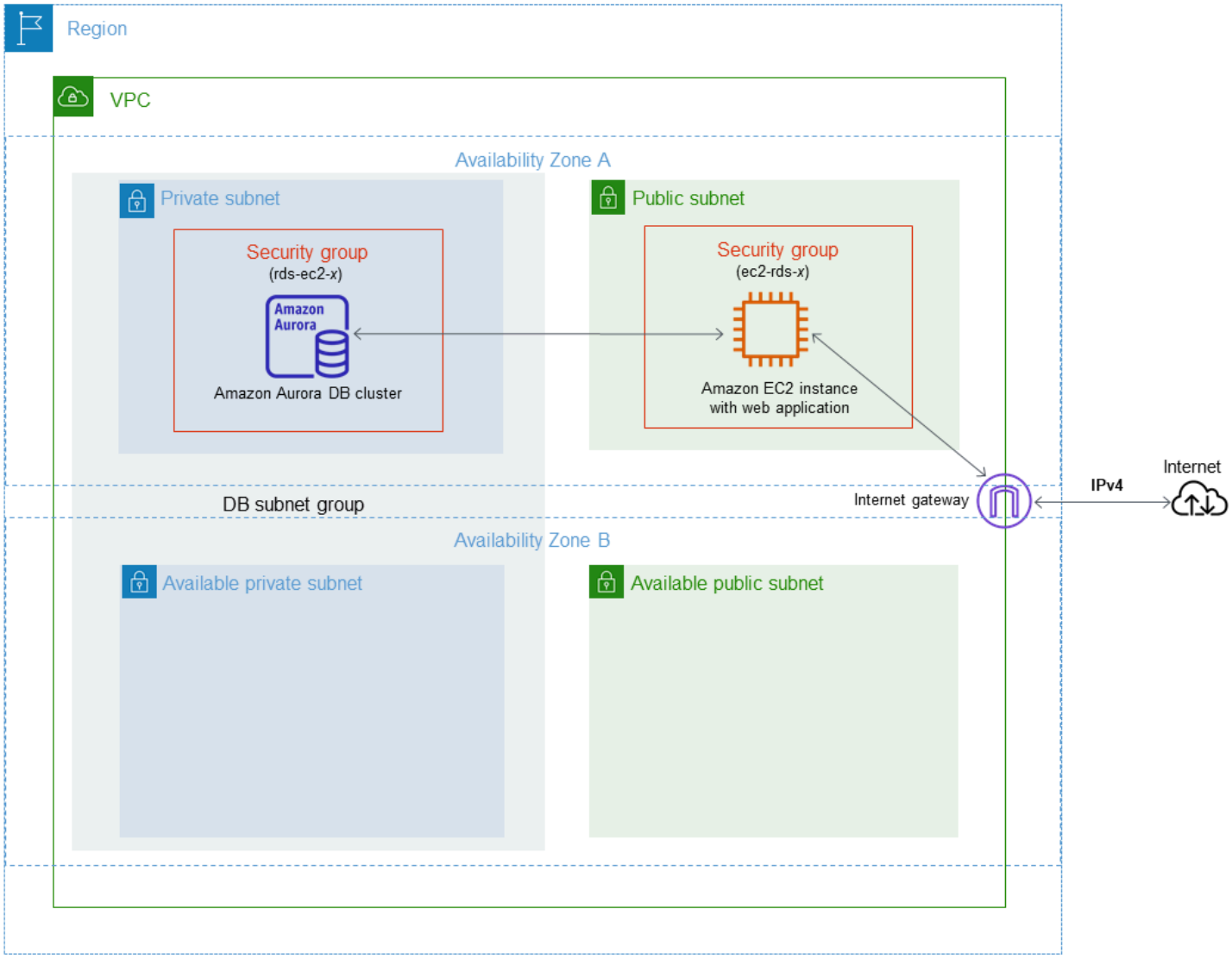
> **Important**
>
> 创建 AWS 账户并不会收费；但是，在完成本教程过程中使用 AWS 资源可能会产生费用。完成本教程后，如果不再需要这些资源，可以将其删除。

> **Note**
>
> 本教程适用于 Amazon Linux 2023，可能不适用于其他版本的 Linux。

在下面的教程中，您将创建一个对您的 AWS 账户使用默认 VPC、子网和安全组的 EC2 实例。本教程说明如何创建数据库集群，并自动设置与您创建的 EC2 实例的连接。然后，本教程向您展示如何在 EC2 实例上安装 Web 服务器。您使用数据库集群写入器端点，将 Web 服务器连接到 VPC 中的数据库集群。

1. 启动 EC2 实例以连接数据库集群
2. 创建 Amazon Aurora 数据库集群
3. 在 EC2 实例上安装 Web 服务器

下图显示了教程完成时的配置。



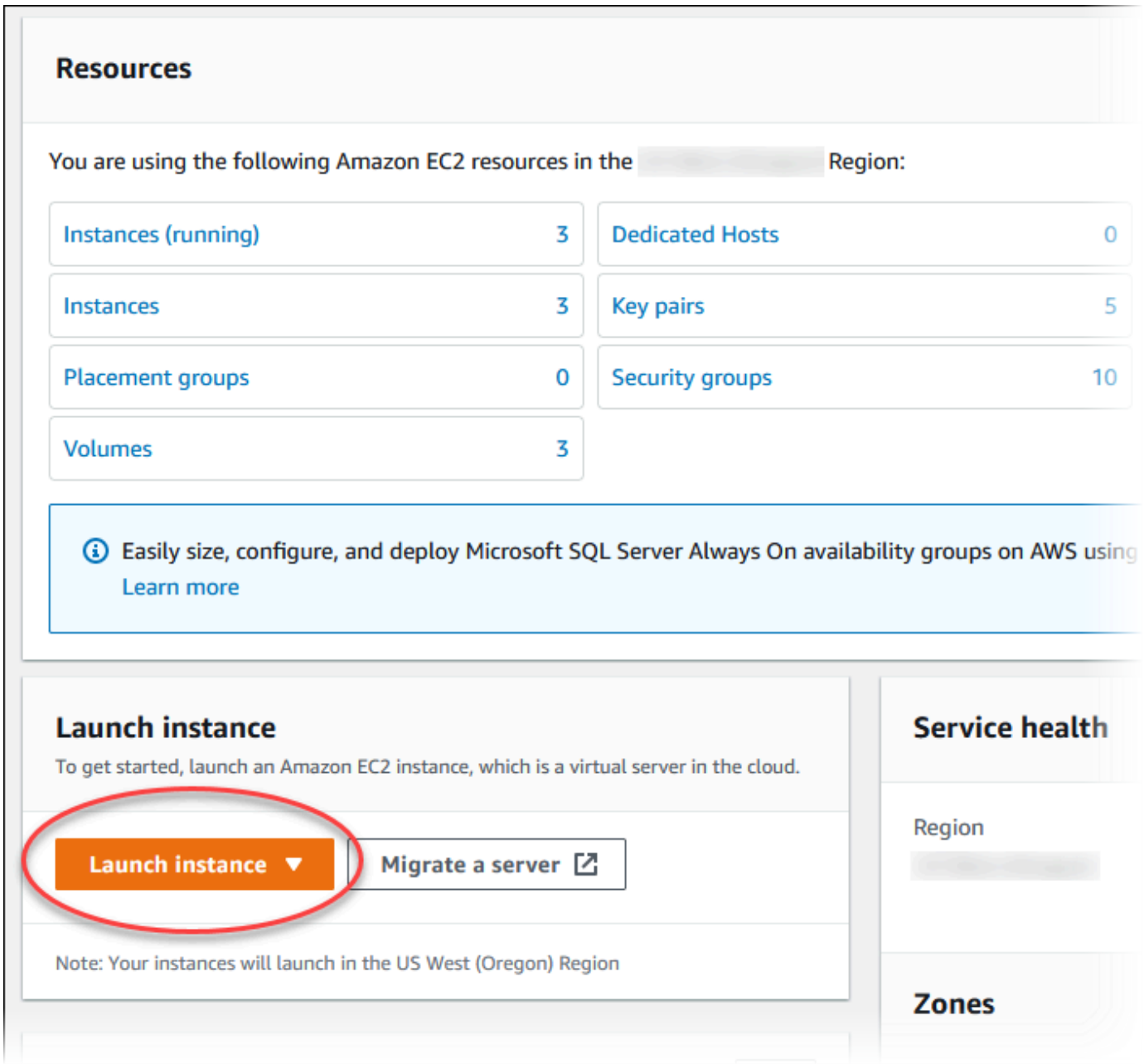
> **Note**
>
> 完成本教程后，VPC 的每个可用区中都有一个公有和私有子网。本教程对您的 AWS 账户使用默认 VPC，并自动设置您的 EC2 实例与数据库集群之间的连接。如果您更愿意为此场景配置新的 VPC，请完成教程：创建 VPC 以用于数据库集群（仅限 IPv4）中的任务。

启动 EC2 实例以连接数据库集群

在您的 VPC 的公有子网中创建 Amazon EC2 实例。

启动 EC2 实例

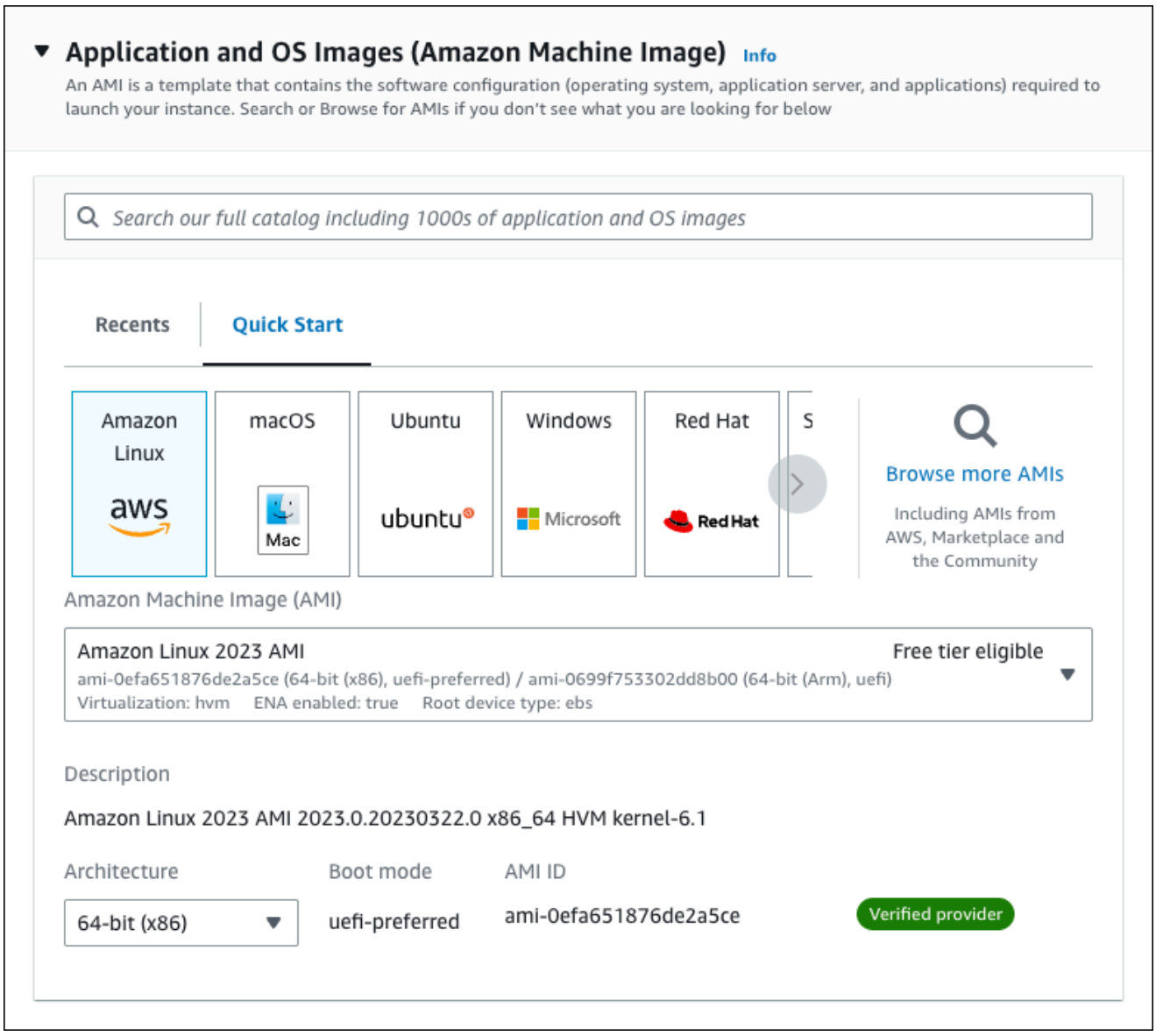
1. 登录 AWS Management Console，打开 Amazon EC2 控制台：https://console.aws.amazon.com/ec2/。
2. 在 AWS Management Console的右上角，选择要在其中创建 EC2 实例的 AWS 区域。
3. 选择 EC2 控制面板，然后选择启动实例，如下所示。



4. 在启动实例页面中选择以下设置。

a. 在 Name and tags（名称和标签）下，对于 Name（名称），输入 **`tutorial-ec2-instance-web-server`**。

b. 在应用程序和操作系统映像（Amazon 机器映像）下，选择 Amazon Linux，然后选择 Amazon Linux 2023 AMI。对于其他选项保留原定设置值。



c. 在 Instance type（实例类型）下，选择 t2.micro。

d. 在 Key pair (login) [密钥对（登录）] 下，选择 Key pair name（密钥对名称）以使用现有密钥对。要为 Amazon EC2 实例创建新的密钥对，请选择 Create new key pair（创建新的密钥对），然后使用 Create key pair（创建密钥对）窗口来创建它。

有关创建新的密钥对的更多信息，请参阅《Amazon EC2 用户指南》中的创建密钥对。

e. 在 Network settings（网络设置）下，设置以下值并将其他值保留为其原定设置值：

- 对于 Allow SSH traffic from（允许 SSH 流量来自），选择 EC2 实例的 SSH 连接来源。

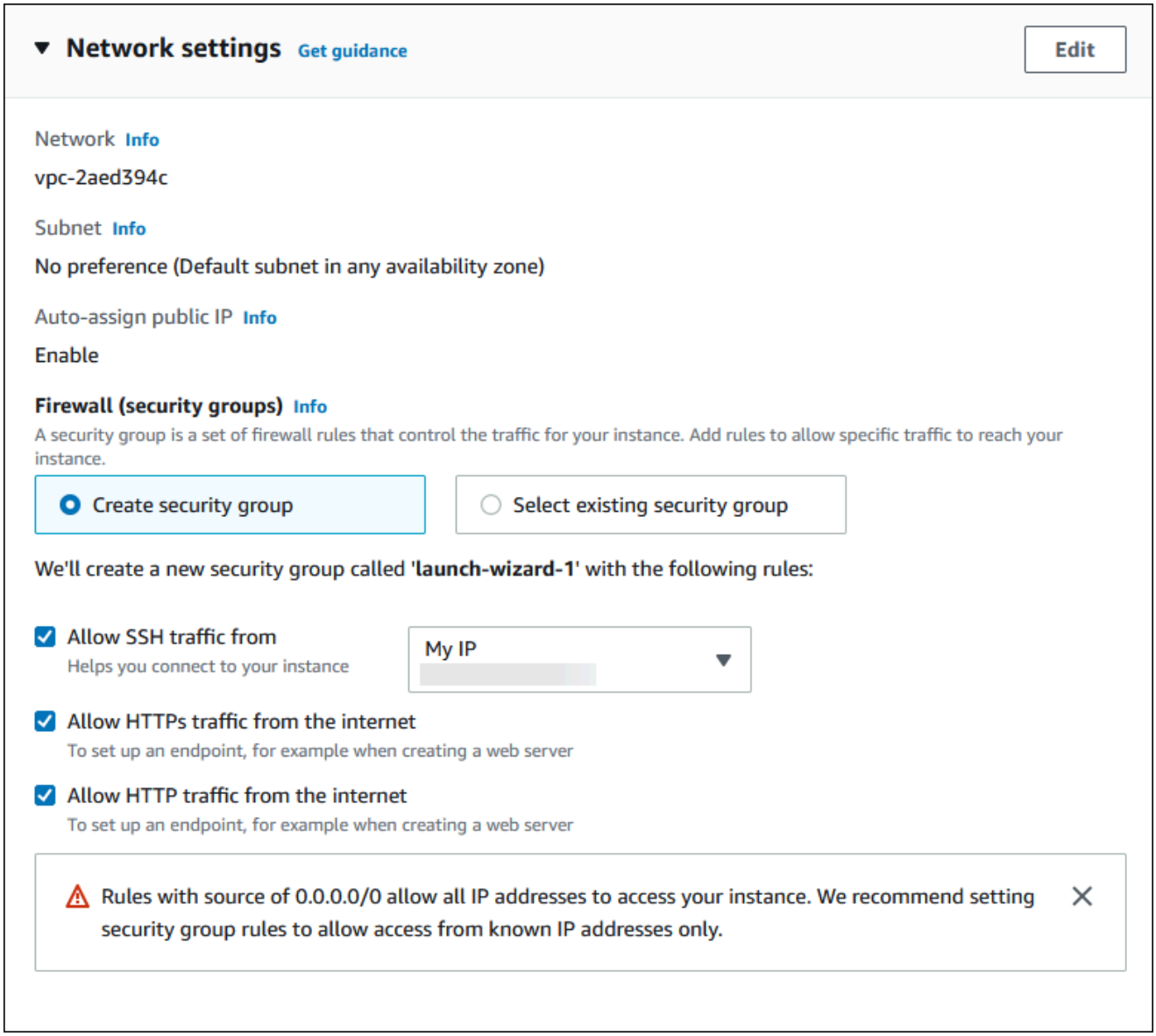
  如果显示的 IP 地址对于 SSH 连接是正确的，您可以选择 My IP（我的 IP）。

  否则，您可以确定要用来通过 Secure Shell（SSH）连接到 VPC 中的 EC2 实例的 IP 地址。要确定您的公有 IP 地址，请在新的浏览器窗口或标签页中，使用 https://checkip.amazonaws.com 上的服务。IP 地址的一个示例为 `203.0.113.25/32`。

  在许多情况下，您可能通过互联网服务提供商（ISP）进行连接，或者在不使用静态 IP 地址的情况下从防火墙之后进行连接。如果是这样，请确保确定客户端计算机使用的 IP 地址范围。

  > **Warning**
  >
  > 如果您使用 `0.0.0.0/0` 进行 SSH 访问，则所有 IP 地址可能能够使用 SSH 访问您的公有实例。在测试环境下短时间内，此方法尚可接受，但它对于生产环境并不安全。在生产环境中，将仅向特定 IP 地址或地址范围授权使用 SSH 访问您的实例。

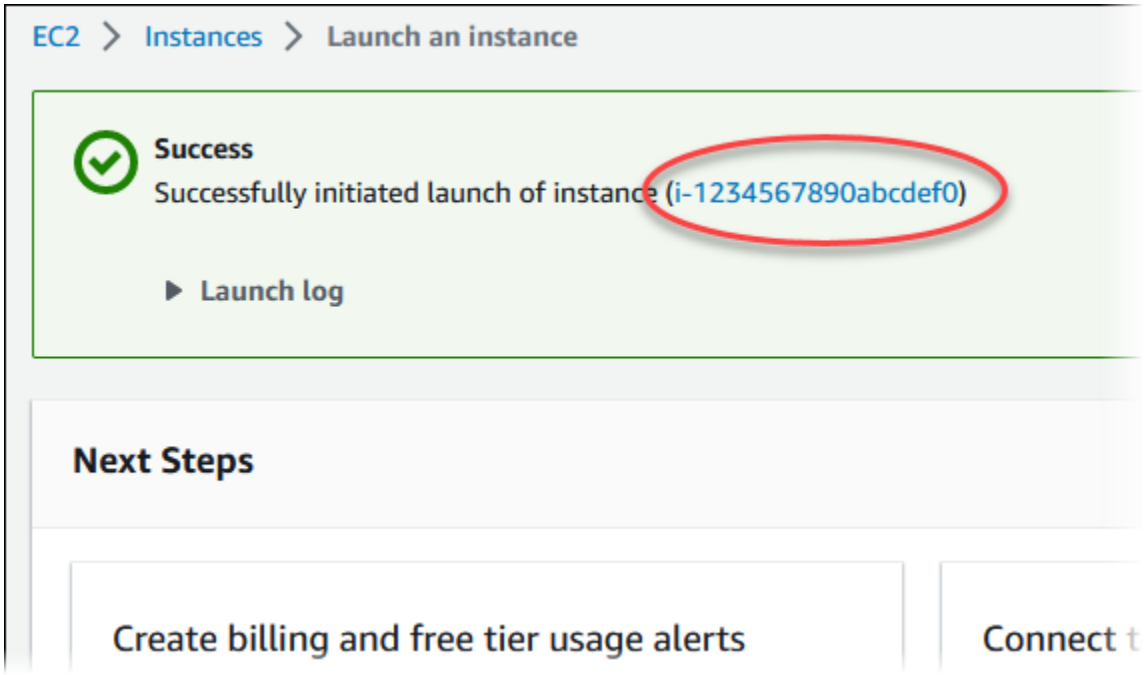
- 开启 Allow HTTPs traffic from the internet（允许来自互联网的 HTTPs 流量）。
- 开启 Allow HTTP traffic from the internet（允许来自互联网的 HTTP 流量）。



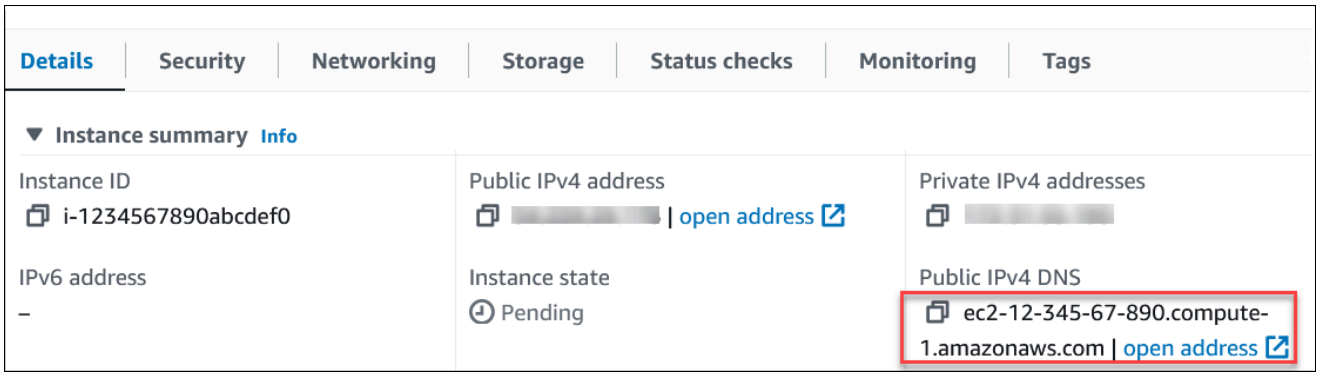
f. 对于其余部分保留原定设置值。

g. 查看 Summary（摘要）面板中您的实例配置的摘要，当您准备好后，选择 Launch instance（启动实例）。

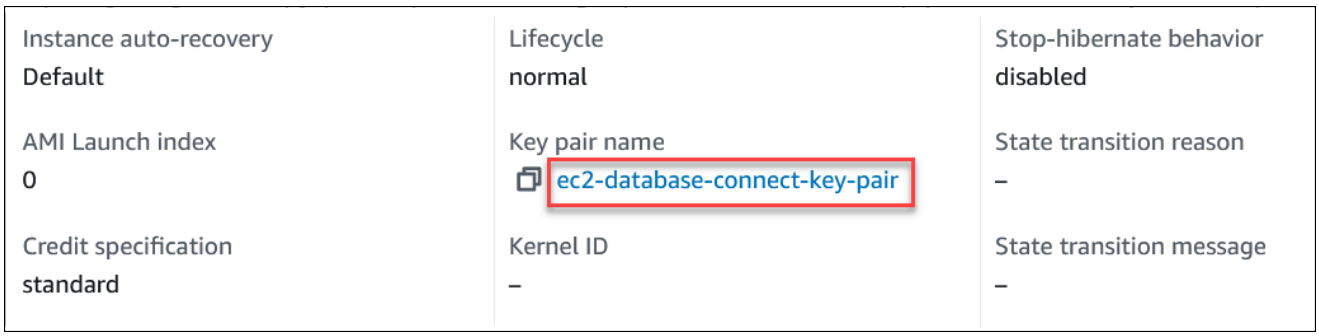
5. 在启动状态页面上，记下新 EC2 实例的标识符，例如：`i-1234567890abcdef0`。



6. 选择 EC2 实例标识符以打开 EC2 实例列表，然后选择您的 EC2 实例。
7. 在详细信息选项卡中，记下使用 SSH 进行连接时所需的以下值：
   a. 在实例摘要中，记下公有 IPv4 DNS 的值。



   b. 在实例详细信息中，记下密钥对名称的值。



8. 一直等到实例的 Instance state（实例状态）变为 Running（正在运行），才能继续。

9. 完成 创建 Amazon Aurora 数据库集群。

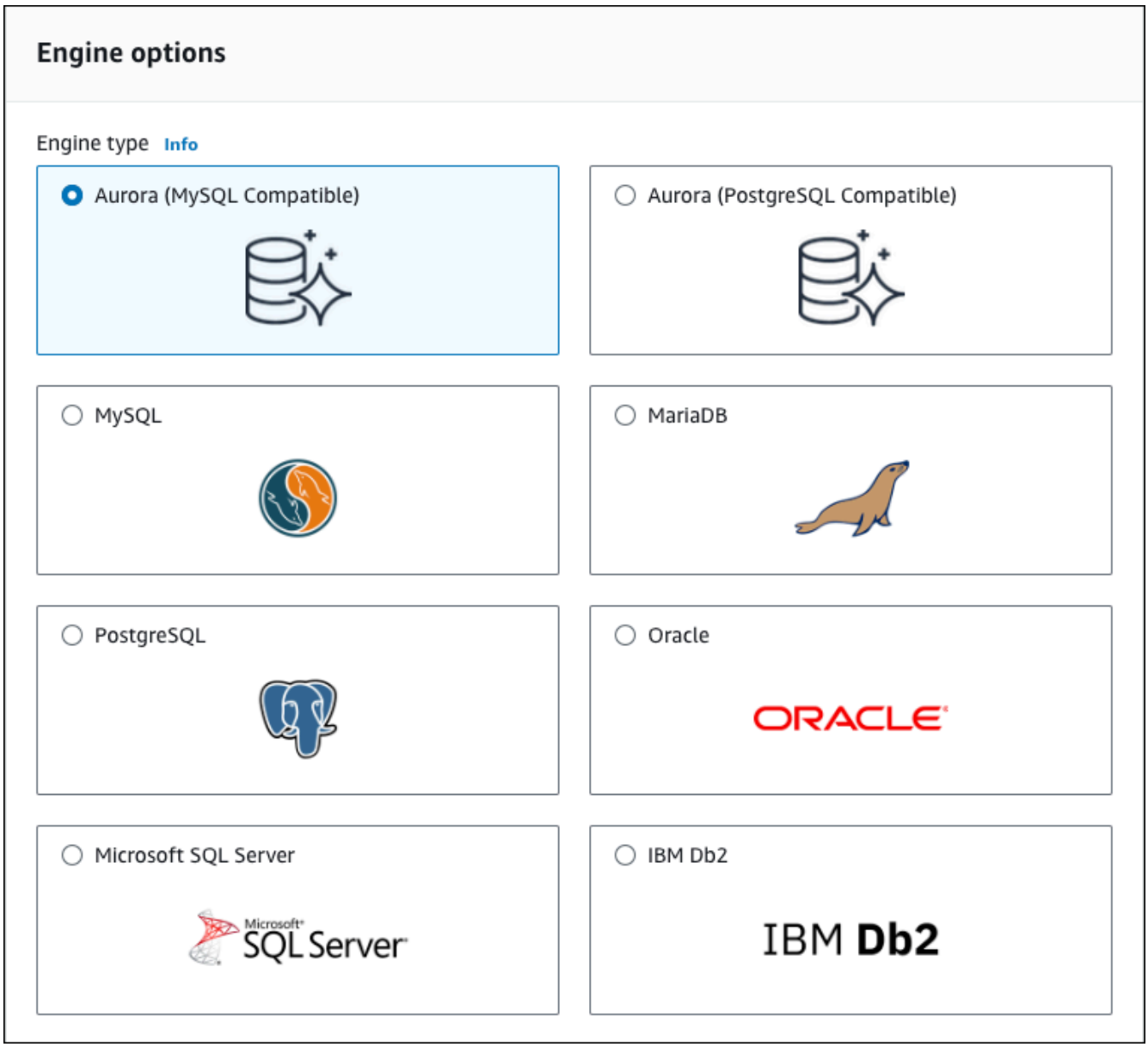
创建 Amazon Aurora 数据库集群

创建维护 Web 应用程序所用数据的 Amazon Aurora MySQL 或 Aurora PostgreSQL 数据库集群。

Aurora MySQL

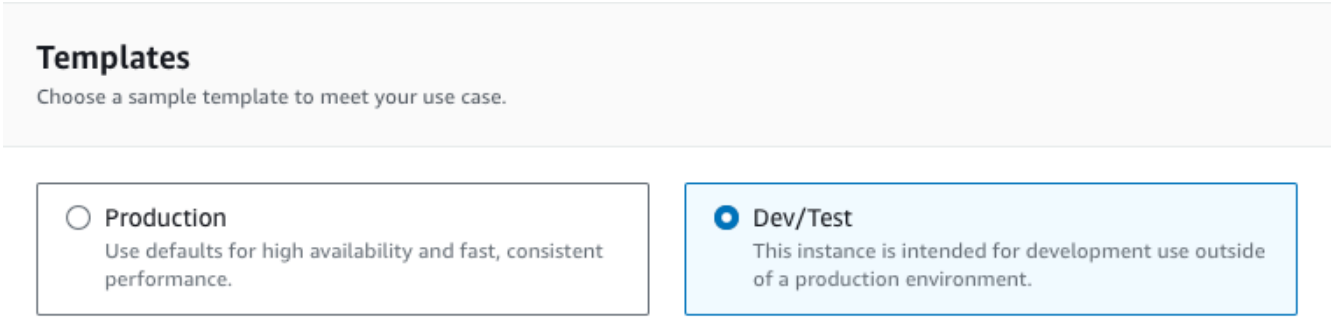
创建 Aurora MySQL 数据库集群的步骤

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在 AWS Management Console的右上角，请确保 AWS 区域与您在其中创建了 EC2 实例的区域相同。
3. 在导航窗格中，选择 Databases（数据库）。
4. 选择创建数据库。
5. 在创建数据库页面上，选择标准创建。
6. 对于引擎选项，选择 Aurora（MySQL 兼容）。



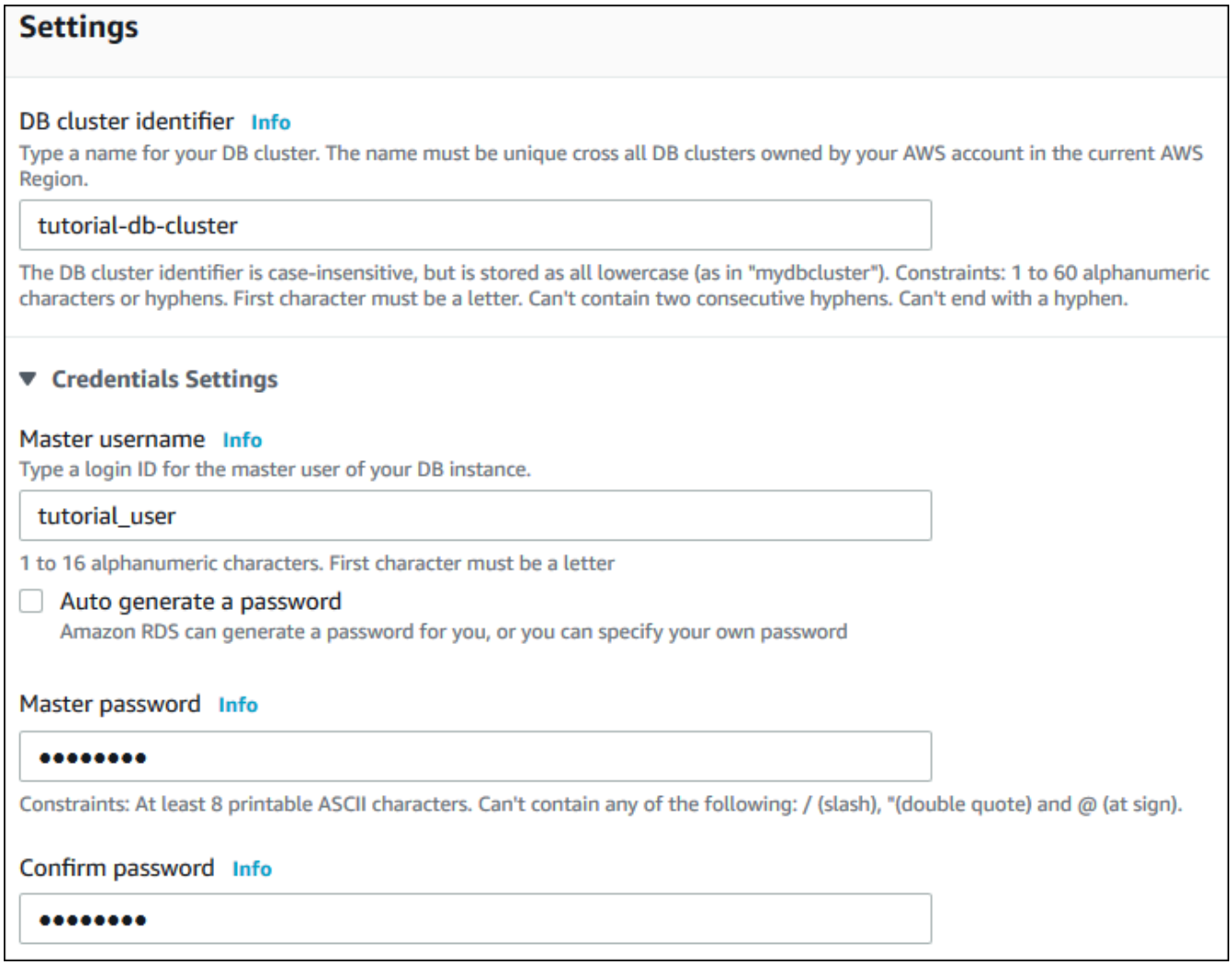
保留 Version（版本）和其他引擎选项的默认值。

7. 在模板部分中，选择开发/测试。



8. 在设置部分中，选择这些值：

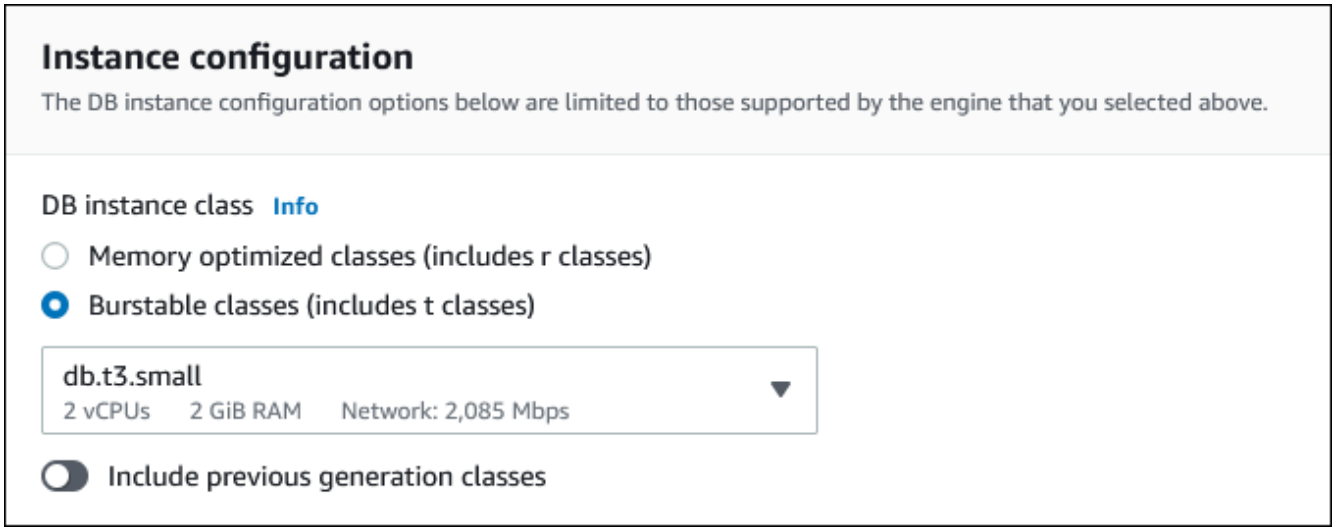
   - DB cluster identifier（数据库集群标识符）– 键入 **`tutorial-db-cluster`**。
   - Master username（主用户名）– 键入 **`tutorial_user`**。
   - Auto generate a password（自动生成密码）– 将该选项保留为关闭状态。
   - Master password（主密码）– 键入密码。
   - 确认密码 – 重新键入密码。



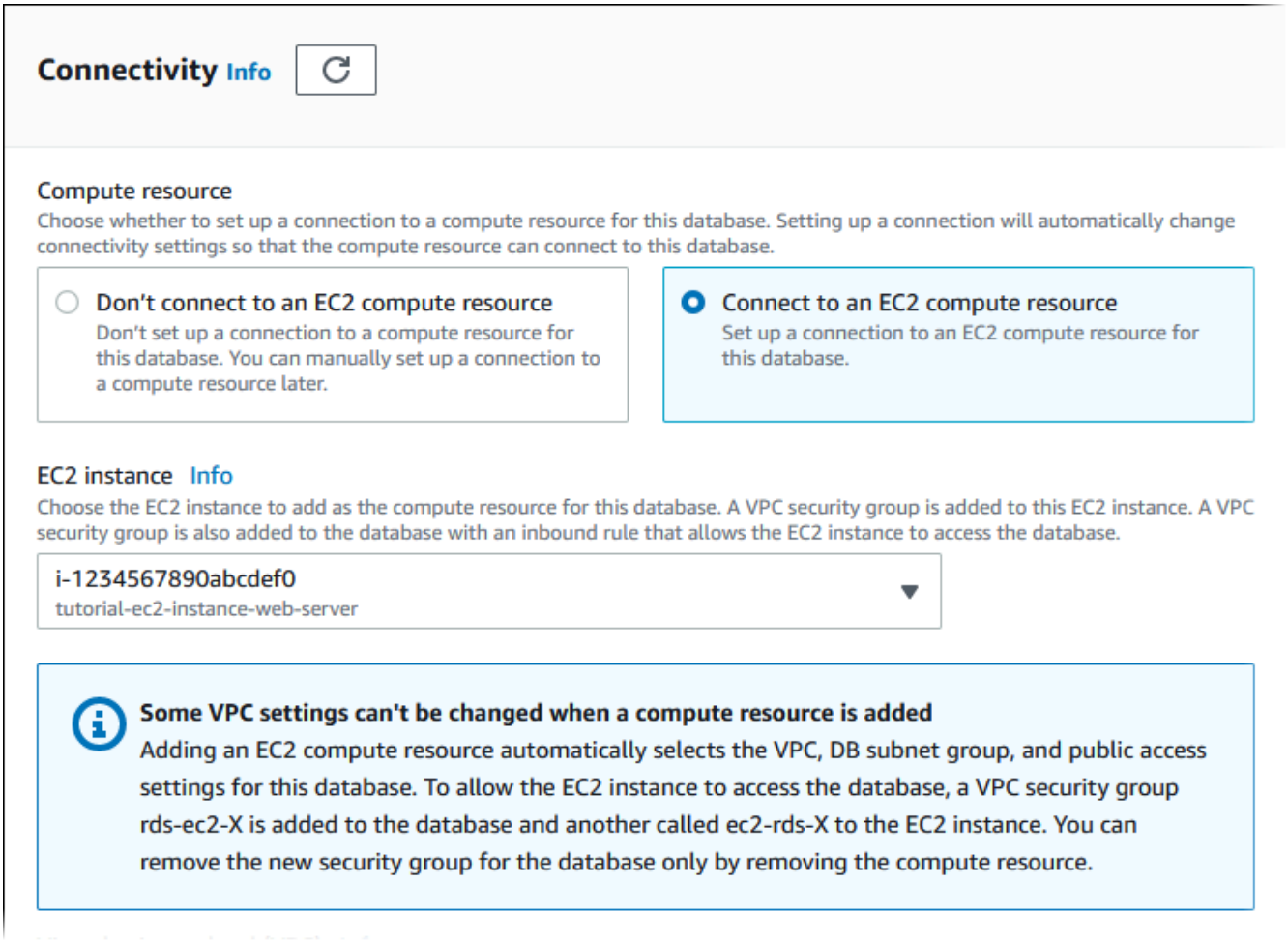
9. 在 Instance configuration（实例配置）部分中，设置以下值：

   - 可突增类（包括 t 类）
   - db.t3.small 或 db.t3.medium

> **Note**
>
> 建议仅将 T 数据库实例类用于开发和测试服务器，或其他非生产服务器。有关 T 实例类的更多详细信息，请参阅数据库实例类类型。

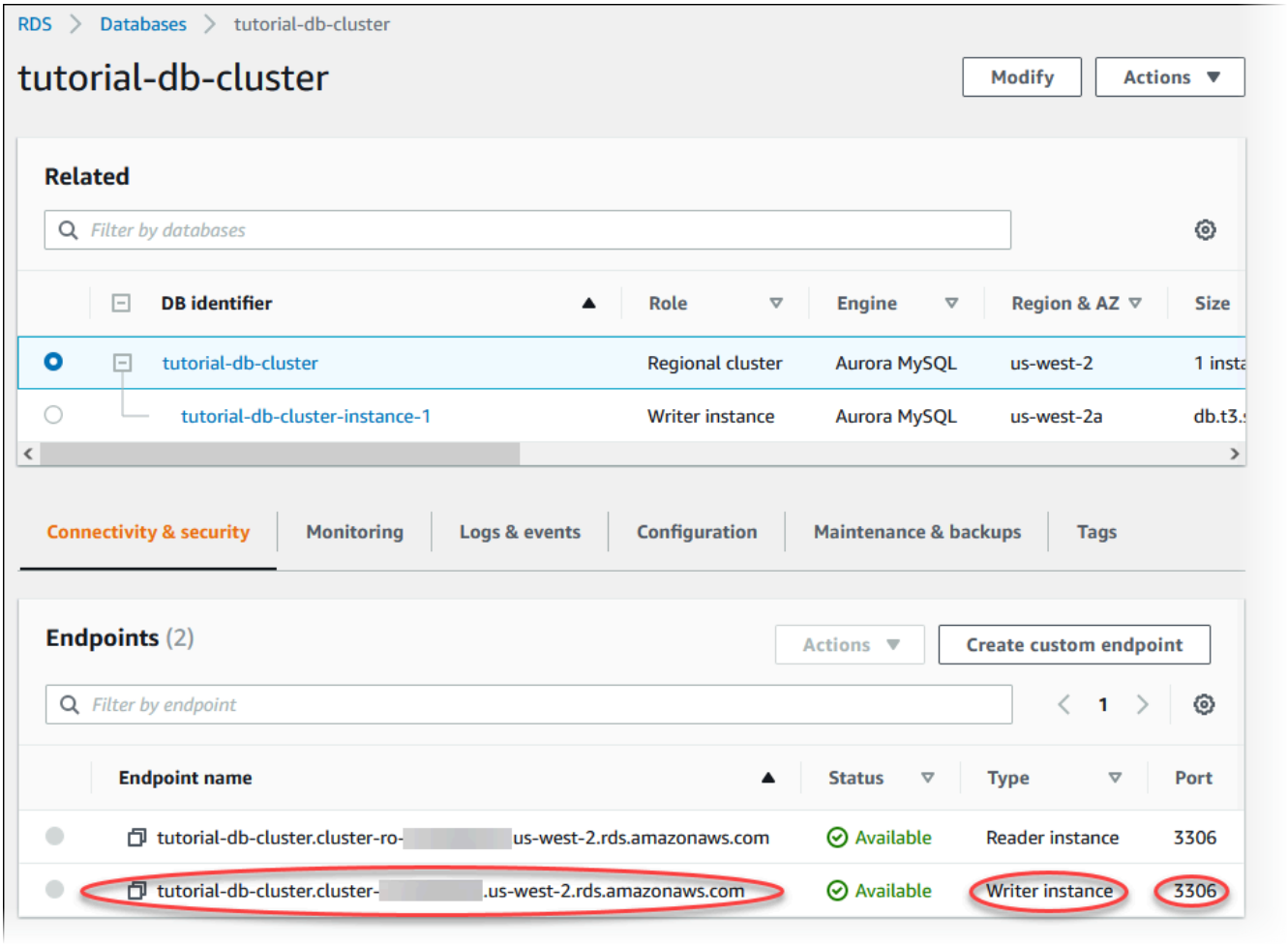


10. 在 Availability and durability（可用性和持久性）部分中，使用默认值。
11. 在 Connectivity（连接）部分中，设置以下值并将其他值保留为其默认值：
    - 对于 Compute resource（计算资源），选择 Connect to an EC2 compute resource（连接到 EC2 计算资源）。
    - 对于 EC2 instance（EC2 实例），选择您之前创建的 EC2 实例，例如 tutorial-ec2-instance-web-server。



12. 打开附加配置部分，然后为初始数据库名称输入 **`sample`**。保留其他选项的默认设置。
13. 要创建 Aurora MySQL 数据库集群，请选择 Create database (创建数据库)。

    您的新数据库集群显示在 Databases (数据库) 列表中，状态为 Creating (正在创建)。
14. 等待新数据库集群的 Status (状态) 显示为 Available (可用)。然后选择数据库集群名称以显示其详细信息。
15. 在 Connectivity & security (连接性和安全性) 部分中，查看写入器数据库实例的 Endpoint (终端节点) 和 Port (端口)。



记下写入器数据库实例的终端节点和端口。您使用这些信息将 Web 服务器连接到数据库集群。

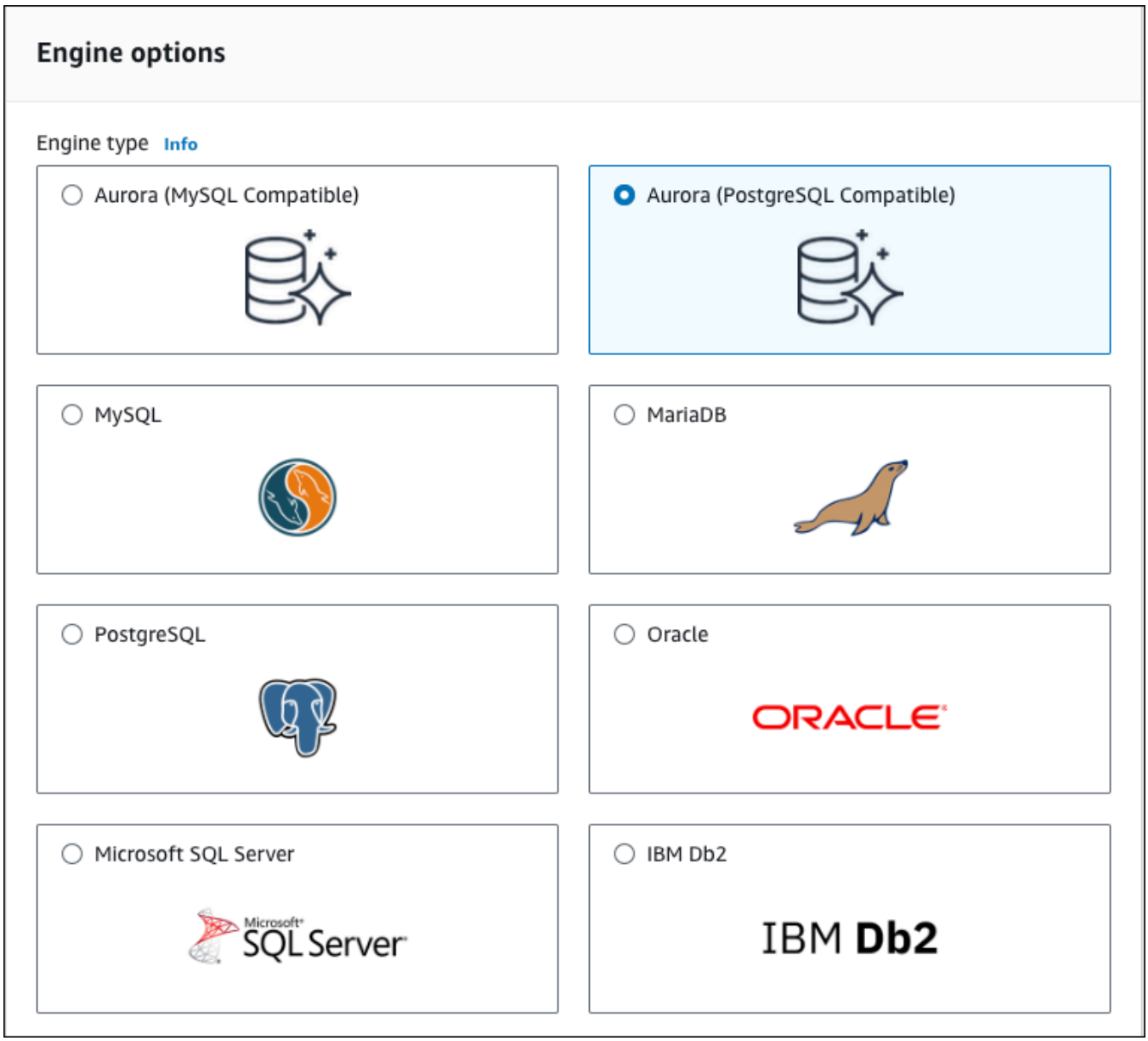
16. 完成在 EC2 实例上安装 Web 服务器。

Aurora PostgreSQL

创建 Aurora PostgreSQL 数据库集群

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在 AWS Management Console的右上角，请确保 AWS 区域与您在其中创建了 EC2 实例的区域相同。
3. 在导航窗格中，选择 Databases（数据库）。
4. 选择创建数据库。

5. 在创建数据库页面上，选择标准创建。
6. 对于引擎选项，选择 Aurora（PostgreSQL 兼容）。



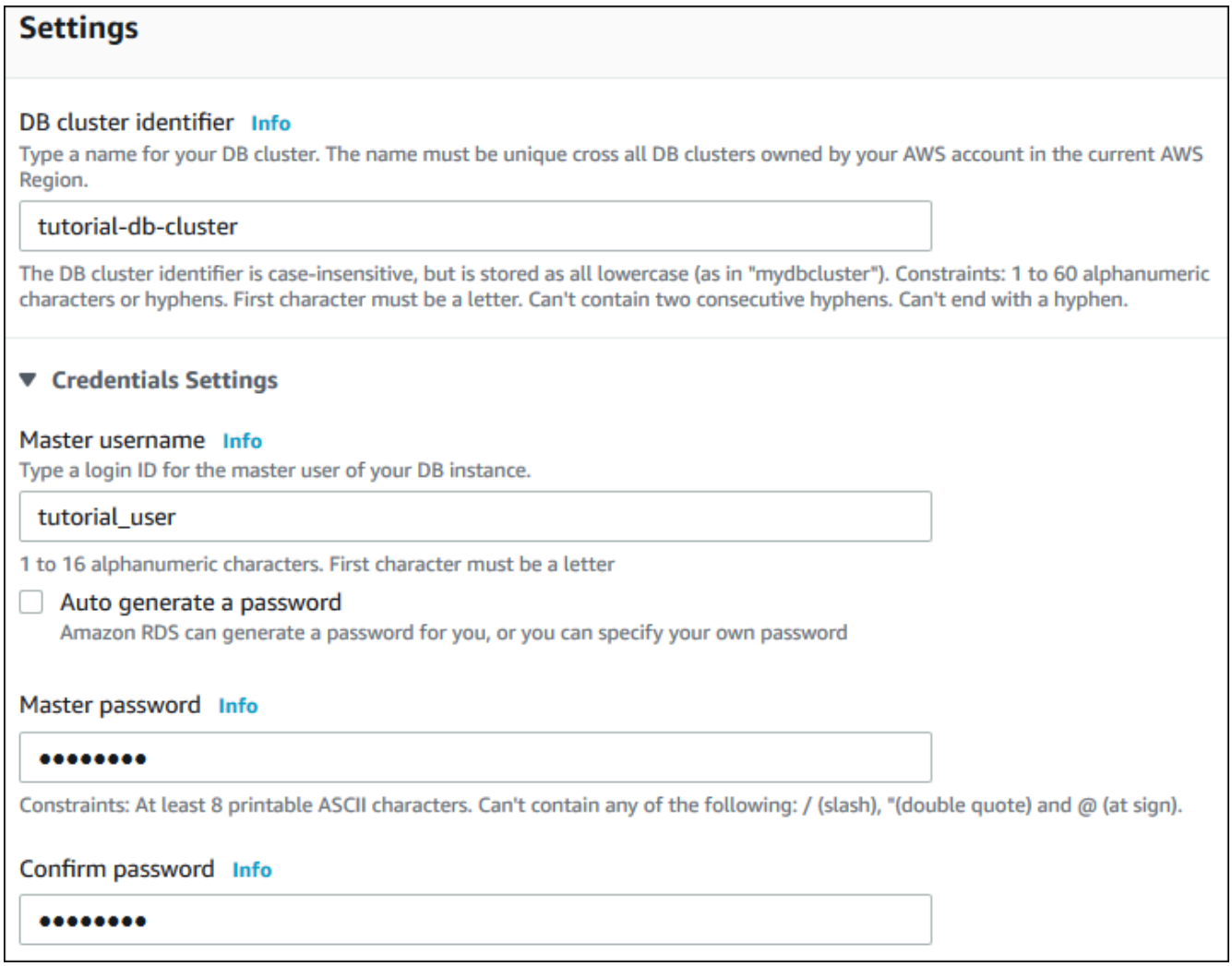
保留 Version（版本）和其他引擎选项的默认值。

7. 在模板部分中，选择开发/测试。



8. 在设置部分中，选择这些值：

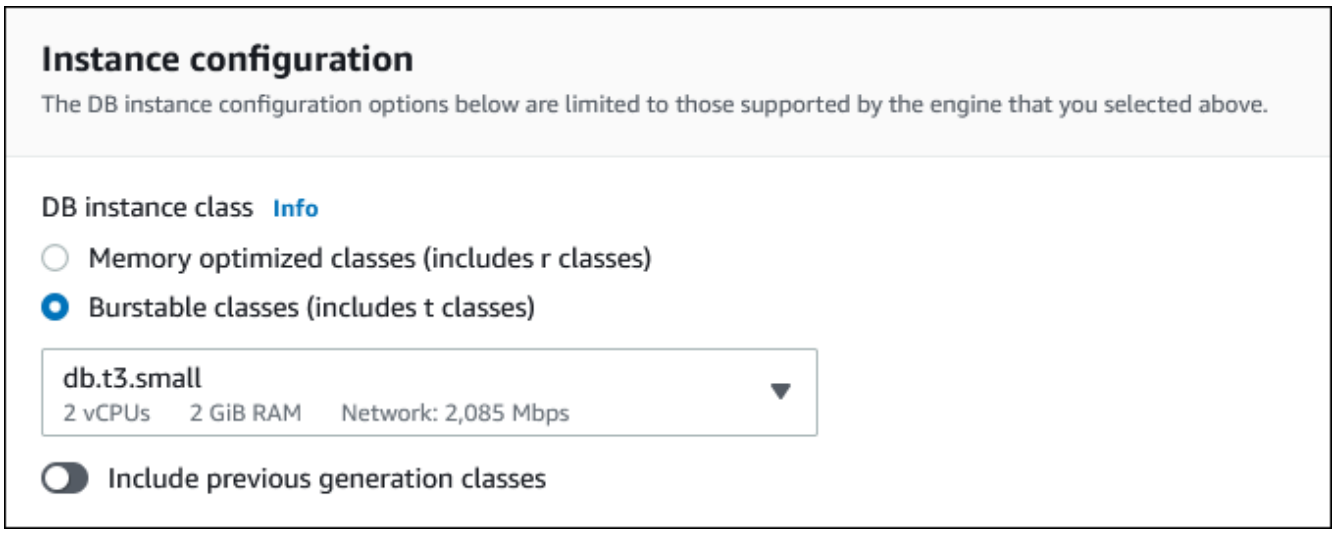
   - DB cluster identifier（数据库集群标识符）– 键入 **`tutorial-db-cluster`**。
   - Master username（主用户名）– 键入 **`tutorial_user`**。
   - Auto generate a password（自动生成密码）– 将该选项保留为关闭状态。
   - Master password（主密码）– 键入密码。
   - 确认密码 – 重新键入密码。



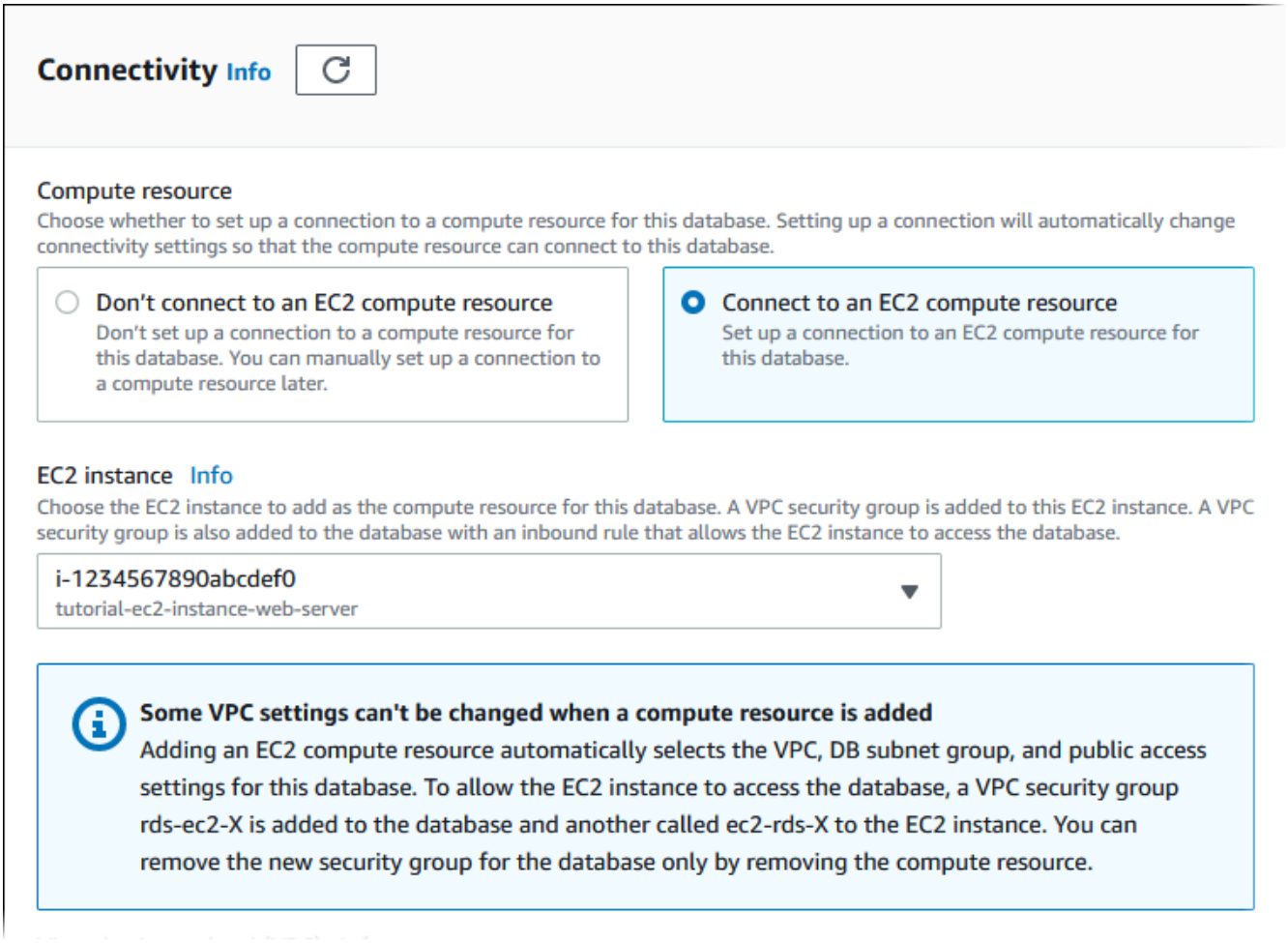
9. 在 Instance configuration（实例配置）部分中，设置以下值：

   - 可突增类（包括 t 类）
   - db.t3.small 或 db.t3.medium

   > **Note**
   >
   > 建议仅将 T 数据库实例类用于开发和测试服务器，或其他非生产服务器。有关 T 实例类的更多详细信息，请参阅数据库实例类类型。

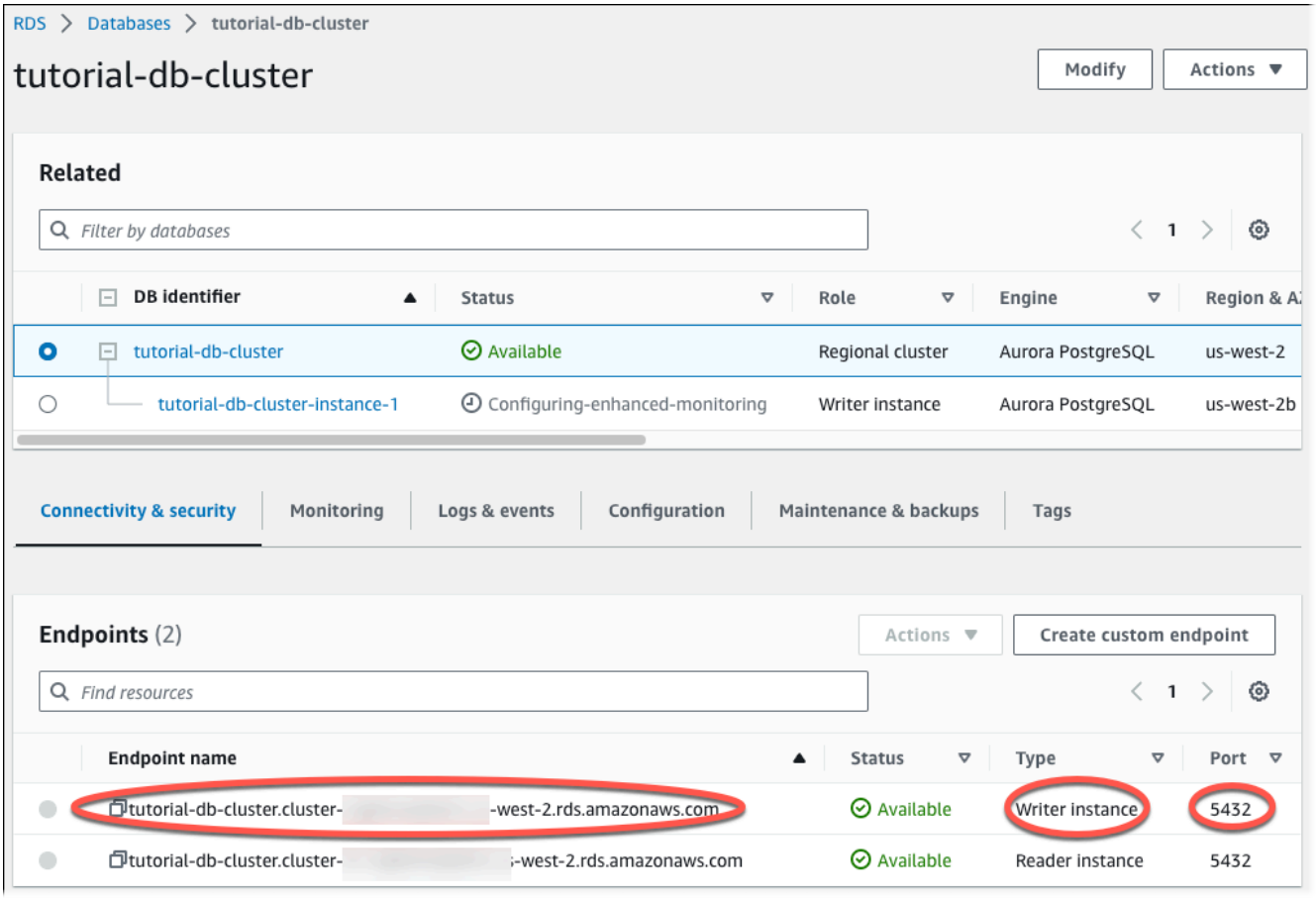


10. 在 Availability and durability（可用性和持久性）部分中，使用默认值。
11. 在 Connectivity（连接）部分中，设置以下值并将其他值保留为其默认值：
    - 对于 Compute resource（计算资源），选择 Connect to an EC2 compute resource（连接到 EC2 计算资源）。
    - 对于 EC2 instance（EC2 实例），选择您之前创建的 EC2 实例，例如 tutorial-ec2-instance-web-server。



12. 打开附加配置部分，然后为初始数据库名称输入 **`sample`**。保留其他选项的默认设置。
13. 要创建 Aurora PostgreSQL 数据库集群，请选择创建数据库。

    您的新数据库集群显示在 Databases (数据库) 列表中，状态为 Creating (正在创建)。
14. 等待新数据库集群的 Status (状态) 显示为 Available (可用)。然后选择数据库集群名称以显示其详细信息。
15. 在 Connectivity & security (连接性和安全性) 部分中，查看写入器数据库实例的 Endpoint (终端节点) 和 Port (端口)。



记下写入器数据库实例的终端节点和端口。您使用这些信息将 Web 服务器连接到数据库集群。

16. 完成在 EC2 实例上安装 Web 服务器。

在 EC2 实例上安装 Web 服务器

在您在启动 EC2 实例以连接数据库集群中创建的 EC2 实例上安装 Web 服务器。Web 服务器连接到您在创建 Amazon Aurora 数据库集群中创建的 Amazon Aurora 数据库集群。

使用 PHP 和 MariaDB 安装 Apache Web 服务器

连接到 EC2 实例并安装 Web 服务器。

连接到 EC2 实例并安装带有 PHP 的 Apache Web 服务器

1. 按照《Amazon EC2 用户指南》中的连接到 Linux 实例中的步骤操作，连接到您之前创建的 EC2 实例。

我们建议您使用 SSH 连接到 EC2 实例。如果 SSH 客户端实用程序安装在 Windows、Linux 或 Mac 上，则可以使用以下命令格式连接到该实例：

```
ssh -i location_of_pem_file ec2-user@ec2-instance-public-dns-name
```

例如，假设在 Linux 上 `ec2-database-connect-key-pair.pem` 存储在 `/dir1` 中，而 EC2 实例的公有 IPv4 DNS 为 `ec2-12-345-678-90.compute-1.amazonaws.com`。SSH 命令将如下所示：

```
ssh -i /dir1/ec2-database-connect-key-pair.pem ec2-
user@ec2-12-345-678-90.compute-1.amazonaws.com
```

2. 通过更新 EC2 实例上的软件，获取最新的错误修复和安全更新。要执行此操作，请使用以下命令。

> **Note**
>
> -y 选项安装更新时不提示确认。要在安装前检查更新，请忽略该选项。

```
sudo dnf update -y
```

3. 更新完成后，使用以下命令安装 Apache Web 服务器、PHP 和 MariaDB 或 PostgreSQL 软件。此命令同时安装多个软件包和相关依赖项。

   MariaDB & MySQL

   ```
   sudo dnf install -y httpd php php-mysqli mariadb105
   ```

   PostgreSQL

   ```
   sudo dnf install -y httpd php php-pgsql postgresql15
   ```

   如果您收到错误，则您的实例可能不是使用 Amazon Linux 2023 AMI 启动的。您可能使用的是 Amazon Linux 2 AMI。您可以使用以下命令查看 Amazon Linux 的版本。

```
cat /etc/system-release
```

有关更多信息，请参阅更新实例软件。

4. 使用下面所示的命令启动 Web 服务器。

```
sudo systemctl start httpd
```

您可以测试 Web 服务器是否已正确安装和启动。为此，请在 Web 浏览器的地址栏中输入 EC2 实例的公有域名系统 (DNS) 名称，例如：`http://ec2-42-8-168-21.us-west-1.compute.amazonaws.com`。如果 Web 服务器正在运行，您将看到 Apache 测试页面。

如果您没有看到 Apache 测试页面，请检查您在教程：创建 VPC 以用于数据库集群（仅限 IPv4）中创建的 VPC 安全组的入站规则。确保入站规则包括一条允许 HTTP（端口 80）访问 IP 地址以连接到 Web 服务器的规则。

> **Note**
>
> Apache 测试页面仅在文档根目录 `/var/www/html` 中无内容时才显示。将内容添加到文档根目录后，您的内容将显示在 EC2 实例的公有 DNS 地址处。在此之前，它出现在 Apache 测试页面上。

5. 使用 `systemctl` 命令配置 Web 服务器以使其在每次系统启动时启动。

```
sudo systemctl enable httpd
```

要允许 `ec2-user` 在 Apache Web 服务器的默认根目录中管理文件，请修改 `/var/www` 目录的所有权和权限。有多种方式可以完成此任务。在本教程中，可将 `ec2-user` 添加到 `apache` 组，将 `apache` 目录的所有权授予 `/var/www` 组，并为该组指定写入权限。

设置 Apache Web 服务器的文件权限

1. 将 `ec2-user` 用户添加到 `apache` 组。

```
sudo usermod -a -G apache ec2-user
```

2. 注销以刷新您的权限并包含新的 `apache` 组。

```
exit
```

3. 再重新登录并使用 apache 命令验证 groups 组是否存在。

```
groups
```

输出看上去类似于以下内容：

```
ec2-user adm wheel apache systemd-journal
```

4. 将 /var/www 目录的组所有权及其内容更改到 apache 组。

```
sudo chown -R ec2-user:apache /var/www
```

5. 更改 /var/www 及其子目录的目录权限，以添加组写入权限并设置未来创建的子目录上的组 ID。

```
sudo chmod 2775 /var/www
find /var/www -type d -exec sudo chmod 2775 {} \;
```

6. 递归地更改 /var/www 目录及其子目录中的文件的权限，以添加组写入权限。

```
find /var/www -type f -exec sudo chmod 0664 {} \;
```

现在，ec2-user（和 apache 组的任何将来成员）可以添加、删除和编辑 Apache 文档根目录中的文件。这使您可以添加内容，例如静态网站或 PHP 应用程序。

> **Note**
>
> 运行 HTTP 协议的 Web 服务器不为其发送或接收的数据提供传输安全。当您使用 Web 浏览器连接 HTTP 服务器时，窃取者可在沿网络路径的任何位置看到许多信息。这些信息包括您访问的 URL、您接收的网页内容以及任何 HTML 表单的内容（包括密码）。
> 保护您的 Web 服务器的最佳实践是安装对于 HTTPS（HTTP Secure）的支持。此协议利用 SSL/TLS 加密保护您的数据。有关更多信息，请参阅在Amazon EC2用户指南中的教程：使用 Amazon Linux AMI 配置 SSL/TLS。

将您的 Apache Web 服务器连接到数据库集群

接着，将内容添加到连接到 Amazon Aurora 数据库集群的 Apache Web 服务器。

将内容添加到连接到数据库集群的 Apache Web 服务器

1. 在仍连接到 EC2 实例时，将目录更改到 `/var/www` 并创建名为 `inc` 的新子目录。

```
cd /var/www
mkdir inc
cd inc
```

2. 在名为 `inc` 的 `dbinfo.inc` 目录中新建文件，然后通过调用 nano（或您选择的编辑器）编辑文件。

```
>dbinfo.inc
nano dbinfo.inc
```

3. 将以下内容添加到 `dbinfo.inc` 文件。在这里，*db_instance_endpoint* 是数据库实例的不带端口的数据库实例端点。

> **Note**
>
> 我们建议将用户名和密码信息放在不属于 Web 服务器的文档根目录的文件夹中。这样做会减少您的安全信息被泄露的可能性。
>
> 确保在应用程序中将 `master password` 更改为合适的密码。

```
<?php

define('DB_SERVER', 'db_cluster_writer_endpoint');
define('DB_USERNAME', 'tutorial_user');
define('DB_PASSWORD', 'master password');
define('DB_DATABASE', 'sample');
?>
```

4. 保存并关闭 `dbinfo.inc`文件。如果您使用的是 nano，请使用 Ctrl+S 和 Ctrl+X 保存并关闭文件。

5. 将目录更改为 `/var/www/html`。

```
cd /var/www/html
```

6. 在名为 html 的 SamplePage.php 目录中新建文件，然后通过调用 nano（或您选择的编辑器）编辑文件。

```
>SamplePage.php
nano SamplePage.php
```

7. 将以下内容添加到 SamplePage.php 文件：

   MariaDB & MySQL

```
<?php include "../inc/dbinfo.inc"; ?>
<html>
<body>
<h1>Sample page</h1>
<?php

  /* Connect to MySQL and select the database. */
  $connection = mysqli_connect(DB_SERVER, DB_USERNAME, DB_PASSWORD);

  if (mysqli_connect_errno()) echo "Failed to connect to MySQL: " .
 mysqli_connect_error();

  $database = mysqli_select_db($connection, DB_DATABASE);

  /* Ensure that the EMPLOYEES table exists. */
  VerifyEmployeesTable($connection, DB_DATABASE);

  /* If input fields are populated, add a row to the EMPLOYEES table. */
  $employee_name = htmlentities($_POST['NAME']);
  $employee_address = htmlentities($_POST['ADDRESS']);

  if (strlen($employee_name) || strlen($employee_address)) {
    AddEmployee($connection, $employee_name, $employee_address);
  }
?>

<!-- Input form -->
<form action="<?PHP echo $_SERVER['SCRIPT_NAME'] ?>" method="POST">
  <table border="0">
    <tr>
```

```
      <td>NAME</td>
      <td>ADDRESS</td>
    </tr>
    <tr>
      <td>
        <input type="text" name="NAME" maxlength="45" size="30" />
      </td>
      <td>
        <input type="text" name="ADDRESS" maxlength="90" size="60" />
      </td>
      <td>
        <input type="submit" value="Add Data" />
      </td>
    </tr>
  </table>
</form>

<!-- Display table data. -->
<table border="1" cellpadding="2" cellspacing="2">
  <tr>
    <td>ID</td>
    <td>NAME</td>
    <td>ADDRESS</td>
  </tr>

<?php

$result = mysqli_query($connection, "SELECT * FROM EMPLOYEES");

while($query_data = mysqli_fetch_row($result)) {
  echo "<tr>";
  echo "<td>",$query_data[0], "</td>",
       "<td>",$query_data[1], "</td>",
       "<td>",$query_data[2], "</td>";
  echo "</tr>";
}
?>

</table>

<!-- Clean up. -->
<?php

  mysqli_free_result($result);
```

```
  mysqli_close($connection);

?>

</body>
</html>


<?php

/* Add an employee to the table. */
function AddEmployee($connection, $name, $address) {
   $n = mysqli_real_escape_string($connection, $name);
   $a = mysqli_real_escape_string($connection, $address);

   $query = "INSERT INTO EMPLOYEES (NAME, ADDRESS) VALUES ('$n', '$a');";

   if(!mysqli_query($connection, $query)) echo("<p>Error adding employee data.</
p>");
}

/* Check whether the table exists and, if not, create it. */
function VerifyEmployeesTable($connection, $dbName) {
  if(!TableExists("EMPLOYEES", $connection, $dbName))
  {
     $query = "CREATE TABLE EMPLOYEES (
         ID int(11) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
         NAME VARCHAR(45),
         ADDRESS VARCHAR(90)
       )";

     if(!mysqli_query($connection, $query)) echo("<p>Error creating table.</
p>");
  }
}

/* Check for the existence of a table. */
function TableExists($tableName, $connection, $dbName) {
  $t = mysqli_real_escape_string($connection, $tableName);
  $d = mysqli_real_escape_string($connection, $dbName);

  $checktable = mysqli_query($connection,
      "SELECT TABLE_NAME FROM information_schema.TABLES WHERE TABLE_NAME = '$t'
 AND TABLE_SCHEMA = '$d'");
```

```
  if(mysqli_num_rows($checktable) > 0) return true;

  return false;
}
?>
```

PostgreSQL

```
<?php include "../inc/dbinfo.inc"; ?>

<html>
<body>
<h1>Sample page</h1>
<?php

/* Connect to PostgreSQL and select the database. */
$constring = "host=" . DB_SERVER . " dbname=" . DB_DATABASE . " user=" .
 DB_USERNAME . " password=" . DB_PASSWORD ;
$connection = pg_connect($constring);

if (!$connection){
 echo "Failed to connect to PostgreSQL";
 exit;
}

/* Ensure that the EMPLOYEES table exists. */
VerifyEmployeesTable($connection, DB_DATABASE);

/* If input fields are populated, add a row to the EMPLOYEES table. */
$employee_name = htmlentities($_POST['NAME']);
$employee_address = htmlentities($_POST['ADDRESS']);

if (strlen($employee_name) || strlen($employee_address)) {
  AddEmployee($connection, $employee_name, $employee_address);
}

?>

<!-- Input form -->
<form action="<?PHP echo $_SERVER['SCRIPT_NAME'] ?>" method="POST">
  <table border="0">
```

```
    <tr>
      <td>NAME</td>
      <td>ADDRESS</td>
    </tr>
    <tr>
      <td>
    <input type="text" name="NAME" maxlength="45" size="30" />
      </td>
      <td>
    <input type="text" name="ADDRESS" maxlength="90" size="60" />
      </td>
      <td>
    <input type="submit" value="Add Data" />
      </td>
    </tr>
  </table>
</form>
<!-- Display table data. -->
<table border="1" cellpadding="2" cellspacing="2">
  <tr>
    <td>ID</td>
    <td>NAME</td>
    <td>ADDRESS</td>
  </tr>

<?php

$result = pg_query($connection, "SELECT * FROM EMPLOYEES");

while($query_data = pg_fetch_row($result)) {
  echo "<tr>";
  echo "<td>",$query_data[0], "</td>",
       "<td>",$query_data[1], "</td>",
       "<td>",$query_data[2], "</td>";
  echo "</tr>";
}
?>
</table>

<!-- Clean up. -->
<?php

  pg_free_result($result);
  pg_close($connection);
```

```
?>
</body>
</html>


<?php

/* Add an employee to the table. */
function AddEmployee($connection, $name, $address) {
   $n = pg_escape_string($name);
   $a = pg_escape_string($address);
   echo "Forming Query";
   $query = "INSERT INTO EMPLOYEES (NAME, ADDRESS) VALUES ('$n', '$a');";

   if(!pg_query($connection, $query)) echo("<p>Error adding employee data.</
p>");
}

/* Check whether the table exists and, if not, create it. */
function VerifyEmployeesTable($connection, $dbName) {
  if(!TableExists("EMPLOYEES", $connection, $dbName))
  {
     $query = "CREATE TABLE EMPLOYEES (
         ID serial PRIMARY KEY,
         NAME VARCHAR(45),
         ADDRESS VARCHAR(90)
       )";

     if(!pg_query($connection, $query)) echo("<p>Error creating table.</p>");
  }
}
/* Check for the existence of a table. */
function TableExists($tableName, $connection, $dbName) {
  $t = strtolower(pg_escape_string($tableName)); //table name is case sensitive
  $d = pg_escape_string($dbName); //schema is 'public' instead of 'sample' db
 name so not using that

  $query = "SELECT TABLE_NAME FROM information_schema.TABLES WHERE TABLE_NAME =
 '$t';";
  $checktable = pg_query($connection, $query);

  if (pg_num_rows($checktable) >0) return true;
  return false;
```

```
}
?>
```

8. 保存并关闭 SamplePage.php 文件。
9. 打开 Web 浏览器并浏览到 http://*EC2 instance endpoint*/SamplePage.php（例如：http://ec2-12-345-67-890.us-west-2.compute.amazonaws.com/SamplePage.php）来验证 Web 服务器是否已成功连接到数据库集群。

您可以使用 SamplePage.php 将数据添加到数据库集群。您添加的数据之后将显示在该页面上。要验证数据是否已插入到表中，请在 Amazon EC2 实例上安装 MySQL 客户端。然后，连接到数据库集群并查询表。

有关连接到数据库集群的更多信息，请参阅 连接到 Amazon Aurora 数据库集群。

要确保您的数据库集群尽可能安全，请验证 VPC 外部的源是否无法连接到您的数据库集群。

在您完成 Web 服务器和数据库测试后，应删除您的数据库集群 和 Amazon EC2 实例。

- 要删除数据库集群，请按照 删除 Aurora 数据库集群和数据库实例 中的说明操作。您无需创建最终快照。
- 要终止 Amazon EC2 实例，请按照 Amazon EC2 用户指南中的终止实例中的说明操作。

Amazon Aurora 教程和示例代码

AWS 文档包括多个教程，可指导您完成常见的 Amazon Aurora 使用案例。这些教程中的许多教程都向您展示了如何将 Amazon Aurora 与其他AWS服务一起使用。此外，您可以在 GitHub 中访问示例代码。

> **Note**
>
> 您可以在AWS数据库博客中找到更多教程。有关培训的信息，请参阅 AWS Training and Certification。

主题

- 本指南中的教程
- 其他 AWS 指南中的教程
- Amazon Aurora PostgreSQL 的 AWS 研讨会和实验室内容门户
- Amazon Aurora MySQL 的 AWS 研讨会和实验室内容门户
- GitHub 中的教程和示例代码
- 将此服务与 AWS SDK 结合使用

本指南中的教程

本指南中的以下教程为您演示如何使用 Amazon Aurora 执行常见任务。

- 教程：创建 VPC 以用于数据库集群（仅限 IPv4）

  了解如何在虚拟私有云（VPC）中基于 Amazon VPC 服务包括数据库集群。在这种情况下，VPC 与在同一 VPC 中的 Amazon EC2 实例上运行的 Web 服务器共享数据。

- 教程：创建 VPC 以用于数据库集群（双堆栈模式）

  了解如何在虚拟私有云（VPC）中基于 Amazon VPC 服务包括数据库集群。在这种情况下，VPC 与同一 VPC 中的 Amazon EC2 实例共享数据。在本教程中，您将为此场景创建 VPC，该 VPC 与在双堆栈模式下运行的数据库一起使用。

- 教程：创建 Web 服务器和 Amazon Aurora 数据库集群

了解如何使用 PHP 安装 Apache Web 服务器并创建 MySQL 数据库。Web 服务器在使用 Amazon Linux 的 Amazon EC2 实例上运行，而 MySQL 数据库是 一个 Aurora MySQL 数据库集群。Amazon EC2 实例和数据库集群都在 Amazon VPC 中运行。

- 教程：从数据库集群快照中还原 Amazon Aurora 数据库集群

  了解如何从数据库集群快照中还原数据库集群。

- 教程：使用标签指定要停止哪些 Aurora 数据库集群

  了解如何使用标签指定要停止哪些 Aurora 数据库集群。

- 教程：使用 Amazon EventBridge 记录数据库实例的状态更改

  了解如何使用 Amazon EventBridge 和 AWS Lambda 记录数据库实例阶段更改。

其他 AWS 指南中的教程

其他AWS指南中的以下教程为您演示了如何使用 Amazon Aurora 执行常见任务：

> **Note**
>
> 一些教程使用 Amazon RDS 数据库实例，但它们可以进行调整以使用 Aurora 数据库集群。

- AWS AppSync 开发人员指南中的教程：Aurora Serverless

  了解如何使用 AWS AppSync 提供数据来源，以便在启用数据 API 的情况下对 Aurora Serverless 数据库集群运行 SQL 命令。您可以使用 AWS AppSync 解析程序通过 GraphQL 查询、更改和订阅对数据 API 运行 SQL 语句。

- AWS Secrets Manager 用户指南中的教程：为 AWS 数据库轮换密钥

  了解如何为 AWS 数据库创建密钥并将密钥配置为按计划轮换。您手动触发一次轮换，然后确认密钥的新版本能否继续提供访问权限。

- AWS Elastic Beanstalk 开发人员指南中的教程和示例

  了解如何部署将 Amazon RDS 数据库与AWS Elastic Beanstalk结合使用的应用程序。

- Amazon Machine Learning Developer Guide 中的使用来自 Amazon RDS 数据库的数据创 Amazon ML 数据源

了解如何从存储在 MySQL 数据库实例中的数据创建 Amazon Machine Learning (Amazon ML) 数据源对象。

- Amazon QuickSight 用户指南 中的在 VPC 中手动启用 Amazon RDS 实例访问权限

  了解如何启用 QuickSight 对 VPC 中的 Amazon RDS 数据库实例的访问权限。

Amazon Aurora PostgreSQL 的 AWS 研讨会和实验室内容门户

以下一组研讨会和其他实践内容集可帮助您了解 Amazon Aurora PostgreSQL 的特征和功能：

- Creating a New Aurora Cluster Manually

  了解如何手动创建 Amazon Aurora PostgreSQL 集群。

- Configure Cloud9 and Initialize Database

  了解如何配置 Cloud9 和初始化 PostgreSQL 数据库。

- 快速克隆

  了解如何创建 Aurora 快速克隆。

- 查询计划管理

  了解如何使用查询计划管理来控制一组语句的执行计划。

- 集群缓存管理

  了解 Aurora PostgreSQL 中的集群缓存管理特征。

- 数据库活动流

  了解如何通过此特征监控和审计您的数据库活动。

- 使用性能详情

  了解如何使用性能详情来监控和调整数据库实例。

- 使用 RDS 工具监控性能

  了解如何使用 AWS 和 Postgres 工具（Cloudwatch、增强监控、慢速查询日志、Performance Insights、PostgreSQL 目录视图）来了解性能问题，并确定提高数据库性能的方法。

- 自动扩缩只读副本

了解 Aurora 只读副本自动扩缩是如何使用负载生成器脚本在实践中发挥作用的。

- 测试容错能力

  了解数据库集群如何容错。

- Aurora 全局数据库

  了解 Aurora 全局数据库。

- 使用机器学习

  了解 Aurora 机器学习。

- Aurora Serverless v2

  了解 Aurora Serverless v2。

- 适用于 Aurora PostgreSQL 的可信语言扩展

  学习如何构建在 Aurora PostgreSQL 上安全运行的高性能扩展。

Amazon Aurora MySQL 的 AWS 研讨会和实验室内容门户

以下一组研讨会和其他实践内容集可帮助您了解 Amazon Aurora MySQL 的特征和功能：

- 创建 Aurora 集群

  了解如何手动创建 Amazon Aurora MySQL 集群。

- 创建 Cloud9 基于云的 IDE 环境以连接到您的数据库

  了解如何配置 Cloud9 和初始化 MySQL 数据库。

- 快速克隆

  了解如何创建 Aurora 快速克隆。

- 回溯集群

  了解如何回溯数据库集群。

- 使用 Performance Insights

  了解如何使用 Performance Insights 来监控和调整数据库实例。

- 使用 RDS 工具监控性能

了解如何使用 AWS 和 SQL 工具来了解性能问题并确定提高数据库性能的方法。

- 分析查询性能

  了解如何使用不同的工具排查与 SQL 性能相关的问题。

- 自动扩缩只读副本

  了解自动扩缩只读副本的工作原理。

- 测试容错能力

  了解 Aurora MySQL 中的高可用性和容错特征。

- Aurora 全局数据库

  了解 Aurora 全局数据库。

- Aurora Serverless v2

  了解 Aurora Serverless v2。

- 使用机器学习

  了解 Aurora 机器学习。

GitHub 中的教程和示例代码

GitHub 中的以下教程和示例代码为您演示如何使用 Amazon Aurora 执行常见任务：

- 创建 Aurora Serverless v2 借阅图书馆

  了解如何创建一个借阅图书馆应用程序，读者可以在其中借书和还书。该示例使用 Aurora Serverless v2 和 适用于 Python (Boto3) 的 AWS SDK。

- 使用 SDK for Java 2.x，通过用于查询 Aurora Serverless v2 数据的 Spring REST API 创建 Amazon Aurora 项目跟踪器应用程序

  了解如何创建用于查询 Aurora Serverless v2 数据的 Spring REST API。它可供使用 SDK for Java 2.x 的 React 应用程序使用。

- 创建一个使用 Aurora Serverless v2 查询 适用于 PHP 的 AWS SDK 数据的 Amazon Aurora 项目跟踪器应用程序

了解如何创建使用数据 API 的 `RdsDataClient` 以及 Aurora Serverless v2 来跟踪和报告工作项目的应用程序。本示例使用 适用于 PHP 的 AWS SDK。

- 创建一个使用 Aurora Serverless v2 查询 适用于 Python (Boto3) 的 AWS SDK 数据的 Amazon Aurora 项目跟踪器应用程序

  了解如何创建使用数据 API 的 `RdsDataClient` 以及 Aurora Serverless v2 来跟踪和报告工作项目的应用程序。本示例使用 适用于 Python (Boto3) 的 AWS SDK。

将此服务与 AWS SDK 结合使用

AWS 软件开发工具包（SDK）适用于许多常用编程语言。每个软件开发工具包都提供 API、代码示例和文档，使开发人员能够更轻松地以其首选语言构建应用程序。

SDK 文档	代码示例
适用于 C++ 的 AWS SDK	适用于 C++ 的 AWS SDK 代码示例
AWS CLI	AWS CLI 代码示例
适用于 Go 的 AWS SDK	适用于 Go 的 AWS SDK 代码示例
适用于 Java 的 AWS SDK	适用于 Java 的 AWS SDK 代码示例
适用于 JavaScript 的 AWS SDK	适用于 JavaScript 的 AWS SDK 代码示例
适用于 Kotlin 的 AWS SDK	适用于 Kotlin 的 AWS SDK 代码示例
适用于 .NET 的 AWS SDK	适用于 .NET 的 AWS SDK 代码示例
适用于 PHP 的 AWS SDK	适用于 PHP 的 AWS SDK 代码示例
AWS Tools for PowerShell	AWS Tools for PowerShell 代码示例
适用于 Python (Boto3) 的 AWS SDK	适用于 Python (Boto3) 的 AWS SDK 代码示例
适用于 Ruby 的 AWS SDK	适用于 Ruby 的 AWS SDK 代码示例
AWS SDK for Rust	AWS SDK for Rust 代码示例

SDK 文档	代码示例
适用于 SAP ABAP 的 AWS SDK	适用于 SAP ABAP 的 AWS SDK 代码示例
AWS SDK for Swift	AWS SDK for Swift 代码示例

有关特定于此服务的示例，请参阅使用 AWS SDK 的 Aurora 代码示例。

> **示例可用性**
>
> 找不到所需的内容？ 通过使用此页面底部的提供反馈链接请求代码示例。

以编程方式访问 Amazon Aurora

Amazon RDS 为您提供以下工具，以便以编程方式管理 Amazon Aurora 资源。

AWS Command Line Interface (AWS CLI)

可以在命令行 Shell 中使用 AWS CLI 来创建和管理 Amazon Aurora 资源。对于 Amazon RDS 等 AWS 服务，AWS CLI 提供对 API 的直接访问。有关 Amazon RDS 命令的语法和示例，请参阅《AWS CLI 命令参考》中的 rds。

AWS CloudFormation

使用此 AWS 基础设施即代码（IaC）工具，可以创建模板来描述您想要的所有 Amazon Aurora 资源，而 AWS CloudFormation 为您预调配和配置这些资源。有关更多信息，请参阅 the section called “使用 AWS CloudFormation 创建资源”。

AWS 软件开发工具包（SDK）

AWS 为许多流行的技术和编程语言提供 SDK。这些 SDK 使您能够更轻松地从应用程序中使用该语言或技术调用 AWS 服务。有关这些 SDK 的更多信息，请参阅用于在 AWS 上开发和管理应用程序的工具。

Amazon RDS API

此 API 是 Amazon RDS 的协议级接口。使用此 API 时，必须正确格式化每个 HTTPS 请求，并向每个请求添加有效的数字签名。有关更多信息，请参阅 Amazon RDS API 参考。

Console-to-Code

使用此工具，可以为在 Amazon RDS 控制台中执行的操作生成代码，并在其它工具（例如 AWS CloudFormation）中使用该代码。有关更多信息，请参阅the section called “Console-to-Code”。

RDS 数据 API

RDS 数据 API（数据 API）提供对 Amazon Aurora Serverless 数据库中数据的编程访问。借助数据 API，可以使用安全的 HTTP 端点来运行 SQL 语句，而无需管理连接。有关更多信息，请参阅使用 Amazon RDS 数据 API。

使用 Console-to-Code 为 Amazon Aurora 控制台操作生成代码

控制台提供了一条用来创建资源和测试原型的引导式路径。如果您希望大规模创建相同的资源，则需要使用自动化代码。Console-to-Code 是 Amazon Q 开发者版的一项功能，可以帮助您开始使用自动

化代码。Console-to-Code 记录控制台操作，包括默认值和您提供的参数。随后，它利用生成式人工智能，以您首选的语言和格式为您选择的操作提供建议的代码。由于控制台工作流程可确保您指定的参数值同时有效，因此，您使用 Console-to-Code 生成的代码具有兼容的参数值。您可以将此代码用作一个起点，然后对其进行自定义，以使它可用于您的特定使用案例的生产。

例如，使用 Console-to-Code，可以记录创建 Aurora 数据库集群的过程，并选择以 AWS CloudFormation JSON 格式生成代码。然后，您可以复制该代码并对其进行自定义，以便在 AWS CloudFormation 模板中使用。

Console-to-Code 目前能够以下列语言和格式生成基础设施即代码（IaC）：

- CDK Java
- CDK Python
- CDK TypeScript
- CloudFormation JSON
- CloudFormation YAML

有关如何使用 Console-to-Code 的更多信息和说明，请参阅《Amazon Q Developer User Guide》中的 Automating AWS services with Amazon Q Developer Console-to-Code。

设置 Amazon Aurora 数据库集群

本节介绍了如何设置 Aurora 数据库集群。在创建 Aurora 数据库集群之前，请确定运行该数据库集群的数据库实例类。此外，选择 AWS 区域以确定运行该数据库集群的位置。接下来，创建该数据库集群。如果在 Aurora 外部包含数据，您可以将数据迁移到 Aurora 数据库集群中。

主题

- 创建 Amazon Aurora 数据库集群
- 使用 AWS CloudFormation 创建 Amazon Aurora 资源
- 连接到 Amazon Aurora 数据库集群
- Amazon Aurora 的参数组
- 将数据迁移到 Amazon Aurora 数据库集群
- 使用 Aurora 数据库集群设置创建 Amazon ElastiCache 缓存
- 使用 AWS Database Migration Service 将 EC2 数据库自动迁移到 Amazon Aurora
- 教程：使用自定义参数组创建 MySQL 数据库集群

创建 Amazon Aurora 数据库集群

Amazon Aurora 数据库集群包括一个与 MySQL 或 PostgreSQL 兼容的数据库实例，以及一个集群卷，该集群卷容纳跨三个可用区复制的数据库集群的数据作为一个单一虚拟卷。默认情况下，Aurora 数据库集群包含一个主数据库实例（执行读取和写入）以及最多 15 个可选的 Aurora 副本（读取器数据库实例）。有关 Aurora 数据库集群的更多信息，请参阅Amazon Aurora 数据库集群。

Aurora 有两种主要的数据库集群类型：

- Aurora 预调配 - 您可以根据预期的工作负载为写入器和读取器实例选择数据库实例类。有关更多信息，请参阅 Amazon Aurora 数据库实例类。Aurora 预调配有多个选项，包括 Aurora Global Database。有关更多信息，请参阅 使用 Amazon Aurora Global Database。
- Aurora Serverless – Aurora Serverless v2 是 Aurora 的按需自动扩缩配置。容量会根据应用程序需求自动调整。您只需为您的数据库集群使用的资源付费。这种自动化对于工作负载变化很大且不可预测的环境特别有用。有关更多信息，请参阅 使用 Aurora Serverless v2。

在下文中，您可以了解如何创建 Aurora 数据库集群。要了解其用法，请先参阅数据库集群先决条件。

有关连接到 Aurora 数据库集群的说明，请参阅 连接到 Amazon Aurora 数据库集群。

目录

- 数据库集群先决条件
  - 为数据库集群配置网络
    - 配置与 EC2 实例的自动网络连接
    - 手动配置网络
  - 其他先决条件
- 创建数据库集群
  - 创建主（写入器）数据库实例
- Aurora 数据库集群的设置
- 不适用于 Amazon Aurora 的数据库集群设置
- 不适用于 Amazon Aurora 数据库实例的设置

数据库集群先决条件

> **Important**
>
> 您必须先完成为 Amazon Aurora 设置环境一节中的任务，然后才能创建 Aurora 数据库集群。

以下是创建数据库集群之前需要完成的先决条件。

主题

- 为数据库集群配置网络
- 其他先决条件

为数据库集群配置网络

在具有至少两个可用区的 AWS 区域中，您只能在基于 Amazon VPC 服务的虚拟私有云（VPC）中创建一个 Amazon Aurora 数据库集群。为数据库集群选择的数据库子网组必须包含至少两个可用区。在极少数情况下，在发生可用区故障时，该配置确保数据库集群始终具有至少一个可用于失效转移的数据库实例。

如果您计划在同一 VPC 中的新数据库集群和 EC2 实例之间建立连接，则可以在创建数据库集群时这样做。如果您计划从同一 VPC 中的 EC2 实例以外的资源连接到数据库集群，则可以手动配置网络连接。

主题

- 配置与 EC2 实例的自动网络连接
- 手动配置网络

配置与 EC2 实例的自动网络连接

创建 Aurora 数据库集群时，可以使用 AWS Management Console在 Amazon EC2 实例和新数据库集群之间建立连接。当您这样做时，RDS 会自动配置您的 VPC 和网络设置。数据库集群在与 EC2 实例相同的 VPC 中创建，以便 EC2 实例可以访问数据库集群。

以下是将 EC2 实例与数据库集群连接的要求：

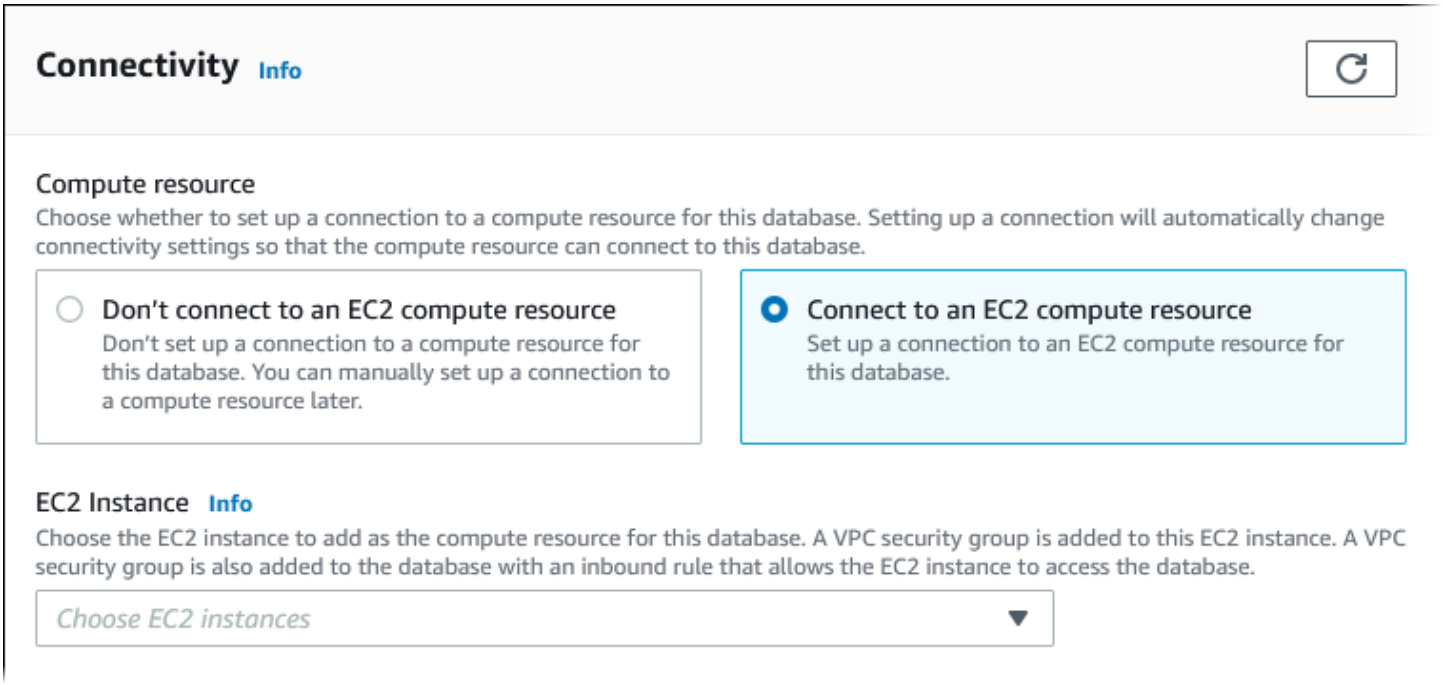
- 在创建数据库集群之前，EC2 实例必须存在于 AWS 区域中。

如果 AWS 区域中不存在任何 EC2 实例，控制台将提供创建一个此类实例的链接。

- 目前，数据库集群不能是 Aurora Serverless 数据库集群，也不能是 Aurora 全局数据库的一部分。
- 创建数据库实例的用户必须具有执行以下操作的权限：
  - `ec2:AssociateRouteTable`
  - `ec2:AuthorizeSecurityGroupEgress`
  - `ec2:AuthorizeSecurityGroupIngress`
  - `ec2:CreateRouteTable`
  - `ec2:CreateSubnet`
  - `ec2:CreateSecurityGroup`
  - `ec2:DescribeInstances`
  - `ec2:DescribeNetworkInterfaces`
  - `ec2:DescribeRouteTables`
  - `ec2:DescribeSecurityGroups`
  - `ec2:DescribeSubnets`
  - `ec2:ModifyNetworkInterfaceAttribute`
  - `ec2:RevokeSecurityGroupEgress`

使用此选项创建私有数据库集群。数据库集群使用仅包含私有子网的数据库子网组，来限制对 VPC 内资源的访问。

要将 EC2 实例连接到数据库集群，请在 Create database（创建数据库）页面上的 Connectivity（连接）部分，选择 Connect to an EC2 compute resource（连接到 EC2 计算资源）。



当您选择 Connect to an EC2 compute resource（连接到 EC2 计算资源）时，RDS 会自动设置以下选项。除非您通过选择 Don't connect to an EC2 compute resource（不要连接到 EC2 计算资源）来选择不与 EC2 实例建立连接，否则您无法更改这些设置。

控制台选项	自动设置
网络类型	RDS 将网络类型设置为 IPv4。当前，在 EC2 实例和数据库集群之间建立连接时，不支持双堆栈模式。
Virtual Private Cloud (VPC)	RDS 将 VPC 设置为与 EC2 实例关联的 VPC。
DB subnet group（数据库子网组）	RDS 要求在与 EC2 实例相同的可用区中具有带私有子网的数据库子网组。如果存在符合此要求的数据库子网组，则 RDS 将使用现有的数据库子网组。默认情况下，此选项设置为 Automatic setup（自动设置）。 当您选择 Automatic setup（自动设置）但没有满足此要求的数据库子网组时，将执行以下操作。RDS 在三个可用区中使用三个可用的私有子网，其中一个可用区与 EC2 实例相同。如果私有子网在可用区中不可用，则 RDS 会在可用区中创建私有子网。然后，RDS 创建数据库子网组。

控制台选项	自动设置
	当私有子网可用时，RDS 使用与该子网关联的路由表，并将它创建的任何子网添加到该路由表中。当没有可用的私有子网时，RDS 会创建一个没有互联网网关访问权限的路由表，并将它创建的子网添加到该路由表中。 RDS 还允许您使用现有的数据库子网组。如果您想使用您选择的现有数据库子网组，请选择 Choose existing（选择现有）。
公有访问权限	RDS 选择 No（否），以使数据库集群不可供公开访问。 出于安全考虑，最好实践是保持数据库为私有，并确保不能从互联网访问数据库。
VPC security group (firewall) [VPC 安全组（防火墙）]	RDS 创建与数据库集群关联的新安全组。安全组命名为 `rds-ec2-`*n*，其中 *n* 是一个数值。该安全组包含一条以 EC2 VPC 安全组（防火墙）作为源的入站规则。这个与数据库集群关联的安全组允许 EC2 实例访问数据库集群。 RDS 还会创建一个与 EC2 实例关联的新安全组。安全组命名为 `ec2-rds-`*n*，其中 *n* 是一个数值。该安全组包含一条以数据库集群的 VPC 安全组作为源的出站规则。该安全组允许 EC2 实例向数据库集群发送流量。 您可以通过选择 Create new（新建）并键入新安全组的名称，添加另一个新安全组。 您可以通过选择 Choose existing（选择现有）并选择要添加的安全组，添加现有安全组。

控制台选项	自动设置
可用区	如果您在创建数据库集群（单可用区部署）期间，没有在 Availability & durability（可用性和耐久性）中创建 Aurora 副本，RDS 会选择 EC2 实例的可用区。 当您在创建数据库集群（多可用区部署）期间创建 Aurora 副本时，RDS 会为数据库集群中的一个数据库实例选择 EC2 实例的可用区。RDS 将为数据库集群中的其他数据库实例随机选择不同的可用区。主数据库实例或 Aurora 副本是在与 EC2 实例相同的可用区中创建的。如果主数据库实例和 EC2 实例位于不同的可用区，则可能会产生跨可用区成本。

有关这些设置的更多信息，请参阅 Aurora 数据库集群的设置。

如果您在创建数据库集群后对这些设置进行任何更改，则这些更改可能会影响 EC2 实例与数据库集群之间的连接。

手动配置网络

如果您计划从同一 VPC 中的 EC2 实例以外的资源连接到数据库集群，则可以手动配置网络连接。如果您使用 AWS Management Console创建数据库集群，您可以让 Amazon RDS 自动为您创建 VPC。或者，您也可以使用现有 VPC 或为 Aurora 数据库集群创建新的 VPC。不论采用哪种方法，要在 Amazon Aurora 数据库集群中使用您的 VPC，该 VPC 必须最少在两个可用区中各拥有至少一个子网。

默认情况下，Amazon RDS 会在可用区中自动为您创建主数据库实例和 Aurora 副本。要选择特定可用区，您需要将 Availability & durability（可用性和持久性）多可用区部署设置更改为 Don't create an Aurora Replica（不创建 Aurora 副本）。这样做会公开 Availability Zone（可用区）设置，此设置可让您从 VPC 内的可用区中进行选择。但是，我们强烈建议您保留默认设置，让 Amazon RDS 创建多可用区部署并为您选择可用区。这样一来，创建的 Aurora 数据库集群将具有快速失效转移和高可用性功能，这是 Aurora 的两大主要优势。

如果没有默认 VPC 或尚未创建 VPC，您可以在使用控制台创建数据库集群时，让 Amazon RDS 自动为您创建 VPC。否则，您必须执行以下操作：

- 在要部署数据库集群的 AWS 区域中，创建最少在两个可用区中均至少有一个子网的 VPC。有关更多信息，请参阅在 VPC 中使用数据库集群和教程：创建 VPC 以用于数据库集群（仅限 IPv4）。

- 指定授权与您的 数据库集群的连接的 VPC 安全组。有关更多信息，请参阅通过创建安全组提供对 VPC 中数据库集群的访问和使用安全组控制访问权限。
- 指定 RDS 数据库子网组，该子网组在 VPC 中定义至少两个可由 数据库集群使用的子网。有关更多信息，请参阅 使用数据库子网组。

有关 VPC 的信息，请参阅 Amazon VPC 和 Amazon Aurora。有关为私有数据库集群配置网络的教程，请参阅教程：创建 VPC 以用于数据库集群（仅限 IPv4）。

如果您想连接到与 Aurora 数据库集群不在同一 VPC 中的资源，请参阅 在 VPC 中访问数据库集群的场景 中的相应方案。

其他先决条件

在创建数据库集群之前，请考虑以下附加先决条件：

- 如果使用 AWS Identity and Access Management（IAM）凭证连接到AWS，您的AWS账户必须拥有 IAM 策略来授予执行 Amazon RDS 操作所需的权限。有关更多信息，请参阅 Amazon Aurora 的 Identity and Access Management。

  如果要使用 IAM 访问 Amazon RDS 控制台，必须先使用您的用户凭证登录 AWS Management Console。然后通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
- 要定制您的数据库集群的配置参数，必须通过必需的参数设置来指定数据库集群参数组和数据库参数组。有关创建或修改数据库集群参数组或数据库参数组的信息，请参阅 Amazon Aurora 的参数组。
- 请确定要为数据库集群指定的 TCP/IP 端口号。有些公司的防火墙不允许连接到 Aurora 的默认端口 (MySQL 为 3306，PostgreSQL 为 5432)。如果您的公司防火墙阻止默认端口，请为数据库集群选择其他端口。数据库集群中的所有实例使用相同的端口。
- 如果数据库的主要引擎版本已到 RDS 标准支持终止日期，则必须使用扩展支持 CLI 选项或 RDS API 参数。有关更多信息，请参阅Aurora 数据库集群的设置中的 RDS 扩展支持。

创建数据库集群

您可以使用AWS Management Console、AWS CLI 或 RDS API 创建 Aurora 数据库集群。

控制台

您可以在启用或未启用轻松创建的情况下使用 AWS Management Console创建数据库集群。启用 Easy create (轻松创建) 的情况下，您可以仅指定数据库引擎类型、数据库实例大小和数据库实例标

识符。Easy create (轻松创建) 为其他配置选项使用默认设置。未启用 Easy create (轻松创建) 的情况下，您在创建数据库时需要指定更多配置选项，包括用于可用性、安全性、备份和维护的选项。

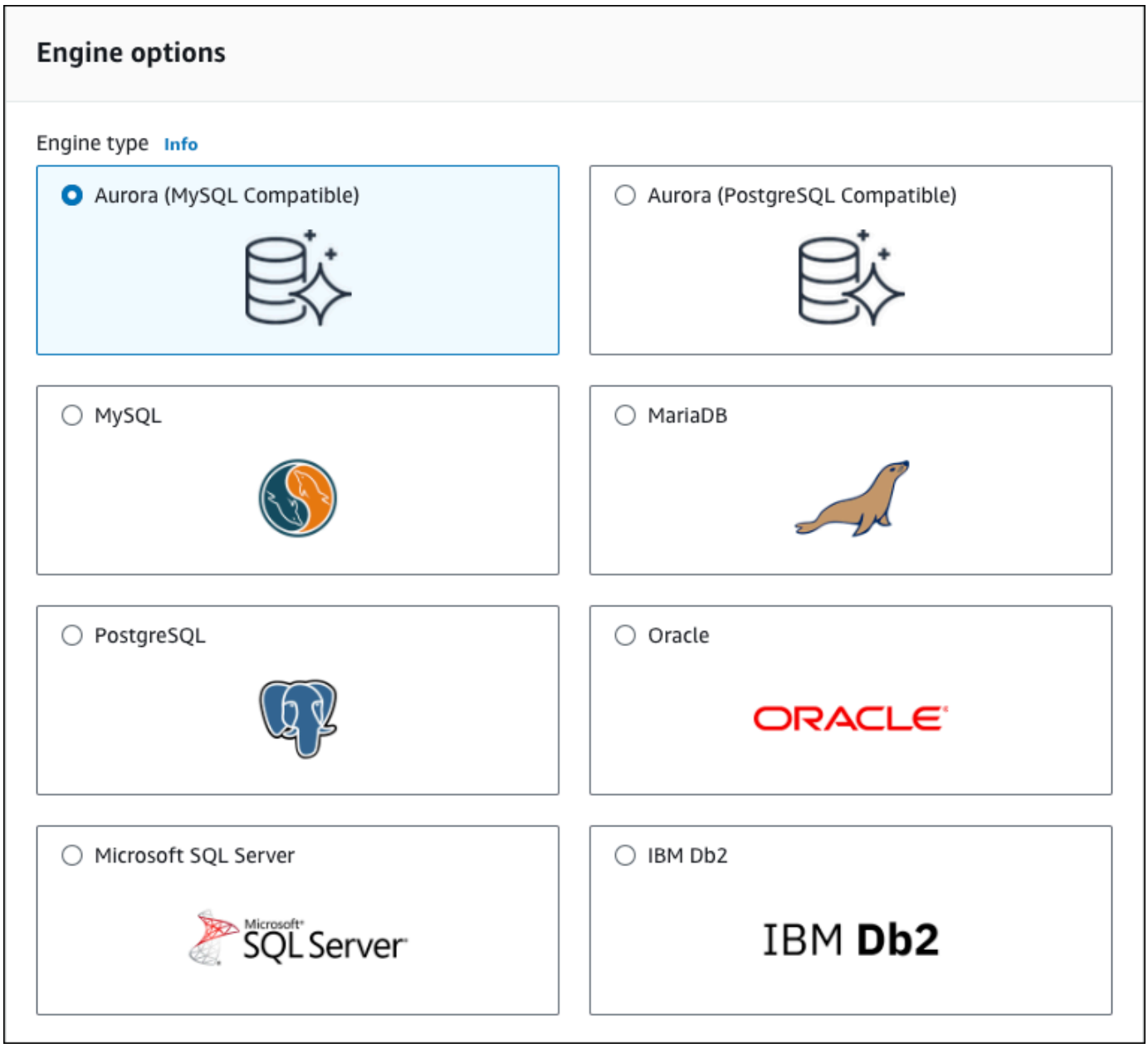
> **Note**
>
> 对于该示例，启用了 Standard create (标准创建)，并且未启用 Easy create (轻松创建)。有关在启用轻松创建的情况下创建数据库集群的信息，请参阅开始使用 Amazon Aurora。

使用控制台创建 Aurora 数据库集群

1. 登录AWS Management Console并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在 AWS Management Console 的右上角，选择要在其中创建数据库集群的 AWS 区域。

   Aurora 尚未在所有AWS区域提供。有关可以使用 Aurora 的AWS区域列表，请参阅区域可用性。
3. 在导航窗格中，选择 Databases (数据库)。
4. 选择创建数据库。
5. 对于选择数据库创建方法，选择标准创建。
6. 对于引擎类型，选择下列选项之一：

   - Aurora（MySQL 兼容）
   - Aurora（PostgreSQL 兼容）



7. 选择引擎版本。

   有关更多信息，请参阅 Amazon Aurora 版本。您可以使用筛选条件来选择与所需功能兼容的版本，例如 Aurora Serverless v2。有关更多信息，请参阅 使用 Aurora Serverless v2。

8. 在 Templates (模板) 中，选择与您的使用案例匹配的模板。

9. 要输入主密码，请执行以下操作：

   a. 在设置部分，展开凭证设置。

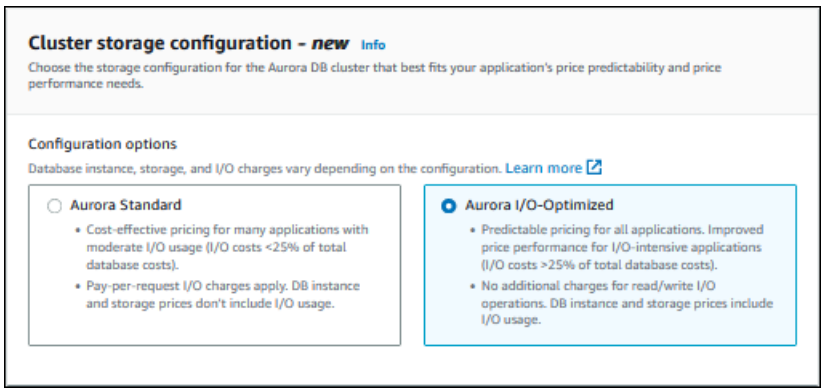
b. 清除自动生成密码复选框。

c. (可选) 更改主用户名值，并在主密码和确认密码中输入相同的密码。

默认情况下，新数据库实例对主用户使用自动生成的密码。

10. 在连接部分的 VPC 安全组（防火墙）下，如果您选择新建，则会创建一个 VPC 安全组，其入站规则允许本地计算机的 IP 地址访问该数据库。

11. 对于集群存储配置，请选择 Aurora I/O-Optimized 或 Aurora Standard。有关更多信息，请参阅 Amazon Aurora 数据库集群的存储配置。



12. （可选）为该数据库集群设置与计算资源的连接。

在创建数据库集群期间，您可以配置 Amazon EC2 实例和新数据库集群之间的连接。有关更多信息，请参阅 配置与 EC2 实例的自动网络连接。

13. 对于其余部分，请指定数据库集群设置。有关每项设置的信息，请参阅 Aurora 数据库集群的设置。

14. 选择创建数据库。

如果选择使用自动生成的密码，则数据库页面上将显示查看凭证详细信息按钮。

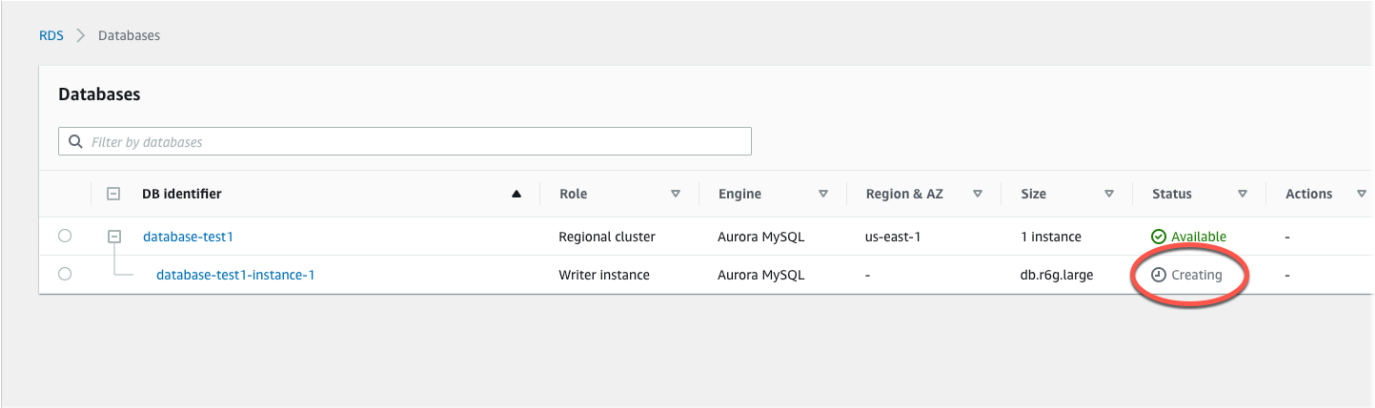
要查看数据库集群的主用户名和密码，请选择查看凭证详细信息。

要以主用户身份连接到数据库实例，请使用显示的用户名和密码。

> **Important**
>
> 您无法再次查看主用户密码。如果您不记录它，您可能需要更改它。如果需要在数据库实例可用后更改主用户密码，则可以修改数据库实例以执行此操作。有关修改 数据库实例的更多信息，请参阅 修改 Amazon Aurora 数据库集群。

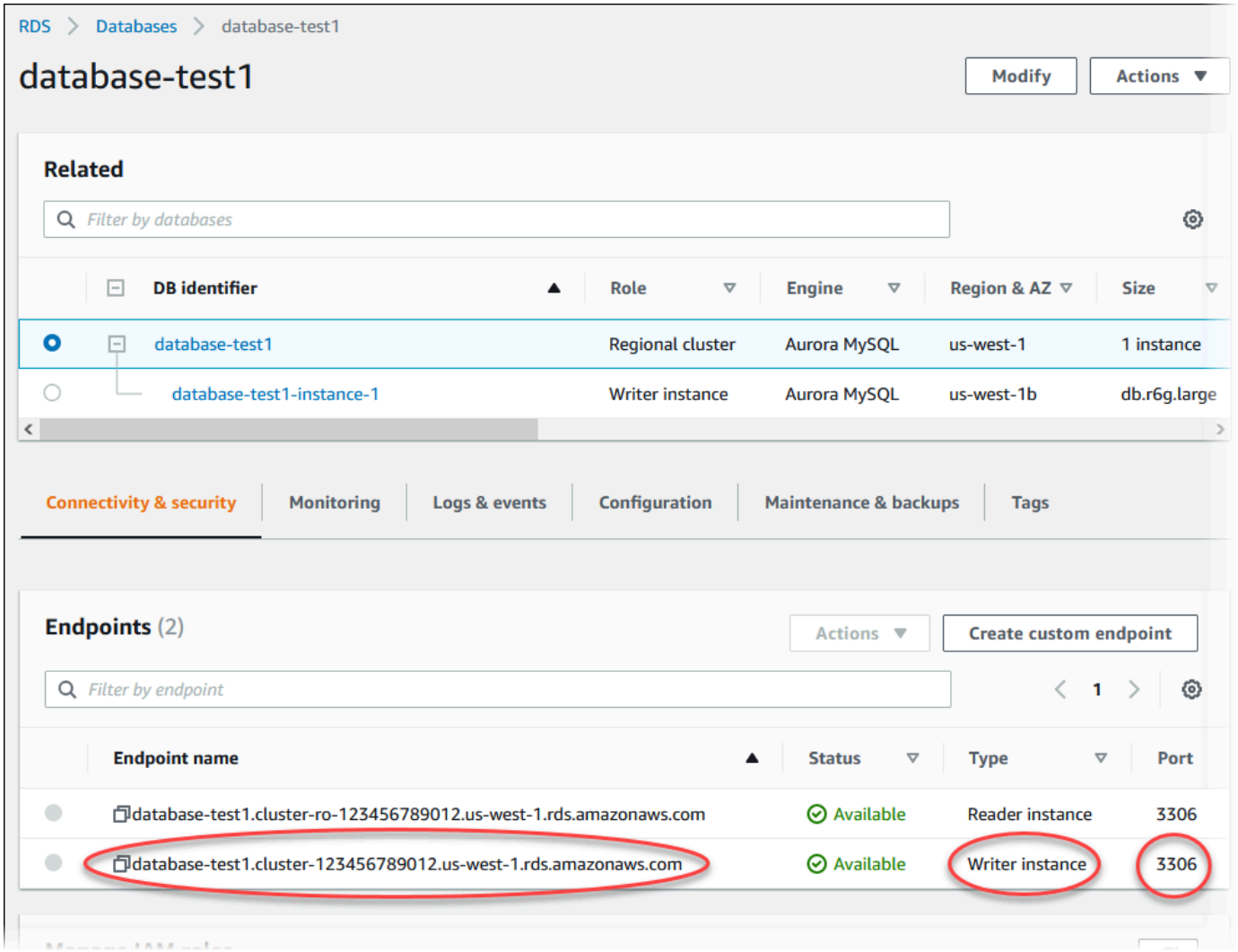
15. 对于 Databases (数据库)，选择新 Aurora 数据库集群的名称。

    在 RDS 控制台上，将显示新数据库集群的详细信息。在此数据库集群可供使用之前，数据库集群及其数据库实例的状态为 creating (正在创建)。



    当状态变为 available (可用) 时，您可以连接到该数据库集群。根据数据库实例类和存储量，新数据库集群可能需要等待 20 分钟时间才可用。

    要查看新创建的集群，请在 Amazon RDS 控制台导航窗格中选择 Databases (数据库)。然后选择要显示数据库集群详细信息的数据库集群。有关更多信息，请参阅“查看 Amazon Aurora 数据库集群”。



在 Connectivity & security (连接和安全性) 选项卡上，记下写入器数据库实例的端口和端点。在执行写入或读取操作的任何应用程序的 JDBC 和 ODBC 连接字符串中，请使用集群的端点和端口。

AWS CLI

> **Note**
>
> 在使用 AWS CLI 创建 Aurora 数据库集群之前，您必须满足所需的先决条件，例如，创建 VPC 和 RDS 数据库子网组。有关更多信息，请参阅“数据库集群先决条件”。

您可以使用 AWS CLI 创建 Aurora MySQL 数据库集群或 Aurora PostgreSQL 数据库集群。

使用 AWS CLI 创建 Aurora MySQL 数据库集群

在创建 Aurora MySQL 8.0 兼容版或 5.7 兼容版数据库集群或数据库实例时，您必须为 `aurora-mysql` 选项指定 `--engine`。

完成以下步骤：

1. 为新的数据库集群指定数据库子网组和 VPC 安全组 ID，然后调用 create-db-cluster AWS CLI 命令以创建 Aurora MySQL 数据库集群。

   例如，以下命令创建一个名为 `sample-cluster` 的新的 MySQL 8.0 兼容数据库集群。集群使用默认的引擎版本和 Aurora I/O-Optimized 存储类型。

   对于 Linux、macOS 或 Unix：

   ```
   aws rds create-db-cluster --db-cluster-identifier sample-cluster \
       --engine aurora-mysql --engine-version 8.0 \
       --storage-type aurora-iopt1 \
       --master-username user-name --manage-master-user-password \
       --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2
   ```

   对于 Windows：

   ```
   aws rds create-db-cluster --db-cluster-identifier sample-cluster ^
       --engine aurora-mysql --engine-version 8.0 ^
       --storage-type aurora-iopt1 ^
       --master-username user-name --manage-master-user-password ^
       --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2
   ```

   以下命令创建一个名为 `sample-cluster` 的新 MySQL 5.7 兼容版数据库集群。集群使用默认的引擎版本和 Aurora Standard 存储类型。

   对于 Linux、macOS 或 Unix：

   ```
   aws rds create-db-cluster --db-cluster-identifier sample-cluster  \
       --engine aurora-mysql --engine-version 5.7 \
       --storage-type aurora \
       --master-username user-name --manage-master-user-password \
       --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2
   ```

   对于 Windows：

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster sample-cluster  ^
    --engine aurora-mysql --engine-version 5.7 ^
    --storage-type aurora ^
    --master-username user-name --manage-master-user-password ^
    --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2
```

2. 如果您使用控制台创建数据库集群，则 Amazon RDS 为您的数据库集群自动创建主实例 (写入器)。如果您使用 AWS CLI 创建数据库集群，则必须明确为数据库集群创建主实例。主实例是在数据库集群中创建的第一个实例。在创建主数据库实例之前，数据库集群端点将保持 `Creating` 状态。

   调用 create-db-instance AWS CLI 命令，以便为数据库集群创建主实例。包括数据库集群名称以作为 `--db-cluster-identifier` 选项值。

   > **Note**
   >
   > 您无法为数据库实例设置 `--storage-type` 选项。您只能为数据库集群设置此选项。

   例如，以下命令创建一个名为 `sample-instance` 的新的 MySQL 5.7 兼容版或 MySQL 8.0 兼容版数据库实例。

   对于 Linux、macOS 或 Unix：

   ```
   aws rds create-db-instance --db-instance-identifier sample-instance \
        --db-cluster-identifier sample-cluster --engine aurora-mysql --db-instance-
   class db.r5.large
   ```

   对于 Windows：

   ```
   aws rds create-db-instance --db-instance-identifier sample-instance ^
        --db-cluster-identifier sample-cluster --engine aurora-mysql --db-instance-
   class db.r5.large
   ```

使用 AWS CLI 创建 Aurora PostgreSQL 数据库集群

1. 为新的数据库集群指定数据库子网组和 VPC 安全组 ID，然后调用 create-db-cluster AWS CLI 命令以创建 Aurora PostgreSQL 数据库集群。

例如，以下命令创建名为 `sample-cluster` 的新数据库集群。集群使用默认的引擎版本和 Aurora I/O-Optimized 存储类型。

对于 Linux、macOS 或 Unix：

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster \
    --engine aurora-postgresql \
    --storage-type aurora-iopt1 \
    --master-username user-name --manage-master-user-password \
    --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2
```

对于 Windows：

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster ^
    --engine aurora-postgresql ^
    --storage-type aurora-iopt1 ^
    --master-username user-name --manage-master-user-password ^
    --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2
```

2. 如果您使用控制台创建数据库集群，则 Amazon RDS 为您的数据库集群自动创建主实例 (写入器)。如果您使用 AWS CLI 创建数据库集群，则必须明确为数据库集群创建主实例。主实例是在数据库集群中创建的第一个实例。在创建主数据库实例之前，数据库集群端点将保持 `Creating` 状态。

   调用 create-db-instance AWS CLI 命令，以便为数据库集群创建主实例。包括数据库集群名称以作为 `--db-cluster-identifier` 选项值。

   对于 Linux、macOS 或 Unix：

   ```
   aws rds create-db-instance --db-instance-identifier sample-instance \
        --db-cluster-identifier sample-cluster --engine aurora-postgresql --db-
   instance-class db.r5.large
   ```

   对于 Windows：

   ```
   aws rds create-db-instance --db-instance-identifier sample-instance ^
        --db-cluster-identifier sample-cluster --engine aurora-postgresql --db-
   instance-class db.r5.large
   ```

这些示例指定了生成主用户密码并在 Secrets Manager 中对其进行管理的 `--manage-master-user-password` 选项。有关更多信息，请参阅 使用 Amazon Aurora 和 AWS Secrets Manager 管理密码。或者，您可以使用 `--master-password` 选项自行指定和管理密码。

RDS API

> **Note**
>
> 在使用 AWS CLI 创建 Aurora 数据库集群之前，您必须满足所需的先决条件，例如，创建 VPC 和 RDS 数据库子网组。有关更多信息，请参阅 数据库集群先决条件。

为新的数据库集群指定数据库子网组和 VPC 安全组 ID，然后调用 CreateDBCluster 操作以创建数据库集群。

在创建 Aurora MySQL 版本 2 或 3 数据库集群或数据库实例时，请为 `Engine` 参数指定 `aurora-mysql`。

在创建 Aurora PostgreSQL 数据库集群或数据库实例时，请为 `aurora-postgresql` 参数指定 `Engine`。

如果您使用控制台创建数据库集群，则 Amazon RDS 为您的数据库集群自动创建主实例 (写入器)。如果使用 RDS API 创建数据库集群，则必须使用 CreateDBInstance 显式创建数据库集群的主实例。主实例是在数据库集群中创建的第一个实例。在创建主数据库实例之前，数据库集群端点将保持 `Creating` 状态。

创建主（写入器）数据库实例

如果您使用 AWS Management Console创建数据库集群，则 Amazon RDS 为您的数据库集群自动创建主实例（写入器）。如果您使用 AWS CLI 或 RDS API 创建数据库集群，则必须显式为数据库集群创建主实例。主实例是在数据库集群中创建的第一个实例。在创建主数据库实例之前，数据库集群端点将保持 `Creating` 状态。

有关更多信息，请参阅 创建数据库集群。

> **Note**
>
> 如果您的数据库集群没有写入器数据库实例（也称为无头集群），则无法使用控制台创建写入器实例。必须使用 AWS CLI 或 RDS API。

以下示例使用 create-db-instance AWS CLI 命令为名为 `headless-test` 的 Aurora PostgreSQL 数据库集群创建写入器实例。

```
aws rds create-db-instance \
    --db-instance-identifier no-longer-headless \
    --db-cluster-identifier headless-test \
    --engine aurora-postgresql \
    --db-instance-class db.t4g.medium
```

Aurora 数据库集群的设置

下表包含您在创建 Aurora 数据库集群时所选设置的详细信息。

控制台设置	设置说明	CLI 选项和 RDS API 参数
自动次要版本升级	如果要在数据库引擎的首选次要版本升级可用时，让您的 Aurora 数据库集群自动接收这些升级，请选择 Enable auto minor version upgrade (启用自动次要版本升级)。 自动次要版本升级设置同时适用于 Aurora PostgreSQL 和 Aurora MySQL 数据库集群。 有关 Aurora PostgreSQL 引擎更新的更多信息，请参阅 Amazon Aurora PostgreSQL 的数据库引擎更新。 有关 Aurora MySQL 引擎更新的更多信息，请参阅 Amazon Aurora MySQL 的数据库引擎更新。	为 Aurora 集群中的每个数据库实例设置此值。如果对集群中的任何数据库实例禁用了此设置，则不会自动升级集群。 通过使用 AWS CLI 来运行 `create-db-instance` 并设置 `--auto-minor-version-upgrade|--no-auto-minor-version-upgrade` 选项。 通过使用 RDS API 来调用 `CreateDBInstance` 并设置 `AutoMinorVersionUpgrade` 参数。
AWS KMS key	仅当加密设置为启用加密时可用。选择 AWS KMS key 用于加密该数据库集群。有关更多信息，请参阅加密 Amazon Aurora 资源。	通过使用 AWS CLI 来运行 `create-db-cluster` 并设置 `--kms-key-id` 选项。

控制台设置	设置说明	CLI 选项和 RDS API 参数
		通过使用 RDS API 来调用 CreateDBCluster 并设置 KmsKeyId 参数。
回溯	仅适用于 Aurora MySQL。选择 Enable Backtrack (启用回溯) 可启用回溯，选择 Disable Backtrack (禁用回溯) 可禁用回溯。使用回溯可以将数据库集群倒回到特定时间，而无需创建新的数据库集群。默认情况下它是禁用的。如果启用回溯，则还可以指定希望能够回溯您的数据库集群的时间长度 (目标回溯时段)。有关更多信息，请参阅回溯 Aurora 数据库集群。	通过使用 AWS CLI 来运行 create-db-cluster 并设置 --backtrack-window 选项。 通过使用 RDS API 来调用 CreateDBCluster 并设置 BacktrackWindow 参数。
证书颁发机构	数据库集群中的数据库实例使用的服务器证书的证书颁发机构（CA）。 有关更多信息，请参阅 使用 SSL/TLS 加密与数据库集群的连接。	通过使用 AWS CLI 来运行 create-db-instance 并设置 --ca-certificate-identifier 选项。 通过使用 RDS API 来调用 CreateDBInstance 并设置 CACertificateIdentifier 参数。
集群存储配置	数据库集群的存储类型：Aurora I/O-Optimized 或 Aurora Standard。 有关更多信息，请参阅 Amazon Aurora 数据库集群的存储配置。	通过使用 AWS CLI 来运行 create-db-cluster 并设置 --storage-type 选项。 通过使用 RDS API 来调用 CreateDBCluster 并设置 StorageType 参数。

控制台设置	设置说明	CLI 选项和 RDS API 参数
将标签复制到快照	选择该选项可在创建快照时将任何数据库实例标签复制到数据库快照。 有关更多信息，请参阅 为 Amazon Aurora 和 Amazon RDS 资源添加标签。	通过使用 AWS CLI 来运行 `create-db-cluster` 并设置 `--copy-tags-to-snapshot` | `--no-copy-tags-to-snapshot` 选项。 通过使用 RDS API 来调用 `CreateDBCluster` 并设置 `CopyTagsToSnapshot` 参数。

控制台设置	设置说明	CLI 选项和 RDS API 参数
数据库身份验证	您想要使用的数据库身份验证。 适用于 MySQL： • 选择密码身份验证以仅使用数据库密码验证数据库用户的身份。 • 选择密码和 IAM 数据库身份验证以使用 IAM 用户和角色的数据库密码和用户凭证验证数据库用户的身份。有关更多信息，请参阅“IAM 数据库身份验证”。 适用于 PostgreSQL： • 选择 IAM database authentication（IAM 数据库身份验证），以通过用户和角色使用数据库密码和用户凭证验证数据库用户的身份。有关更多信息，请参阅 IAM 数据库身份验证。 • 选择 Kerberos 身份验证以使用 Kerberos 身份验证来验证数据库密码和用户凭证。有关更多信息，请参阅 在 Aurora PostgreSQL 中使用 Kerberos 身份验证。	要在 AWS CLI 中使用 IAM 数据库身份验证，请运行 `create-db-cluster` 并设置 `--enable-iam-database-authentication | --no-enable-iam-database-authentication` 选项。 要将 IAM 数据库身份验证与 RDS API 搭配使用，请调用 `CreateDBCluster` 并设置 `EnableIAMDatabaseAuthentication` 参数。 要在 AWS CLI 中使用 Kerberos 身份验证，请运行 `create-db-cluster` 并设置 `--domain` 和 `--domain-iam-role-name` 选项。 要将 Kerberos 身份验证与 RDS API 搭配使用，请调用 `CreateDBCluster` 并设置 `Domain` 和 `DomainIAMRoleName` 参数。

控制台设置	设置说明	CLI 选项和 RDS API 参数
数据库端口	指定应用程序和实用程序用来访问数据库的端口。默认情况下，Aurora MySQL 数据库集群使用默认 MySQL 端口 3306；Aurora PostgreSQL 数据库集群使用默认 PostgreSQL 端口 5432。有些公司的防火墙不允许连接到这些默认端口。如果您的公司防火墙阻止使用默认端口，请为新数据库集群选择其他端口。	通过使用 AWS CLI 来运行 `create-db-cluster` 并设置 `--port` 选项。 通过使用 RDS API 来调用 `CreateDBCluster` 并设置 `Port` 参数。
数据库集群标识符	为数据库集群输入一个名称，该名称在所选的 AWS 区域中对于您的账户是唯一的。此标识符在数据库集群的集群端点地址中使用。有关集群端点的信息，请参阅 Amazon Aurora 端点连接。 数据库集群标识符具有以下限制： • 它必须包含 1 到 63 个字母数字字符或连字符。 • 它的第一个字符必须是字母。 • 它不能以连字符结束或包含两个连续连字符。 • 它对于每个 AWS 区域的每个 AWS 账户的所有数据库集群必须是唯一的。	通过使用 AWS CLI 来运行 `create-db-cluster` 并设置 `--db-cluster-identifier` 选项。 通过使用 RDS API 来调用 `CreateDBCluster` 并设置 `DBClusterIdentifier` 参数。

控制台设置	设置说明	CLI 选项和 RDS API 参数
数据库集群参数组	选择数据库集群参数组。Aurora 具有一个可使用的默认数据库集群参数组，您也可以创建自己的数据库集群参数组。有关数据库集群参数组的更多信息，请参阅 Amazon Aurora 的参数组。	通过使用 AWS CLI 来运行 create-db-cluster 并设置 --db-cluster-parameter-group-name 选项。 通过使用 RDS API 来调用 CreateDBCluster 并设置 DBClusterParameterGroupName 参数。
数据库实例类	仅适用于已配置的容量类型。选择定义数据库集群中每个实例的处理和内存要求的数据库实例类。有关数据库实例类的更多信息，请参阅 Amazon Aurora 数据库实例类。	为 Aurora 集群中的每个数据库实例设置此值。 通过使用 AWS CLI 来运行 create-db-instance 并设置 --db-instance-class 选项。 通过使用 RDS API 来调用 CreateDBInstance 并设置 DBInstanceClass 参数。
数据库参数组	选择参数组。Aurora 具有一个可使用的默认参数组，您也可以创建自己的参数组。有关参数组的更多信息，请参阅 Amazon Aurora 的参数组。	为 Aurora 集群中的每个数据库实例设置此值。 通过使用 AWS CLI 来运行 create-db-instance 并设置 --db-parameter-group-name 选项。 通过使用 RDS API 来调用 CreateDBInstance 并设置 DBParameterGroupName 参数。

控制台设置	设置说明	CLI 选项和 RDS API 参数
DB subnet group（数据库子网组）	要用于数据库集群的数据库子网组。 选择 Choose existing（选择现有）以使用现有的数据库子网组。然后，从 Existing DB subnet groups（现有数据库子网组）下拉列表中选择所需的子网组。 选择 Automatic setup（自动设置），让 RDS 选择兼容的数据库子网组。如果不存在任何子网组，RDS 会为您的集群创建一个新的子网组。 有关更多信息，请参阅 数据库集群先决条件。	通过使用 AWS CLI 来运行 create-db-cluster 并设置 --db-subnet-group-name 选项。 通过使用 RDS API 来调用 CreateDBCluster 并设置 DBSubnetGroupName 参数。
启用删除保护	选择 Enable deletion protection (启用删除保护) 以禁止删除数据库集群。如果使用控制台创建生产数据库集群，将默认启用删除保护。	通过使用 AWS CLI 来运行 create-db-cluster 并设置 --deletion-protection | --no-deletion-protection 选项。 通过使用 RDS API 来调用 CreateDBCluster 并设置 DeletionProtection 参数。
启用加密	选择 Enable encryption 可对此数据库集群启用静态加密。有关更多信息，请参阅 加密 Amazon Aurora 资源。	通过使用 AWS CLI 来运行 create-db-cluster 并设置 --storage-encrypted | --no-storage-encrypted 选项。 通过使用 RDS API 来调用 CreateDBCluster 并设置 StorageEncrypted 参数。

控制台设置	设置说明	CLI 选项和 RDS API 参数
启用增强监控	选择启用增强监控可启用您的数据库集群在其上运行的操作系统的实时指标收集。有关更多信息，请参阅 使用增强监控来监控操作系统指标。	为 Aurora 集群中的每个数据库实例设置这些值。 通过使用 AWS CLI 来运行 create-db-instance 并设置 --monitoring-interval 和 --monitoring-role-arn 选项。 通过使用 RDS API 来调用 CreateDBInstance 并设置 MonitoringInterval 和 MonitoringRoleArn 参数。
启用 RDS 数据 API	选择启用 RDS 数据 API 以启用 RDS 数据 API（数据 API）。数据 API 为运行 SQL 语句提供了一个安全的 HTTP 端点，而且无需管理连接。有关更多信息，请参阅 使用 Amazon RDS 数据 API。	通过使用 AWS CLI 来运行 create-db-cluster 并设置 --enable-http-endpoint | --no-enable-http-endpoint 选项。 通过使用 RDS API 来调用 CreateDBCluster 并设置 EnableHttpEndpoint 参数。
引擎类型	选择用于此数据库集群的数据库引擎。	通过使用 AWS CLI 来运行 create-db-cluster 并设置 --engine 选项。 通过使用 RDS API 来调用 CreateDBCluster 并设置 Engine 参数。

控制台设置	设置说明	CLI 选项和 RDS API 参数
引擎版本	仅适用于已配置的容量类型。选择数据库引擎的版本号。	通过使用 AWS CLI 来运行 `create-db-cluster` 并设置 `--engine-version` 选项。 通过使用 RDS API 来调用 `CreateDBCluster` 并设置 `EngineVersion` 参数。
Failover priority (失效转移优先级)	选择实例的失效转移优先级。如果您未选择值，则默认值为 tier-1。此优先级决定从主实例故障恢复时提升 Aurora 副本的顺序。有关更多信息，请参阅 Aurora 数据库集群的容错能力。	为 Aurora 集群中的每个数据库实例设置此值。 通过使用 AWS CLI 来运行 `create-db-instance` 并设置 `--promotion-tier` 选项。 通过使用 RDS API 来调用 `CreateDBInstance` 并设置 `PromotionTier` 参数。

控制台设置	设置说明	CLI 选项和 RDS API 参数
Initial database name (初始数据库名称)	输入默认数据库的名称。如果您未为 Aurora MySQL 数据库集群提供名称，则 Amazon RDS 不会在您创建的数据库集群上创建数据库。如果您未为 Aurora PostgreSQL 数据库集群提供名称，Amazon RDS 将创建一个名为 `postgres` 的数据库。 对于 Aurora MySQL，默认数据库名称具有以下约束： • 必须包含 1–64 个字母数字字符。 • 它不能是数据库引擎的保留字。 对于 Aurora PostgreSQL，默认数据库名称具有以下约束： • 必须包含 1–63 个字母数字字符。 • 必须以字母开头。后续字符可以是字母、下划线或数字 (0–9)。 • 它不能是数据库引擎的保留字。 要创建其他数据库，请连接到数据库集群并使用 SQL 命令 CREATE DATABASE。有关连接到数据库集群的更多信息，请参阅 连接到 Amazon Aurora 数据库集群。	通过使用 AWS CLI 来运行 `create-db-cluster` 并设置 `--database-name` 选项。 通过使用 RDS API 来调用 `CreateDBCluster` 并设置 `DatabaseName` 参数。

控制台设置	设置说明	CLI 选项和 RDS API 参数
日志导出	在 Log exports (日志导出) 部分，选择要开始发布到 Amazon CloudWatch Logs 的日志。有关将 Aurora MySQL 日志发布到 CloudWatch Logs 的更多信息，请参阅 将 Amazon Aurora MySQL 日志发布到 Amazon CloudWatch Logs。有关将 Aurora PostgreSQL 日志发布到 CloudWatch Logs 的更多信息，请参阅 将 Aurora PostgreSQL 日志发布到 Amazon CloudWatch Logs。	通过使用 AWS CLI 来运行 create-db-cluster 并设置 --enable-cloudwatch-logs-exports 选项。 通过使用 RDS API 来调用 CreateDBCluster 并设置 EnableCloudwatchLogsExports 参数。
维护窗口	选择 Select window (选择时段) 并指定可以进行系统维护的每周时间范围。或者，为 Amazon RDS 选择无首选项以随机分配一个时段。	通过使用 AWS CLI 来运行 create-db-cluster 并设置 --preferred-maintenance-window 选项。 通过使用 RDS API 来调用 CreateDBCluster 并设置 PreferredMaintenanceWindow 参数。

控制台设置	设置说明	CLI 选项和 RDS API 参数
在 AWS Secrets Manager 中管理主凭证	选择在 AWS Secrets Manager 中管理主凭证，以在 Secrets Manager 的密钥中管理主用户密码。 （可选）选择用于保护密钥的 KMS 密钥。请从您的账户的 KMS 密钥中进行选择，或输入来自其他账户的密钥。 有关更多信息，请参阅 使用 Amazon Aurora 和 AWS Secrets Manager 管理密码。	通过使用 AWS CLI 来运行 `create-db-cluster` 并设置 `--manage-master-user-password | --no-manage-master-user-password` 和 `--master-user-secret-kms-key-id` 选项。 通过使用 RDS API 来调用 `CreateDBCluster` 并设置 `ManageMasterUserPassword` 和 `MasterUserSecretKmsKeyId` 参数。
主密码	输入密码以登录到您的数据库集群： • 对于 Aurora MySQL，密码必须包含 8–41 个可打印 ASCII 字符。 • 对于 Aurora PostgreSQL，它必须包含 8–99 个可打印的 ASCII 字符。 • 它不能包含 /、"、@ 或空格。	通过使用 AWS CLI 来运行 `create-db-cluster` 并设置 `--master-user-password` 选项。 通过使用 RDS API 来调用 `CreateDBCluster` 并设置 `MasterUserPassword` 参数。

控制台设置	设置说明	CLI 选项和 RDS API 参数
主用户名	输入一个名称以用作主用户名，用于登录数据库集群： • 对于 Aurora MySQL，该名称必须包含 1–16 个字母数字字符。 • 对于 Aurora PostgreSQL，必须包含 1–63 个字母数字字符。 • 第一个字符必须是字母。 • 名称不能是数据库引擎的保留字。 您无法在创建数据库集群之后更改主用户名。	通过使用 AWS CLI 来运行 `create-db-cluster` 并设置 `--master-username` 选项。 通过使用 RDS API 来调用 `CreateDBCluster` 并设置 `MasterUsername` 参数。
多可用区部署	仅适用于已配置的容量类型。确定是否要在其他可用区中创建 Aurora 副本以实现失效转移支持。如果您选择在不同区域创建副本，Amazon RDS 将在不同可用区的数据库集群中创建 Aurora 副本，而不是在数据库集群的主实例中创建。有关多可用区的详细信息，请参阅区域及可用区。	通过使用 AWS CLI 来运行 `create-db-cluster` 并设置 `--availability-zones` 选项。 通过使用 RDS API 来调用 `CreateDBCluster` 并设置 `AvailabilityZones` 参数。

控制台设置	设置说明	CLI 选项和 RDS API 参数
网络类型	数据库集群支持的 IP 寻址协议。 IPv4，指定资源只能通过 IPv4 寻址协议与数据库集群通信。 Dual-stack mode（双堆栈模式），指定资源可通过 IPv4 和/或 IPv6 与数据库集群通信。如果您有任何必须通过 IPv6 寻址协议与数据库集群通信的资源，请使用双堆栈模式。要使用双堆栈模式，请确保至少有两个子网跨越两个同时支持 IPv4 和 IPv6 网络协议的可用区。此外，请确保将 IPv6 CIDR 块与您指定的数据库子网组中的子网进行关联。 有关更多信息，请参阅 Amazon Aurora IP 寻址。	通过使用 AWS CLI 来运行 `create-db-cluster` 并设置 `--network-type` 选项。 通过使用 RDS API 来调用 `CreateDBCluster` 并设置 `NetworkType` 参数。

控制台设置	设置说明	CLI 选项和 RDS API 参数
公有访问权限	选择可公开访问为数据库集群提供公有 IP 地址，或选择不可公开访问。数据库集群可以混合使用公有和私有数据库实例。有关隐藏实例以防止公开访问的更多信息，请参阅对互联网隐藏 VPC 中的数据库集群。 要从 Amazon VPC 外部连接到数据库实例，数据库实例必须可公开访问，必须使用数据库实例安全组的入站规则授予访问权限，并且必须满足其他要求。有关更多信息，请参阅“无法连接到 Amazon RDS 数据库实例”。 如果您的数据库实例不可公开访问，则您还可以使用AWS Site-to-Site VPN 连接或 AWS Direct Connect 连接从专用网络访问该实例。有关更多信息，请参阅 互联网络流量隐私。	为 Aurora 集群中的每个数据库实例设置此值。 通过使用 AWS CLI 来运行 `create-db-instance` 并设置 `--publicly-accessible | --no-publicly-accessible` 选项。 通过使用 RDS API 来调用 `CreateDBInstance` 并设置 `PubliclyAccessible` 参数。

控制台设置	设置说明	CLI 选项和 RDS API 参数
RDS 扩展支持	选择启用 RDS 扩展支持，以允许受支持的主要引擎版本在 Aurora 标准支持终止日期后继续运行。 创建数据库集群时，Amazon Aurora 默认使用 RDS 扩展支持。为了防止在 Aurora 标准支持终止日期后创建新的数据库集群并避免支付 RDS 扩展支持费用，请禁用此设置。在 RDS 扩展支持定价开始日期之前，您的现有数据库集群不会产生费用。 有关更多信息，请参阅 Amazon Aurora 的 Amazon RDS 扩展支持。	通过使用 AWS CLI 来运行 create-db-cluster 并设置 `--engine-lifecycle-support` 选项。 通过使用 RDS API 来调用 CreateDBCluster 并设置 `EngineLifecycleSupport` 参数。
RDS Proxy（RDS 代理）	选择 Create an RDS Proxy（创建 RDS 代理），以便为您的数据库集群创建代理。Amazon RDS 会自动为代理创建 IAM 角色和 Secrets Manager 密钥。 有关更多信息，请参阅 适用于 Aurora 的 Amazon RDS 代理。	创建数据库集群时不可用。
保留期	选择 Aurora 保留数据库的备份副本的时间长度（1 到 35 天）。可使用备份副本对数据库执行时间点还原 (PITR)，以还原到第二个时间点。	通过使用 AWS CLI 来运行 create-db-cluster 并设置 `--backup-retention-period` 选项。 通过使用 RDS API 来调用 CreateDBCluster 并设置 `BackupRetentionPeriod` 参数。

控制台设置	设置说明	CLI 选项和 RDS API 参数
Turn on DevOps Guru（开启 DevOps Guru）	选择 Turn on DevOps Guru（开启 DevOps Guru），以便为您的 Aurora 数据库开启 Amazon DevOps Guru。为让 DevOps Guru for RDS 提供对性能异常情况的详细分析，必须开启性能详情。有关更多信息，请参阅 设置适用于 RDS 的 DevOps Guru。	您可以从 RDS 控制台中开启 DevOps Guru for RDS，但不能使用 RDS API 或 CLI 开启它。有关更多信息，请参阅 Amazon DevOps Guru 用户指南。
开启性能详情	选择开启性能详情以开启 Amazon RDS 性能详情。有关更多信息，请参阅 在 Amazon Aurora 上使用性能详情监控数据库负载。	为 Aurora 集群中的每个数据库实例设置这些值。 通过使用 AWS CLI 来运行 `create-db-instance` 并设置 `--enable-performance-insights | --no-enable-performance-insights` 、`--performance-insights-kms-key-id` 和 `--performance-insights-retention-period` 选项。 通过使用 RDS API，调用 `CreateDBInstance` 并设置 `EnablePerformanceInsights` 、`PerformanceInsightsKMSKeyId` 和 `PerformanceInsightsRetentionPeriod` 参数。
Virtual Private Cloud (VPC)	选择要托管数据库集群的 VPC。选择创建新的 VPC 以让 Amazon RDS 为您创建 VPC。有关更多信息，请参阅 数据库集群先决条件。	对于 AWS CLI 和 API，您可以指定 VPC 安全组 ID。

控制台设置	设置说明	CLI 选项和 RDS API 参数
VPC security group (firewall) [VPC 安全组（防火墙）]	选择新建以让 Amazon RDS 为您创建 VPC 安全组。或者，选择选择现有，并指定一个或多个 VPC 安全组以保护对数据库集群的网络访问。 在 RDS 控制台中选择 Create new (新建) 时，将使用一个入站规则来创建新的安全组，该入站规则允许从浏览器中检测到的 IP 地址访问数据库实例。 有关更多信息，请参阅 数据库集群先决条件。	通过使用 AWS CLI 来运行 `create-db-cluster` 并设置 `--vpc-security-group-ids` 选项。 通过使用 RDS API 来调用 `CreateDBCluster` 并设置 `VpcSecurityGroupIds` 参数。

不适用于 Amazon Aurora 的数据库集群设置

AWS CLI 命令 `create-db-cluster` 和 RDS API 操作 `CreateDBCluster` 中的以下设置不适用于 Amazon Aurora 数据库集群。

> **Note**
>
> AWS Management Console 不显示 Aurora 数据库集群的以下设置。

AWS CLI 设置	RDS API 设置
`--allocated-storage`	`AllocatedStorage`
`--auto-minor-version-upgrade | --no-auto-minor-version-upgrade`	`AutoMinorVersionUpgrade`
`--db-cluster-instance-class`	`DBClusterInstanceClass`

AWS CLI 设置	RDS API 设置
`--enable-performance-insights | --no-enable-performance-insights`	`EnablePerformanceInsights`
`--iops`	`Iops`
`--monitoring-interval`	`MonitoringInterval`
`--monitoring-role-arn`	`MonitoringRoleArn`
`--option-group-name`	`OptionGroupName`
`--performance-insights-kms-key-id`	`PerformanceInsightsKMSKeyId`
`--performance-insights-retention-period`	`PerformanceInsightsRetentionPeriod`
`--publicly-accessible | --no-publicly-accessible`	`PubliclyAccessible`

不适用于 Amazon Aurora 数据库实例的设置

AWS CLI 命令 `create-db-instance` 和 RDS API 操作 `CreateDBInstance` 中的以下设置不适用于数据库实例 Amazon Aurora 数据库集群。

> **Note**
>
> AWS Management Console 不显示 Aurora 数据库实例的以下设置。

AWS CLI 设置	RDS API 设置
`--allocated-storage`	`AllocatedStorage`
`--availability-zone`	`AvailabilityZone`

AWS CLI 设置	RDS API 设置
--backup-retention-period	BackupRetentionPeriod
--backup-target	BackupTarget
--character-set-name	CharacterSetName
--character-set-name	CharacterSetName
--custom-iam-instance-profile	CustomIamInstanceProfile
--db-security-groups	DBSecurityGroups
--deletion-protection | --no-deletion-protection	DeletionProtection
--domain	Domain
--domain-iam-role-name	DomainIAMRoleName
--enable-cloudwatch-logs-exports	EnableCloudwatchLogsExports
--enable-customer-owned-ip | --no-enable-customer-owned-ip	EnableCustomerOwnedIp
--enable-iam-database-authentication | --no-enable-iam-database-authentication	EnableIAMDatabaseAuthentication
--engine-version	EngineVersion
--iops	Iops
--kms-key-id	KmsKeyId
--master-username	MasterUsername
--master-user-password	MasterUserPassword
--max-allocated-storage	MaxAllocatedStorage

AWS CLI 设置	RDS API 设置
--multi-az | --no-multi-az	MultiAZ
--nchar-character-set-name	NcharCharacterSetName
--network-type	NetworkType
--option-group-name	OptionGroupName
--preferred-backup-window	PreferredBackupWindow
--processor-features	ProcessorFeatures
--storage-encrypted | --no-storage-encrypted	StorageEncrypted
--storage-type	StorageType
--tde-credential-arn	TdeCredentialArn
--tde-credential-password	TdeCredentialPassword
--timezone	Timezone
--vpc-security-group-ids	VpcSecurityGroupIds

使用 AWS CloudFormation 创建 Amazon Aurora 资源

Amazon Aurora 与 AWS CloudFormation 集成，后者是一项服务，可帮助您对AWS资源进行建模和设置，这样您就可以花较少的时间来创建和管理资源与基础设施。您可以创建一个模板，描述所需的所有 AWS 资源（例如数据库集群和数据库集群参数组），而且 AWS CloudFormation 会为您预置和配置这些资源。

使用 AWS CloudFormation 时，可以重复使用您的模板来不断地重复设置您的 Aurora 资源。仅描述您的资源一次，然后在多个AWS账户和区域中反复配置相同的资源。

Aurora 和 AWS CloudFormation 模板

AWS CloudFormation 模板是 JSON 或 YAML 格式的文本文件。这些模板可描述您要在 AWS CloudFormation 堆栈中调配的资源。如果您不熟悉 JSON 或 YAML，可以在 AWS CloudFormation Designer 的帮助下开始使用 AWS CloudFormation 模板。有关更多信息，请参阅 AWS CloudFormation 用户指南中的什么是 AWS CloudFormation Designer？。

Aurora 支持在 AWS CloudFormation 中创建资源。有关更多信息（包括这些资源的 JSON 和 YAML 模板示例），请参阅 AWS CloudFormation 用户指南中的 RDS 资源类型参考。

了解有关 AWS CloudFormation 的更多信息

要了解有关 AWS CloudFormation 的更多信息，请参阅以下资源：

- AWS CloudFormation
- AWS CloudFormation 用户指南
- AWS CloudFormation API Reference
- AWS CloudFormation 命令行界面用户指南

连接到 Amazon Aurora 数据库集群

您可以使用用于连接到 MySQL 或 PostgreSQL 数据库的同一工具来连接到 Aurora 数据库集群。您可以使用连接到 MySQL 或 PostgreSQL 数据库实例的任何脚本、实用程序或应用程序指定连接字符串。您可以使用同一个公有密钥进行安全套接字层 (SSL) 连接。

在连接字符串中，通常使用与数据库集群关联的特殊端点的主机和端口信息。借助这些端点，无论集群中有多少个数据库实例，您都可以使用相同的连接参数。您还可以将 Aurora 数据库集群中特定数据库实例的主机和端口信息用于专业任务，例如故障排除。

> **Note**
>
> 对于 Aurora Serverless 数据库集群，您可以连接到数据库端点而非数据库实例。您可以在 AWS Management Console的 Connectivity & security (连接和安全) 选项卡上找到 Aurora Serverless 数据库集群的数据库端点。有关更多信息，请参阅“使用 Amazon Aurora Serverless v1”。

无论您使用哪种 Aurora 数据库引擎和特定工具来处理数据库集群或实例，端点都必须可访问。Aurora 数据库集群只能在基于 Amazon VPC 服务的虚拟私有云（VPC）中创建。这意味着您可以使用以下方法之一从 VPC 内部或 VPC 外部访问端点。

- 从 VPC 内部访问 Aurora 数据库集群 – 通过 VPC 访问 Aurora 数据库集群。为实现此目的，您可以编辑 VPC 的安全组上的入站规则以允许访问特定 Aurora 数据库集群。要了解更多信息，包括如何为不同的 Aurora 数据库集群场景配置 VPC，请参阅 Amazon Virtual Private Cloud VPC 和 Amazon Aurora。
- 从 VPC 外部访问 Aurora 数据库集群 – 要从 VPC 外部访问 Aurora 数据库集群，请使用数据库集群的公有端点地址。

有关更多信息，请参阅 排除 Aurora 连接故障。

目录

- 使用 AWS 驱动程序连接到 Aurora 数据库集群
- 连接到 Amazon Aurora MySQL 数据库集群
  - 适用于 Aurora MySQL 的连接实用程序
  - 使用 MySQL 实用程序连接到 Aurora MySQL
  - 使用 Amazon Web Services (AWS) JDBC 驱动程序连接到 Aurora MySQL

  - 使用 Amazon Web Services (AWS) Python 驱动程序连接到 Aurora MySQL
  - 使用适用于 MySQL 的 Amazon Web Services (AWS) ODBC 驱动程序连接到 Aurora MySQL
  - 使用 Amazon Web Services (AWS) Advanced NodeJS Wrapper 连接到 Aurora MySQL
  - 使用 SSL 连接到 Aurora MySQL
- 连接到 Amazon Aurora PostgreSQL 数据库集群
  - 适用于 Aurora PostgreSQL 的连接实用程序
  - 使用 Amazon Web Services (AWS) JDBC 驱动程序连接到 Aurora PostgreSQL
  - 使用 Amazon Web Services (AWS) Python 驱动程序连接到 Aurora PostgreSQL
  - 使用 Amazon Web Services (AWS) Advanced NodeJS Wrapper 连接到 Aurora PostgreSQL
- 排除 Aurora 连接故障

使用 AWS 驱动程序连接到 Aurora 数据库集群

借助 AWS 驱动程序套件，可显著缩短切换和失效转移时间，并支持使用 AWS Secrets Manager、AWS Identity and Access Management（IAM）和联合身份进行身份验证。AWS 驱动程序依靠监控数据库集群状态和了解集群拓扑，来确定新的写入器。这种方法将切换和失效转移时间缩短到几秒钟，而开源驱动程序的切换和失效转移时间则为几十秒。

下表列出了每个驱动程序支持的功能。随着新服务功能的推出，使用 AWS 驱动程序套件可为这些服务功能提供内置支持。

功能	AWS JDBC 驱动程序	AWS Python 驱动程序	适用于 MySQL 的 AWS ODBC 驱动程序	AWS Advanced NodeJS Wrapper
失效转移支持	是	是	是	是
增强了失效转移监控功能	是	是	是	是
读/写分离	是	是	否	是
Aurora Connection Tracker	是	是	否	是

功能	AWS JDBC 驱动程序	AWS Python 驱动程序	适用于 MySQL 的 AWS ODBC 驱动程序	AWS Advanced NodeJS Wrapper
驱动程序元数据连接	是	不适用	不适用	不适用
遥测	是	是	否	是
Secrets Manager	是	是	是	是
IAM 身份验证	是	是	是	是
联合身份验证（AD FS）	是	是	否	是
联合身份（Okta）	是	是	是	是
Aurora PostgreSQL Limitless Database	是（仅限 Aurora PostgreSQL）	否	否	是（仅限 Aurora PostgreSQL）

有关 AWS 驱动程序的更多信息，请参阅 Aurora MySQL 或 Aurora PostgreSQL 数据库集群的相应语言驱动程序。

连接到 Amazon Aurora MySQL 数据库集群

要对 Aurora 数据库集群进行身份验证，您可以使用 MySQL 用户名和密码身份验证或 AWS Identity and Access Management（IAM）数据库身份验证。有关使用 MySQL 用户名和密码身份验证的更多信息，请参阅 MySQL 文档中的访问控制和账户管理。有关使用 IAM 数据库身份验证的更多信息，请参阅 IAM 数据库身份验证。

在连接到与 MySQL 8.0 兼容的 Amazon Aurora 数据库集群时，您可以运行与 MySQL 8.0 版兼容的 SQL 命令。最低兼容版本是 MySQL 8.0.23。有关 MySQL 8.0 SQL 语法的更多信息，请参阅 MySQL 8.0 参考手册。有关适用于 Aurora MySQL 版本 3 的限制的信息，请参阅 比较 Aurora MySQL 版本 3 和 MySQL 8.0 社群版。

在连接到与 MySQL 5.7 兼容的 Amazon Aurora 数据库集群时，您可以运行与 MySQL 5.7 版兼容的 SQL 命令。有关 MySQL 5.7 SQL 语法的更多信息，请参阅 MySQL 5.7 参考手册。有关适用于 Aurora MySQL 5.7 的限制的信息，请参阅与 MySQL 5.7 兼容的 Aurora MySQL 版本 2。

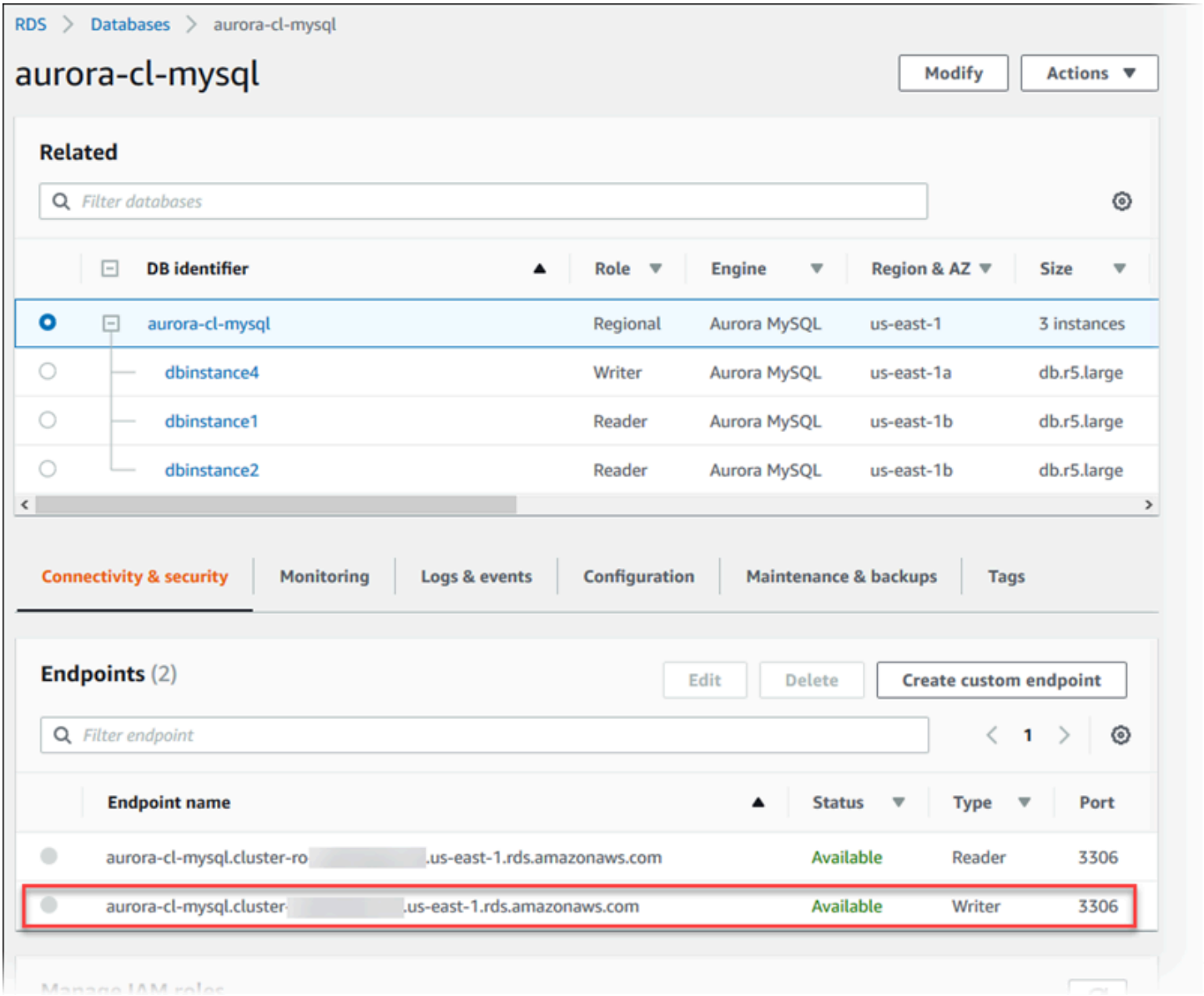
> **Note**
>
> 有关连接到 Amazon Aurora MySQL 数据库集群的实用详细指南，您可以参阅 Aurora 连接管理手册。

在数据库集群的详细信息视图中，您可以找到集群端点，可在 MySQL 连接字符串中使用此端点。该端点由数据库集群的域名和端口组成。例如，如果端点值为 `mycluster.cluster-123456789012.us-east-1.rds.amazonaws.com:3306`，则需要在 MySQL 连接字符串中指定以下值：

- 对于主机或主机名，请指定 `mycluster.cluster-123456789012.us-east-1.rds.amazonaws.com`
- 对于端口，请指定 `3306` 或在创建数据库集群时使用的端口值

集群端点将您连接到数据库集群的主实例。可使用集群端点执行读取和写入操作。数据库集群还可以具有最多 15 个 Aurora 副本，这些副本支持对数据库集群中的数据进行只读访问。主实例和每个 Aurora 副本具有唯一的端点，该端点独立于集群端点，允许您直接连接到集群中的特定数据库实例。集群端点始终指向主实例。如果主实例发生故障并被替换，则集群端点将指向新的主实例。

要查看集群端点（写入器端点），请在 Amazon RDS 控制台中选择 Databases (数据库)，然后选择数据库集群的名称以显示数据库集群详细信息。



主题

- 适用于 Aurora MySQL 的连接实用程序
- 使用 MySQL 实用程序连接到 Aurora MySQL
- 使用 Amazon Web Services (AWS) JDBC 驱动程序连接到 Aurora MySQL
- 使用 Amazon Web Services (AWS) Python 驱动程序连接到 Aurora MySQL
- 使用适用于 MySQL 的 Amazon Web Services (AWS) ODBC 驱动程序连接到 Aurora MySQL
- 使用 Amazon Web Services (AWS) Advanced NodeJS Wrapper 连接到 Aurora MySQL
- 使用 SSL 连接到 Aurora MySQL

适用于 Aurora MySQL 的连接实用程序

以下是您可以使用的一些连接实用程序：

- 命令行 – 您可以使用 MySQL 命令行实用程序等工具连接到 Amazon Aurora 数据库集群。有关使用 MySQL 实用程序的更多信息，请参阅 MySQL 文档中的 mysql — MySQL 命令行客户端。
- GUI – 您可以使用 MySQL Workbench 实用程序通过 UI 接口进行连接。有关更多信息，请参阅下载 MySQL Workbench 页。
- AWS 驱动程序：
  - 使用 Amazon Web Services (AWS) JDBC 驱动程序连接到 Aurora MySQL
  - 使用 Amazon Web Services (AWS) Python 驱动程序连接到 Aurora MySQL
  - 使用适用于 MySQL 的 Amazon Web Services (AWS) ODBC 驱动程序连接到 Aurora MySQL
  - 使用 Amazon Web Services (AWS) Advanced NodeJS Wrapper 连接到 Aurora MySQL

使用 MySQL 实用程序连接到 Aurora MySQL

使用以下流程。假设您在 VPC 的私有子网中配置了数据库集群。您可以使用根据教程：创建 Web 服务器和 Amazon Aurora 数据库集群中的教程配置的 Amazon EC2 实例进行连接。

> **Note**
>
> 在本教程中，此过程不要求安装 Web 服务器，但要求安装 MariaDB 10.5。

使用 MySQL 实用程序连接到数据库集群

1. 登录到您要用于连接到数据库集群的 EC2 实例。

   您应该可以看到类似于如下所示的输出内容。

```
Last login: Thu Jun 23 13:32:52 2022 from xxx.xxx.xxx.xxx

       __|  __|_  )
       _|  (     /   Amazon Linux 2 AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-2/
[ec2-user@ip-10-0-xxx.xxx ~]$
```

2. 在命令提示符处键入以下命令，以连接到数据库集群的主数据库实例。

   将参数 -h 替换为主实例的端点 DNS 名称。将参数 -u 替换为数据库用户账户的用户 ID。

   ```
   mysql -h primary-instance-endpoint.AWS_account.AWS_Region.rds.amazonaws.com -P 3306
    -u database_user -p
   ```

   例如：

   ```
   mysql -h my-aurora-cluster-instance.c1xy5example.123456789012.eu-
   central-1.rds.amazonaws.com -P 3306 -u admin -p
   ```

3. 输入数据库用户的密码。

   您应该可以看到类似于如下所示的输出内容。

   ```
   Welcome to the MariaDB monitor.  Commands end with ; or \g.
   Your MySQL connection id is 1770
   Server version: 8.0.23 Source distribution

   Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

   Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

   MySQL [(none)]>
   ```

4. 输入 SQL 命令。

使用 Amazon Web Services (AWS) JDBC 驱动程序连接到 Aurora MySQL

Amazon Web Services (AWS) JDBC 驱动程序已设计为高级 JDBC 包装器。此包装器补充并扩展了现有 JDBC 驱动程序的功能，来协助应用程序利用集群数据库（例如 Aurora MySQL）的功能。该驱动程序与社区 MySQL Connector/J 驱动程序和社区 MariaDB Connector/J 驱动程序兼容。

要安装 AWS JDBC 驱动程序，请附加 AWS JDBC 驱动程序.jar 文件（位于应用程序 CLASSPATH 中），并保留对相应社区驱动程序的引用。按如下方式更新相应的连接 URL 前缀：

- `jdbc:mysql://` 到 `jdbc:aws-wrapper:mysql://`
- `jdbc:mariadb://` 到 `jdbc:aws-wrapper:mariadb://`

有关 AWS JDBC 驱动程序的更多信息及其完整使用说明，请参阅 Amazon Web Services (AWS) JDBC Driver GitHub 存储库。

> **Note**
>
> 版本 3.0.3 的 MariaDB Connector/J 实用程序放弃了对 Aurora 数据库集群的支持，因此我们强烈建议改用 AWS JDBC 驱动程序。

使用 Amazon Web Services (AWS) Python 驱动程序连接到 Aurora MySQL

Amazon Web Services（AWS）驱动程序设计为高级 Python 包装器。这款包装器是对开源 Psycopg 驱动程序的各项功能的补充和扩展。AWS Python 驱动程序支持 Python 3.8 及更高版本。您可以使用 `pip` 命令和 `psycopg` 开源软件包安装 `aws-advanced-python-wrapper` 程序包。

有关 AWS Python 驱动程序的更多信息及其完整使用说明，请参阅 Amazon Web Services (AWS) Python 驱动程序 GitHub 存储库。

使用适用于 MySQL 的 Amazon Web Services (AWS) ODBC 驱动程序连接到 Aurora MySQL

适用于 MySQL 的 AWS ODBC 驱动程序是专为实现 Aurora MySQL 的高可用性而设计的客户端驱动程序。该驱动程序可以与 MySQL 连接器/ODBC 驱动程序并存，并且与相同的工作流程兼容。

有关适用于 MySQL 的 AWS ODBC 驱动程序的更多信息及其完整安装和使用说明，请参阅 Amazon Web Services (AWS) ODBC Driver for MySQL GitHub 存储库。

使用 Amazon Web Services (AWS) Advanced NodeJS Wrapper 连接到 Aurora MySQL

AWS Advanced NodeJS Wrapper 是对现有 NodeJS 驱动程序功能的补充和扩展。它可以帮助应用程序利用集群数据库（如 Aurora MySQL）的功能。

有关 AWS Advanced NodeJS Wrapper 的更多信息及其完整使用说明，请参阅 Amazon Web Services (AWS) Advanced NodeJS Wrapper GitHub 存储库。

使用 SSL 连接到 Aurora MySQL

您可以在连接到 Aurora MySQL 数据库实例时使用 SSL 加密功能。有关信息，请参阅与 Aurora MySQL 数据库集群的 TLS 连接。

要使用 SSL 进行连接，请使用以下过程中所述的 MySQL 实用程序。要使用 IAM 数据库身份验证，您必须使用 SSL 连接。有关信息，请参阅 IAM 数据库身份验证。

> **Note**
>
> 要使用 SSL 连接到集群端点，您的客户端连接实用程序必须支持主题替代名称 (SAN)。如果您的客户端连接实用程序不支持 SAN，则可以直接连接到 Aurora 数据库集群中的实例。有关 Aurora 端点的更多信息，请参阅Amazon Aurora 端点连接。

使用 MySQL 实用程序通过 SSL 连接到加密的数据库集群

1. 下载 Amazon RDS 签名证书的公有密钥。

   有关下载证书的信息，请参阅使用 SSL/TLS 加密与数据库集群的连接。

2. 在命令提示符处键入以下命令，以便使用 MySQL 实用程序连接到带 SSL 的数据库集群的主实例。将参数 -h 替换为主实例的端点 DNS 名称。将参数 -u 替换为数据库用户账户的用户 ID。将参数 --ssl-ca 替换为相应的 SSL 证书文件名。根据系统提示键入主用户密码。

   ```
   mysql -h mycluster-primary.123456789012.us-east-1.rds.amazonaws.com -u admin_user -p --ssl-ca=[full path]global-bundle.pem --ssl-verify-server-cert
   ```

您应该可以看到类似于如下所示的输出内容。

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 350
Server version: 8.0.26-log MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

有关构建 RDS for MySQL 连接字符串和查找 SSL 连接的公有密钥的一般说明，请参阅连接到运行 MySQL 数据库引擎的数据库实例。

连接到 Amazon Aurora PostgreSQL 数据库集群

您可以使用用于连接到 PostgreSQL 数据库的同一工具来连接到 Amazon Aurora PostgreSQL 数据库集群中的数据库实例。在连接过程中，您使用相同的公有密钥进行安全套接字层 (SSL) 连接。在连接到 PostgreSQL 数据库实例的任何脚本、实用程序或应用程序的连接字符串中，您可以使用 Aurora PostgreSQL 数据库集群中的主实例或 Aurora 副本的端点和端口信息。在连接字符串中，请指定主实例或 Aurora 副本端点中的 DNS 地址以作为主机参数。指定该端点中的端口号以作为端口参数。

在连接到 Amazon Aurora PostgreSQL 数据库集群中的数据库实例时，您可以运行与 PostgreSQL 兼容的任何 SQL 命令。

您可以在 Aurora PostgreSQL 数据库集群详细信息视图中找到集群端点名称、状态、类型和端口号。请在您的 PostgreSQL 连接字符串中使用端点和端口号。例如，如果端点值为 `mycluster.cluster-123456789012.us-east-1.rds.amazonaws.com`，则需要在 PostgreSQL 连接字符串中指定以下值：

- 对于主机或主机名，请指定 `mycluster.cluster-123456789012.us-east-1.rds.amazonaws.com`
- 对于端口，请指定 `5432` 或在创建数据库集群时使用的端口值

集群端点将您连接到数据库集群的主实例。可使用集群端点执行读取和写入操作。数据库集群还可以具有最多 15 个 Aurora 副本，这些副本支持对数据库集群中的数据进行只读访问。Aurora 集群中的每个数据库实例（即，主实例和每个 Aurora 副本）具有唯一的端点，该端点独立于集群端点。这个唯一的端点允许您与集群中特定的数据库实例直接连接。集群端点始终指向主实例。如果主实例发生故障并替换该实例，集群端点将指向新的主实例。

要查看集群端点（写入器端点），请在 Amazon RDS 控制台中选择 Databases (数据库)，然后选择数据库集群的名称以显示数据库集群详细信息。

适用于 Aurora PostgreSQL 的连接实用程序

以下是您可以使用的一些连接实用程序：

- 命令行 – 您可以使用 psql、PostgreSQL 交互式终端等工具连接到 Aurora PostgreSQL 数据库集群。有关使用 PostgreSQL 交互式终端的更多信息，请参阅 PostgreSQL 文档中的 psql。
- GUI – 您可以使用 pgAdmin 实用程序通过 UI 界面连接到 Aurora PostgreSQL 数据库集群。有关更多信息，请参阅 pgAdmin 网站中的下载页面。
- AWS 驱动程序：

- 使用 Amazon Web Services (AWS) JDBC 驱动程序连接到 Aurora PostgreSQL
- 使用 Amazon Web Services (AWS) Python 驱动程序连接到 Aurora PostgreSQL
- 使用 Amazon Web Services (AWS) Advanced NodeJS Wrapper 连接到 Aurora PostgreSQL

使用 Amazon Web Services (AWS) JDBC 驱动程序连接到 Aurora PostgreSQL

Amazon Web Services（AWS）JDBC 驱动程序已重新设计为高级 JDBC 包装器。此包装器补充并扩展了现有 JDBC 驱动程序的功能，以帮助应用程序充分利用集群数据库（如 Aurora PostgreSQL）的功能。该驱动程序与社区 pgJDBC 驱动程序兼容。

要安装 AWS JDBC 驱动程序，请附加 AWS JDBC 驱动程序.jar 文件（位于 `CLASSPATH` 应用程序中），并保留对 pgJDBC 社区驱动程序的引用。将连接 URL 前缀从 `jdbc:postgresql://` 更新为 `jdbc:aws-wrapper:postgresql://`。

有关 AWS JDBC 驱动程序的更多信息及其完整使用说明，请参阅 Amazon Web Services (AWS) JDBC Driver GitHub 存储库。

使用 Amazon Web Services (AWS) Python 驱动程序连接到 Aurora PostgreSQL

Amazon Web Services (AWS) Python 驱动程序设计为高级 Python 包装器。这款包装器是对开源 Psycopg 驱动程序的各项功能的补充和扩展。AWS Python 驱动程序支持 Python 3.8 及更高版本。您可以使用 `pip` 命令和 `psycopg` 开源软件包安装 `aws-advanced-python-wrapper` 程序包。

有关 AWS Python 驱动程序的更多信息及其完整使用说明，请参阅 Amazon Web Services (AWS) Python 驱动程序 GitHub 存储库。

使用 Amazon Web Services (AWS) Advanced NodeJS Wrapper 连接到 Aurora PostgreSQL

AWS Advanced NodeJS Wrapper 是对现有 NodeJS 驱动程序功能的补充和扩展。它可以帮助应用程序利用集群数据库（如 Aurora PostgreSQL）的功能。

有关 AWS Advanced NodeJS Wrapper 的更多信息及其完整使用说明，请参阅 Amazon Web Services (AWS) Advanced NodeJS Wrapper GitHub 存储库。

排除 Aurora 连接故障

导致新 Aurora 数据库集群连接故障的常见原因包括：

- VPC 中的安全组不允许访问 – 您的 VPC 需要通过正确地配置 VPC 中的安全组来允许来自您的设备或 Amazon EC2 实例的连接。要解决问题，请修改 VPC 的安全组入站规则以允许连接。有关示例，请参阅教程：创建 VPC 以用于数据库集群（仅限 IPv4）。
- 防火墙规则阻止的端口 – 检查为 Aurora 数据库集群配置的端口值。如果防火墙规则阻止该端口，则可以使用不同的端口重新创建实例。
- 不完整或不正确的 IAM 配置 – 如果您创建了 Aurora 数据库实例以使用基于 IAM– 的身份验证，请确保其配置正确。有关更多信息，请参阅“ IAM 数据库身份验证”。

有关 Aurora 数据库连接问题排查的更多信息，请参阅 无法连接到 Amazon RDS 数据库实例。

Amazon Aurora 的参数组

数据库参数指定数据库的配置方式。例如，数据库参数可以指定要分配给数据库的资源量（如内存）。

您可以通过将数据库实例和 Aurora 数据库集群与参数组进行关联来管理数据库配置。Aurora 使用默认设置定义参数组。您还可以使用自定义设置定义您自己的参数组。

主题

- 参数组概述
- Amazon Aurora 数据库集群的数据库集群参数组
- Amazon Aurora 数据库实例的数据库参数组
- 比较数据库参数组
- 指定数据库参数

参数组概述

数据库集群参数组就像是引擎配置值的容器，这些值应用于 Aurora 数据库集群中的每个数据库实例。例如，Aurora 共享存储模型需要 Aurora 集群中的每个数据库实例为参数（如 `innodb_file_per_table`）使用相同设置。因此，影响物理存储布局的参数是集群参数组的一部分。数据库集群参数组还包括所有实例级参数的原定设置值。

数据库参数组就像是引擎配置值的容器，这些值应用于一个或多个数据库实例。数据库参数组对于 Amazon RDS 和 Aurora 中的数据库实例均适用。这些配置设置适用于在 Aurora 集群内的数据库实例之间可能不同的属性，如内存缓冲区的大小。

主题

- 原定设置和自定义参数组
- 静态和动态数据库集群参数
- 静态和动态数据库实例参数
- 字符集参数
- 支持的参数和参数值

原定设置和自定义参数组

如果创建的数据库实例未指定数据库参数组，数据库实例将使用默认的数据库参数组。同样，如果您在创建 Aurora 数据库集群时未指定数据库集群参数组，数据库集群将使用默认数据库集群参数组。每个默认参数组包含数据库引擎默认值和 Amazon RDS 系统默认值，具体根据引擎、计算等级及实例的分配存储空间而定。

默认参数组的参数设置无法修改。您可以改而执行以下操作：

1. 创建新参数组。
2. 更改所需参数的设置。并非参数组中的所有数据库引擎参数都可供修改。
3. 修改数据库实例或数据库集群以关联新的参数组。

   有关修改数据库集群或数据库实例的信息，请参阅修改 Amazon Aurora 数据库集群。

> **Note**
>
> 如果您修改了数据库实例以使用自定义参数组，然后启动数据库实例，RDS 会在启动过程中自动重启数据库实例。对于启用了 AlwaysOn 或镜像选项的 RDS for SQL Server 多可用区实例，在启动过程结束后重启实例时，预计会进行失效转移。

仅在重启数据库实例后，RDS 才会在新关联的参数组中应用修改后的静态和动态参数。但是，如果在将数据库参数组与数据库实例关联之后修改数据库参数组中的动态参数，这些更改将立即得到应用，而无需重启。有关更改数据库集群参数组的信息，请参阅 修改 Amazon Aurora 数据库集群。

如果您更新了数据库参数组内的参数，更改将应用于与该参数组关联的所有数据库实例。同样，如果您更新了 Aurora 数据库参数组内的参数，更改将应用于与该数据库集群参数组关联的所有 Aurora 数据库集群。

如果您不想从头开始创建参数组，您可以使用 AWS CLI copy-db-parameter-group 命令或 copy-db-cluster-parameter-group 命令复制现有的参数组。您可能会发现复制参数组在某些情况下很有用。例如，您可能希望在新的参数组中包含现有参数组的大多数自定义参数和值。

静态和动态数据库集群参数

数据库集群参数是静态的，也可以是动态的。它们在以下方面有所不同：

- 更改静态参数并保存数据库集群参数组时，参数更改将在手动重启关联数据库集群中的数据库实例后生效。当您使用 AWS Management Console 更改静态数据库集群参数值时，它始终将 pending-reboot 用于 ApplyMethod。
- 当您更改动态参数时，原定设置情况下，参数更改将立即生效，而无需重启。当您使用控制台时，它始终将 immediate 用于 ApplyMethod。要将参数更改推迟到重启关联数据库集群中的数据库实例之后，请使用 AWS CLI 或 RDS API。将 ApplyMethod 设置为 pending-reboot 以进行参数更改。

有关使用 AWS CLI 更改参数值的更多信息，请参阅 modify-db-cluster-parameter-group。有关使用 RDS API 更改参数值的更多信息，请参阅 ModifyDBClusterParameterGroup。

如果您更改与数据库集群关联的数据库集群参数组，则重启数据库集群中的数据库实例。重启会将更改应用于数据库集群中的所有数据库实例。要确定是否必须重启数据库集群的数据库实例才能应用更改，请运行以下 AWS CLI 命令。

```
aws rds describe-db-clusters --db-cluster-identifier db_cluster_identifier
```

检查 DBClusterParameterGroupStatus 输出中主数据库实例的值。如果值为 pending-reboot，则重启数据库集群的数据库实例。

静态和动态数据库实例参数

数据库实例参数是静态的，也可以是动态的。它们在以下方面有所不同：

- 更改静态参数并保存数据库参数组时，参数更改将在手动重启关联的数据库实例后生效。对于静态参数，控制台始终将 pending-reboot 用于 ApplyMethod。
- 当您更改动态参数时，原定设置情况下，参数更改将立即生效，而无需重启。当您使用 AWS Management Console 更改数据库实例参数值时，对于动态参数，它始终使用 immediate 作为 ApplyMethod。要将参数更改推迟到重启关联的数据库实例之后，请使用 AWS CLI 或 RDS API。将 ApplyMethod 设置为 pending-reboot 以进行参数更改。

有关使用 AWS CLI 更改参数值的更多信息，请参阅 modify-db-parameter-group。有关使用 RDS API 更改参数值的更多信息，请参阅 ModifyDBParameterGroup。

如果数据库实例未使用对其关联的数据库参数组所做的最新更改，控制台将针对数据库参数组显示 pending-reboot（待重启）状态。此状态不会在下一个维护时段期间导致自动重启。要向该数据库实例应用最新的参数更改，请手动重启数据库实例。

字符集参数

在创建数据库集群之前，在参数组中设置与数据库的字符集或排序规则相关的任何参数。在其中创建数据库之前也要这样做。这样，您可确保原定设置数据库以及新数据库使用您指定的字符集和排序规则值。如果您更改字符集或排序规则参数，则参数更改不会应用于现有数据库。

对于某些数据库引擎，您可使用 ALTER DATABASE 命令更改现有数据库的字符集或排序规则值，例如：

```
ALTER DATABASE database_name CHARACTER SET character_set_name COLLATE collation;
```

有关更改数据库字符集或排序规则值的更多信息，请查阅数据库引擎文档。

支持的参数和参数值

要确定数据库引擎支持的参数，请查看数据库实例或数据库集群使用的数据库参数组和数据库集群参数组中的参数。有关更多信息，请参阅在 Amazon Aurora 中查看数据库参数组的参数值和在 Amazon Aurora 中查看数据库集群参数组的参数值。

在许可情况下，您可以使用表达式、公式和函数指定整数和布尔参数值。函数可以包含数学对数表达式。但是，并非所有参数都支持对参数值使用表达式、公式和函数。有关更多信息，请参阅 指定数据库参数。

对于 Aurora 全局数据库，您可以为各个 Aurora 集群指定不同的配置设置。请确保设置足够相似，以便您在将辅助集群提升到主集群时生成一致的行为。例如，对于跨 Aurora 全局数据库的所有集群的时区和字符集使用相同设置。

在参数组内设置参数不恰当可能会产生意外的不利影响，包括性能降低和系统不稳定。修改数据库参数时应始终保持谨慎，且在修改参数组前备份数据。将参数组更改应用于生产数据库实例或数据库集群前，请在测试数据库实例或数据库集群上尝试进行这些参数组设置更改。

Amazon Aurora 数据库集群的数据库集群参数组

Amazon Aurora 数据库集群使用数据库集群参数组。以下各节介绍配置和管理数据库集群参数组。

主题

- Amazon Aurora 数据库集群和数据库实例参数
- 在 Amazon Aurora 中创建数据库集群参数组
- 在 Amazon Aurora 中将数据库集群参数组与数据库集群关联

- 在 Amazon Aurora 中修改数据库集群参数组中的参数
- 在 Amazon Aurora 中重置数据库集群参数组中的参数
- 在 Amazon Aurora 中复制数据库集群参数组
- 在 Amazon Aurora 中列出数据库集群参数组
- 在 Amazon Aurora 中查看数据库集群参数组的参数值
- 在 Amazon Aurora 中删除数据库集群参数组

Amazon Aurora 数据库集群和数据库实例参数

Aurora 使用二级的配置设置系统：

- 数据库集群参数组中的参数适用于数据库集群中的每个数据库实例。您的数据存储在 Aurora 共享存储子系统中。因此，与表数据的物理布局相关的所有参数对于 Aurora 集群中的所有数据库实例都必须是相同的。同样，由于 Aurora 数据库实例通过复制连接，复制设置的所有参数在整个 Aurora 集群中也必须是一致的。
- 数据库参数组中的参数适用于 Aurora 数据库集群中的单个数据库实例。这些参数与您可以在同一个 Aurora 集群中的各个数据库实例中更改的各个方面（如内存使用率）相关。例如，集群通常包含具有不同 AWS 实例类的数据库实例。

每个 Aurora 集群与一个数据库集群参数组关联。此参数组为相应数据库引擎的每个配置值分配默认值。集群参数组还包括集群级和实例级参数的原定设置。预置或 Aurora Serverless v2 集群中的每个数据库实例都继承来自该数据库集群参数组的设置。

每个数据库实例也都与一个数据库参数组关联。数据库参数组中的值可以覆盖集群参数组中的默认值。例如，如果群集中的一个实例遇到问题，则可以为该实例分配一个自定义数据库参数组。对于与调试或性能优化相关的参数，该自定义参数组可能具有特定设置。

创建集群或新数据库实例时，Aurora 会根据指定的数据库引擎和版本分配默认参数组。您可以改为指定自定义参数组。您可以自行创建这些参数组，并可以编辑参数值。您可以在创建时指定这些自定义参数组。您还可以稍后修改数据库集群或实例以使用自定义参数组。

对于预置和 Aurora Serverless v2 实例，您在数据库集群参数组中修改的任意配置值将覆盖数据库参数组中的默认值。如果您在数据库参数组中编辑对应的值，则这些值将覆盖数据库集群参数组中的设置。

您修改的任何数据库参数设置均优先于数据库集群参数组值，即使您将配置参数更改回其默认值。您可以使用 describe-db-parameters AWS CLI 命令或 DescribeDBParameters RDS API 操作查看哪些参数被覆盖。如果您修改了该参数，`Source` 字段将包含值 `user`。若要重置一个或多个参数，以使数据库

集群参数组的值优先，请使用 reset-db-parameter-group AWS CLI 命令或 ResetDBParameterGroup RDS API 操作。

Aurora 中可供您使用的数据库集群和数据库实例参数因数据库引擎兼容性而异。

数据库引擎	参数
Aurora MySQL	请参阅 Aurora MySQL 配置参数。 对于 Aurora Serverless 集群，请参阅 使用 Aurora Serverless v2 的参数组和 Aurora Serverless v1 的参数组中的其他详细信息。
Aurora PostgreSQL	请参阅Amazon Aurora PostgreSQL 参数。 对于 Aurora Serverless 集群，请参阅 使用 Aurora Serverless v2 的参数组和 Aurora Serverless v1 的参数组中的其他详细信息。

> **Note**
>
> Aurora Serverless v1 集群只有数据库集群参数组，而没有数据库参数组。对于 Aurora Serverless v2 集群，您对数据库集群参数组中的自定义参数进行所有更改。
> Aurora Serverless v2 使用数据库集群参数组和数据库参数组。使用 Aurora Serverless v2，您几乎可以修改所有配置参数。Aurora Serverless v2 覆盖某些与容量相关的配置参数的设置，以便在 Aurora Serverless v2 实例缩减的情况下不会中断工作负载。
> 要了解有关 Aurora Serverless 配置设置以及您可以修改哪些设置的更多信息，请参阅 使用 Aurora Serverless v2 的参数组和 Aurora Serverless v1 的参数组。

在 Amazon Aurora 中创建数据库集群参数组

您可以使用 AWS Management Console、AWS CLI 或 RDS API 创建新数据库集群参数组。

创建数据库集群参数组之后，至少等待 5 分钟，然后创建使用该数据库集群参数组的数据库集群。这样，Amazon RDS 就可以在新数据库集群使用参数组之前完全创建此参数组。您可以使用 Amazon RDS 控制台的 Parameter groups（参数组）页面，或者使用 describe-db-cluster-parameters 命令，以验证是否创建了数据库集群参数组。

以下限制适用于数据库集群参数组名称：

- 名称必须为 1 到 255 个字母、数字或连字符。

  原定设置参数组名称可以包含句点，例如 `default.aurora-mysql5.7`。但是，自定义参数组名称不能包含句点。
- 第一个字符必须是字母。
- 名称不能以连字符结束，也不能包含两个连续的连字符。

控制台

若要创建数据库集群参数组

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择参数组。
3. 选择创建参数组。
4. 对于参数组名称，输入新数据库集群参数组的名称。
5. 对于描述，输入新数据库集群参数组的描述。
6. 对于引擎类型，请选择数据库引擎。
7. 对于参数组系列，请选择一个数据库参数组系列。
8. 选择创建。

AWS CLI

要创建数据库集群参数组，请使用 AWS CLI `create-db-cluster-parameter-group` 命令。

以下示例为 Aurora MySQL 版本 5.7 创建名为 mydbclusterparametergroup 的数据库集群参数组，其说明为“我的新集群参数组”。

包括以下必需参数：

- `--db-cluster-parameter-group-name`
- `--db-parameter-group-family`
- `--description`

要列出所有可用的参数组系列，请使用以下命令：

```
aws rds describe-db-engine-versions --query "DBEngineVersions[].DBParameterGroupFamily"
```

> **Note**
>
> 输出包含重复项。

Example

对于 Linux、macOS 或 Unix：

```
aws rds create-db-cluster-parameter-group \
    --db-cluster-parameter-group-name mydbclusterparametergroup \
    --db-parameter-group-family aurora-mysql5.7 \
    --description "My new cluster parameter group"
```

对于 Windows：

```
aws rds create-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name mydbclusterparametergroup ^
    --db-parameter-group-family aurora-mysql5.7 ^
    --description "My new cluster parameter group"
```

此命令生成类似于下述信息的输出：

```
{
    "DBClusterParameterGroup": {
        "DBClusterParameterGroupName": "mydbclusterparametergroup",
        "DBParameterGroupFamily": "aurora-mysql5.7",
        "Description": "My new cluster parameter group",
        "DBClusterParameterGroupArn": "arn:aws:rds:us-east-1:123456789012:cluster-
pg:mydbclusterparametergroup"
    }
}
```

RDS API

要创建数据库集群参数组，请使用 RDS API CreateDBClusterParameterGroup 操作。

包括以下必需参数：

- `DBClusterParameterGroupName`
- `DBParameterGroupFamily`
- `Description`

在 Amazon Aurora 中将数据库集群参数组与数据库集群关联

您可以使用自定义设置创建自己的数据库集群参数组。之后可以使用 AWS Management Console、AWS CLI 或 RDS API 关联数据库集群参数组与数据库集群。在创建或修改数据库集群时可以执行此操作。

有关创建数据库集群参数组的信息，请参阅在 Amazon Aurora 中创建数据库集群参数组。有关创建数据库集群的信息，请参阅创建 Amazon Aurora 数据库集群。有关修改数据库集群的信息，请参阅修改 Amazon Aurora 数据库集群。

> **Note**
>
> 对于 Aurora PostgreSQL 15.2、14.7、13.10、12.14 和所有 11 版本，当您更改与数据库集群关联的数据库集群参数组时，需重启每个副本实例以应用这些更改。
> 要确定是否必须重启数据库集群的主数据库实例才能应用更改，请运行以下 AWS CLI 命令：`aws rds describe-db-clusters --db-cluster-identifier` *`db_cluster_identifier`*
> 检查 `DBClusterParameterGroupStatus` 输出中主数据库实例的值。如果值为 `pending-reboot`，则重启数据库集群的主数据库实例。

控制台

关联数据库集群参数组与数据库集群

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Databases (数据库)，然后选择您希望修改的数据库集群。
3. 选择修改。此时会显示修改数据库集群页面。
4. 更改数据库集群参数组设置。
5. 选择继续，查看修改摘要。

   无论修改计划设置如何，系统都会立即应用更改。

6. 在确认页面上，检查您的更改。如果更改正确无误，请选择修改集群以保存更改。

   或者，选择 Back (返回) 编辑您的更改，或者选择 Cancel (取消) 取消更改。

AWS CLI

要关联数据库集群参数组与数据库集群，请使用 AWS CLI `modify-db-cluster` 命令及以下选项：

- `--db-cluster-name`
- `--db-cluster-parameter-group-name`

以下示例关联了 `mydbclpg` 数据库参数组与 `mydbcluster` 数据库集群。

Example

对于 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --db-cluster-parameter-group-name mydbclpg
```

对于 Windows：

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --db-cluster-parameter-group-name mydbclpg
```

RDS API

要关联数据库集群参数组与数据库集群，请使用 RDS API `ModifyDBCluster` 操作及以下参数：

- `DBClusterIdentifier`
- `DBClusterParameterGroupName`

在 Amazon Aurora 中修改数据库集群参数组中的参数

您可以修改客户创建的数据库集群参数组中的参数值。您无法更改默认数据库集群参数组中的参数值。对客户创建的数据库集群参数组中的参数所做的更改将应用于与此数据库集群参数组关联的所有数据库集群。

控制台

修改数据库集群参数组

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择参数组。
3. 在此列表中，选择要修改的参数组。
4. 对于 Parameter group actions (参数组操作)，选择 Edit (编辑)。
5. 更改要修改的参数的值。您可使用对话框右上方的箭头键滚动参数。

   您无法更改默认参数组中的值。
6. 选择保存更改。
7. 重启集群中的主（写入器）数据库实例以对其应用更改。
8. 然后，重启读取器数据库实例以对其应用更改。

   如果您不重启数据库实例，则失效转移操作需要的时间可能比正常情况下更长。

AWS CLI

要修改数据库集群参数组，请使用带以下必需参数的 AWS CLI modify-db-cluster-parameter-group 命令：

- `--db-cluster-parameter-group-name`
- `--parameters`

以下示例修改了名为 mydbclusterparametergroup 的数据库集群参数组的 `server_audit_logging` 和 `server_audit_logs_upload` 值。

Example

对于 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster-parameter-group \
    --db-cluster-parameter-group-name mydbclusterparametergroup \
    --parameters
 "ParameterName=server_audit_logging,ParameterValue=1,ApplyMethod=immediate" \
```

```
 "ParameterName=server_audit_logs_upload,ParameterValue=1,ApplyMethod=immediate"
```

对于 Windows：

```
aws rds modify-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name mydbclusterparametergroup ^
    --parameters
 "ParameterName=server_audit_logging,ParameterValue=1,ApplyMethod=immediate" ^

 "ParameterName=server_audit_logs_upload,ParameterValue=1,ApplyMethod=immediate"
```

该命令产生类似下面的输出：

```
DBCLUSTERPARAMETERGROUP  mydbclusterparametergroup
```

RDS API

要修改数据库集群参数组，请使用带以下必需参数的 RDS API ModifyDBClusterParameterGroup 命令：

- DBClusterParameterGroupName
- Parameters

在 Amazon Aurora 中重置数据库集群参数组中的参数

您可以在客户创建的数据库集群参数组中将参数重置为其默认值。对客户创建的数据库集群参数组中的参数所做的更改将应用于与此数据库集群参数组关联的所有数据库集群。

> **Note**
>
> 在默认数据库集群参数组中，参数始终设置为默认值。

控制台

将数据库集群参数组中的参数重置为其默认值

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。

2. 在导航窗格中，选择参数组。
3. 在列表中，选择参数组。
4. 对于 Parameter group actions (参数组操作)，选择 Edit (编辑)。
5. 选择要重置为默认值的参数。您可使用对话框右上方的箭头键滚动参数。

   您无法重置默认参数组中的值。
6. 选择重置，然后通过选择重置参数进行确认。
7. 重启数据库集群中的主数据库实例，以将更改应用于数据库集群中的所有数据库实例。

AWS CLI

要将数据库集群参数组中的参数重置为其默认值，请使用带以下必要选项的 AWS CLI `reset-db-cluster-parameter-group` 命令：`--db-cluster-parameter-group-name`。

要重置数据库集群参数组中的所有参数，请指定 `--reset-all-parameters` 选项。要重置特定参数，请指定 `--parameters` 选项。

以下示例将名为 mydbparametergroup 的数据库参数组中的所有参数重置为其默认值。

Example

对于 Linux、macOS 或 Unix：

```
aws rds reset-db-cluster-parameter-group \
    --db-cluster-parameter-group-name mydbparametergroup \
    --reset-all-parameters
```

对于 Windows：

```
aws rds reset-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name mydbparametergroup ^
    --reset-all-parameters
```

以下示例展示了在名为 mydbclusterparametergroup 的数据库集群参数组中将 `server_audit_logging` 和 `server_audit_logs_upload` 重置为其默认值。

Example

对于 Linux、macOS 或 Unix：

```
aws rds reset-db-cluster-parameter-group \
    --db-cluster-parameter-group-name mydbclusterparametergroup \
    --parameters "ParameterName=server_audit_logging,ApplyMethod=immediate" \
                 "ParameterName=server_audit_logs_upload,ApplyMethod=immediate"
```

对于 Windows：

```
aws rds reset-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name mydbclusterparametergroup ^
    --parameters
 "ParameterName=server_audit_logging,ParameterValue=1,ApplyMethod=immediate" ^

 "ParameterName=server_audit_logs_upload,ParameterValue=1,ApplyMethod=immediate"
```

该命令产生类似下面的输出：

```
DBClusterParameterGroupName  mydbclusterparametergroup
```

RDS API

要将数据库集群参数组中的参数重置为其默认值，请使用带以下必需参数的 RDS API ResetDBClusterParameterGroup 命令：DBClusterParameterGroupName。

要重置数据库集群参数组中的所有参数，请将 ResetAllParameters 参数设置为 true。要重置特定参数，请指定 Parameters 参数。

在 Amazon Aurora 中复制数据库集群参数组

您可以复制您创建的自定义数据库集群参数组。当您已创建一个数据库集群参数组并且想在新的数据库集群参数组中包含该组中的大部分自定义参数和值时，复制参数组是一个方便的解决方案。您可通过使用 AWS CLI copy-db-cluster-parameter-group 命令或 RDS API CopyDBClusterParameterGroup 操作来复制数据库集群参数组。

复制数据库集群参数组之后，至少等待 5 分钟，然后创建使用该数据库集群参数组的数据库集群。这样，Amazon RDS 就可以在新数据库集群使用参数组之前完全复制此参数组。您可以使用 Amazon RDS 控制台的 Parameter groups（参数组）页面，或者使用 describe-db-cluster-parameters 命令，以验证是否创建了数据库集群参数组。

> **Note**
>
> 您无法复制默认参数组。不过，您可以创建基于默认参数组的新参数组。

您无法将数据库集群参数组复制到其他 AWS 账户或 AWS 区域。

控制台

复制数据库集群参数组

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择参数组。
3. 在列表中，选择要复制的自定义参数组。
4. 对于 Parameter group actions (参数组操作)，选择 Copy (复制)。
5. 在 New DB parameter group identifier (新数据库参数组标识符) 中，输入新参数组的名称。
6. 在 Description (描述) 中，输入新参数组的描述。
7. 选择 Copy (复制)。

AWS CLI

要复制数据库集群参数组，请使用 AWS CLI `copy-db-cluster-parameter-group` 命令及以下必要参数：

- `--source-db-cluster-parameter-group-identifier`
- `--target-db-cluster-parameter-group-identifier`
- `--target-db-cluster-parameter-group-description`

以下示例创建一个名为 `mygroup2` 的新数据库参集群数组，它是数据库集群参数组 `mygroup1` 的副本。

Example

对于 Linux、macOS 或 Unix：

```
aws rds copy-db-cluster-parameter-group \
    --source-db-cluster-parameter-group-identifier mygroup1 \
    --target-db-cluster-parameter-group-identifier mygroup2 \
    --target-db-cluster-parameter-group-description "DB parameter group 2"
```

对于 Windows：

```
aws rds copy-db-cluster-parameter-group ^
    --source-db-cluster-parameter-group-identifier mygroup1 ^
    --target-db-cluster-parameter-group-identifier mygroup2 ^
    --target-db-cluster-parameter-group-description "DB parameter group 2"
```

RDS API

要复制数据库集群参数组，请将 RDS API CopyDBClusterParameterGroup 操作与下列必需参数配合使用：

- SourceDBClusterParameterGroupIdentifier
- TargetDBClusterParameterGroupIdentifier
- TargetDBClusterParameterGroupDescription

在 Amazon Aurora 中列出数据库集群参数组

您可以列出为 AWS 账户创建的数据库集群参数组。

> **Note**
>
> 当您为特定数据库引擎和版本创建数据库集群时，将自动从默认参数模板创建默认参数组。这些默认参数组包含首选参数设置，并且无法修改。当您创建自定义参数组时，可以修改参数设置。

控制台

列出 AWS 账户的所有数据库集群参数组

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择参数组。

   数据库集群参数组出现在数据库集群参数组的类型列表中。

AWS CLI

要列出 AWS 账户的所有数据库集群参数组，请使用 AWS CLI [describe-db-cluster-parameter-groups](#) 命令。

Example

下例列出了 AWS 账户的所有可用数据库集群参数组。

```
aws rds describe-db-cluster-parameter-groups
```

以下示例描述了 mydbclusterparametergroup 参数组。

对于 Linux、macOS 或 Unix：

```
aws rds describe-db-cluster-parameter-groups \
    --db-cluster-parameter-group-name mydbclusterparametergroup
```

对于 Windows：

```
aws rds describe-db-cluster-parameter-groups ^
    --db-cluster-parameter-group-name mydbclusterparametergroup
```

此命令会返回类似以下内容的响应：

```
{
    "DBClusterParameterGroups": [
        {
            "DBClusterParameterGroupName": "mydbclusterparametergroup",
            "DBParameterGroupFamily": "aurora-mysql5.7",
            "Description": "My new cluster parameter group",
            "DBClusterParameterGroupArn": "arn:aws:rds:us-east-1:123456789012:cluster-
pg:mydbclusterparametergroup"
        }
    ]
}
```

RDS API

要列出 AWS 账户的所有数据库集群参数组，请使用 RDS API [DescribeDBClusterParameterGroups](#) 操作。

在 Amazon Aurora 中查看数据库集群参数组的参数值

您可获得数据库集群参数组内所有参数的列表及它们的值。

控制台

查看数据库集群参数组的参数值

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择参数组。

   数据库集群参数组出现在数据库集群参数组的类型列表中。
3. 选择数据库集群参数组的名称以查看其参数列表。

AWS CLI

要查看数据库集群参数组的参数值，请使用 AWS CLI `describe-db-cluster-parameters` 命令及以下必要参数。

- `--db-cluster-parameter-group-name`

Example

以下示例以 JSON 格式列出名为 mydbclusterparametergroup 的数据库集群参数组的参数和参数值。

此命令会返回类似以下内容的响应：

```
aws rds describe-db-cluster-parameters --db-cluster-parameter-group-
name mydbclusterparametergroup
```

```
{
    "Parameters": [
        {
            "ParameterName": "allow-suspicious-udfs",
            "Description": "Controls whether user-defined functions that have only an
 xxx symbol for the main function can be loaded",
            "Source": "engine-default",
            "ApplyType": "static",
```

```
            "DataType": "boolean",
            "AllowedValues": "0,1",
            "IsModifiable": false,
            "ApplyMethod": "pending-reboot",
            "SupportedEngineModes": [
                "provisioned"
            ]
        },
        {
            "ParameterName": "aurora_binlog_read_buffer_size",
            "ParameterValue": "5242880",
            "Description": "Read buffer size used by master dump thread when the switch
 aurora_binlog_use_large_read_buffer is ON.",
            "Source": "engine-default",
            "ApplyType": "dynamic",
            "DataType": "integer",
            "AllowedValues": "8192-536870912",
            "IsModifiable": true,
            "ApplyMethod": "pending-reboot",
            "SupportedEngineModes": [
                "provisioned"
            ]
        },

...
```

RDS API

要查看数据库集群参数组的参数值，请使用带下列所需参数的 RDS API [DescribeDBClusterParameters](#) 命令。

- `DBClusterParameterGroupName`

在某些情况下，不显示参数的允许值。这些始终是源为数据库引擎原定设置值的参数。

要查看这些参数的值，可以运行以下 SQL 语句：

- MySQL：

```
-- Show the value of a particular parameter
mysql$ SHOW VARIABLES LIKE '%parameter_name%';

-- Show the values of all parameters
```

```
mysql$ SHOW VARIABLES;
```

- PostgreSQL：

```
-- Show the value of a particular parameter
postgresql=> SHOW parameter_name;

-- Show the values of all parameters
postgresql=> SHOW ALL;
```

在 Amazon Aurora 中删除数据库集群参数组

您可以使用 AWS Management Console、AWS CLI 或 RDS API 删除数据库集群参数组。仅当数据库集群参数组未与数据库集群关联时，才能将其删除。

控制台

删除参数组

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择参数组。

   参数组显示在列表中。
3. 选择要删除的数据库集群参数组的名称。
4. 选择操作，然后选择删除。
5. 查看参数组名称，然后选择删除。

AWS CLI

要删除数据库集群参数组，请使用带以下必需参数的 AWS CLI `delete-db-cluster-parameter-group` 命令。

- `--db-parameter-group-name`

Example

以下示例删除了名为 mydbparametergroup 的数据库集群参数组。

```
aws rds delete-db-cluster-parameter-group --db-parameter-group-name mydbparametergroup
```

RDS API

要删除数据库集群参数组，请使用带以下必需参数的 RDS API `DeleteDBClusterParameterGroup` 命令。

- `DBParameterGroupName`

Amazon Aurora 数据库实例的数据库参数组

数据库实例使用数据库参数组。以下各节介绍配置和管理数据库实例参数组。

主题

- 在 Amazon Aurora 中创建数据库参数组
- 在 Amazon Aurora 中将数据库参数组与数据库实例关联
- 在 Amazon Aurora 中修改数据库参数组中的参数
- 在 Amazon Aurora 中将数据库参数组中的参数重置为其默认值
- 在 Amazon Aurora 中复制数据库参数组
- 在 Amazon Aurora 中列出数据库参数组
- 在 Amazon Aurora 中查看数据库参数组的参数值
- 在 Amazon Aurora 中删除数据库参数组

在 Amazon Aurora 中创建数据库参数组

您可以使用 AWS Management Console、AWS CLI 或 RDS API 创建新数据库参数组。

以下限制适用于数据库参数组名称：

- 名称必须为 1 到 255 个字母、数字或连字符。

  原定设置参数组名称可以包含句点，例如 `default.mysql8.0`。但是，自定义参数组名称不能包含句点。
- 第一个字符必须是字母。
- 名称不能以连字符结束，也不能包含两个连续的连字符。

控制台

创建数据库参数组

1. 登录AWS Management Console并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择参数组。
3. 选择创建参数组。
4. 对于参数组名称，请输入新数据库参数组的名称。
5. 对于描述，请输入对新数据库参数组的描述。
6. 对于引擎类型，请选择所需的数据库引擎。
7. 对于参数组系列，请选择一个数据库参数组系列。
8. 对于类型，如果适用，请选择数据库参数组。
9. 选择创建。

AWS CLI

要创建数据库参数组，请使用 AWS CLI `create-db-parameter-group` 命令。以下示例为 8.0 版 MySQL 创建名为 mydbparametergroup 的数据库参数组，其说明为“我的新参数组”。

包括以下必需参数：

- `--db-parameter-group-name`
- `--db-parameter-group-family`
- `--description`

要列出所有可用的参数组系列，请使用以下命令：

```
aws rds describe-db-engine-versions --query "DBEngineVersions[].DBParameterGroupFamily"
```

> **Note**
>
> 输出包含重复项。

Example

对于 Linux、macOS 或 Unix：

```
aws rds create-db-parameter-group \
    --db-parameter-group-name mydbparametergroup \
    --db-parameter-group-family aurora-mysql5.7 \
    --description "My new parameter group"
```

对于 Windows：

```
aws rds create-db-parameter-group ^
    --db-parameter-group-name mydbparametergroup ^
    --db-parameter-group-family aurora-mysql5.7 ^
    --description "My new parameter group"
```

此命令生成类似于下述信息的输出：

```
DBPARAMETERGROUP  mydbparametergroup  aurora-mysql5.7  My new parameter group
```

RDS API

要创建数据库参数组，请使用 RDS API `CreateDBParameterGroup` 操作。

包括以下必需参数：

- `DBParameterGroupName`
- `DBParameterGroupFamily`
- `Description`

在 Amazon Aurora 中将数据库参数组与数据库实例关联

您可以使用自定义设置创建自己的数据库参数组。之后可以使用 AWS Management Console、AWS CLI 或 RDS API 关联数据库参数组与数据库实例。在创建或修改数据库实例时可以执行此操作。

有关创建数据库参数组的信息，请参阅在 Amazon Aurora 中创建数据库参数组。有关修改数据库实例的信息，请参阅修改数据库集群中的数据库实例。

> **Note**
>
> 将新数据库参数组与数据库实例关联时，修改后的静态和动态参数仅在数据库实例重新启动后得到应用。但是，如果在将数据库参数组与数据库实例关联之后修改数据库参数组中的动态参数，这些更改将立即得到应用，而无需重启。

控制台

关联数据库参数组与数据库实例

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择数据库，然后选择要修改的数据库实例。
3. 选择修改。将显示修改数据库实例页面。
4. 更改数据库参数组设置。
5. 选择继续，查看修改摘要。
6. （可选）选择立即应用以立即应用更改。选择此选项在某些情况下可能导致中断。
7. 在确认页面上，检查您的更改。如果更改正确无误，请选择 Modify DB Instance（修改数据库实例）保存更改。

   也可以选择 Back (返回) 编辑您的更改，或选择 Cancel (取消) 取消更改。

AWS CLI

要关联数据库参数组与数据库实例，请使用 AWS CLI `modify-db-instance` 命令及以下选项：

- `--db-instance-identifier`
- `--db-parameter-group-name`

以下示例关联了 mydbpg 数据库参数组与 `database-1` 数据库实例。使用 `--apply-immediately` 可立即应用更改。使用 `--no-apply-immediately` 可在下一维护时段内应用更改。

Example

对于 Linux、macOS 或 Unix：

```
aws rds modify-db-instance \
    --db-instance-identifier database-1 \
    --db-parameter-group-name mydbpg \
    --apply-immediately
```

对于 Windows：

```
aws rds modify-db-instance ^
    --db-instance-identifier database-1 ^
    --db-parameter-group-name mydbpg ^
    --apply-immediately
```

RDS API

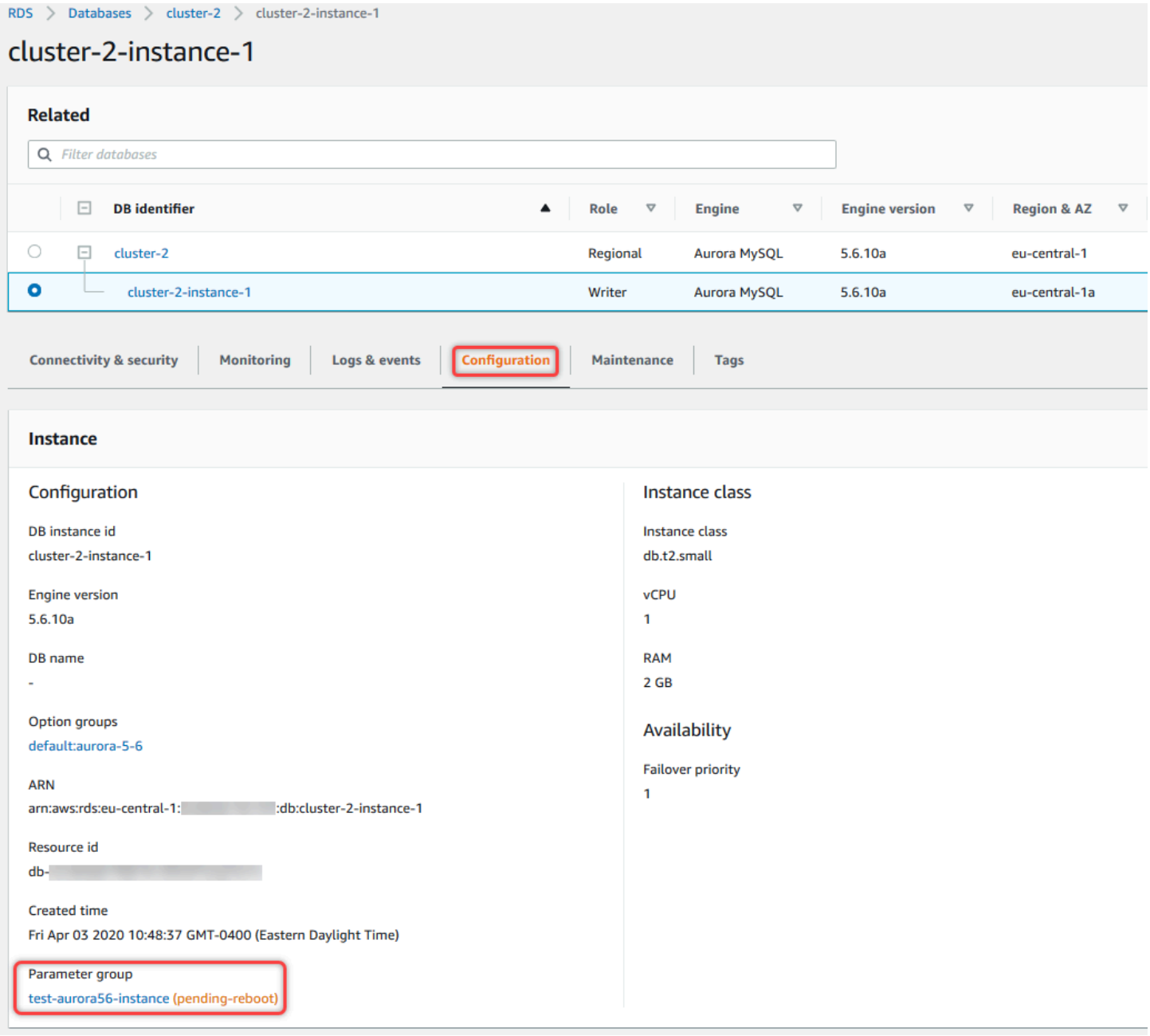
要关联数据库参数组与数据库实例，请使用 RDS API `ModifyDBInstance` 操作及以下参数：

- `DBInstanceName`
- `DBParameterGroupName`

在 Amazon Aurora 中修改数据库参数组中的参数

您可以修改客户创建的数据库参数组中的参数值，但不能更改默认数据库参数组中的参数值。对客户创建的数据库参数组中的参数所做的更改将应用于与此数据库参数组关联的所有数据库实例。

对某些参数的更改将立即应用于数据库实例，而无需重新启动。而对其他一些参数进行的更改，只有在重新启动数据库实例之后，才会应用。RDS 控制台在 Configuration (配置) 选项卡上显示与数据库实例关联的数据库参数组的状态。例如，假设数据库实例未使用对其关联的数据库参数组所做的最新更改。如果是这样，RDS 控制台将显示状态为 pending-reboot（待重启）的数据库参数组。要向该数据库实例应用最新的参数更改，请手动重启数据库实例。我们建议您在修改参数后，始终重启数据库实例。如果您不重启数据库实例，则失效转移操作需要的时间可能比正常情况下更长。



控制台

修改数据库参数组中的参数

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择参数组。
3. 在此列表中，选择要修改的参数组的名称。
4. 对于 Parameter group actions (参数组操作)，选择 Edit (编辑)。

5. 更改要修改的参数的值。您可使用对话框右上方的箭头键滚动参数。

   您无法更改默认参数组中的值。

6. 选择保存更改。

AWS CLI

要修改数据库参数组，请使用 AWS CLI modify-db-parameter-group 命令及以下必需选项：

- --db-parameter-group-name
- --parameters

以下示例修改了名为 mydbparametergroup 的数据库参数组的 max_connections 和 max_allowed_packet 值。

Example

对于 Linux、macOS 或 Unix：

```
aws rds modify-db-parameter-group \
    --db-parameter-group-name mydbparametergroup \
    --parameters
 "ParameterName=max_connections,ParameterValue=250,ApplyMethod=immediate" \

 "ParameterName=max_allowed_packet,ParameterValue=1024,ApplyMethod=immediate"
```

对于 Windows：

```
aws rds modify-db-parameter-group ^
    --db-parameter-group-name mydbparametergroup ^
    --parameters
 "ParameterName=max_connections,ParameterValue=250,ApplyMethod=immediate" ^

 "ParameterName=max_allowed_packet,ParameterValue=1024,ApplyMethod=immediate"
```

该命令产生类似下面的输出：

```
DBPARAMETERGROUP  mydbparametergroup
```

RDS API

要修改数据库参数组，请使用 RDS API `ModifyDBParameterGroup` 操作及以下必要参数：

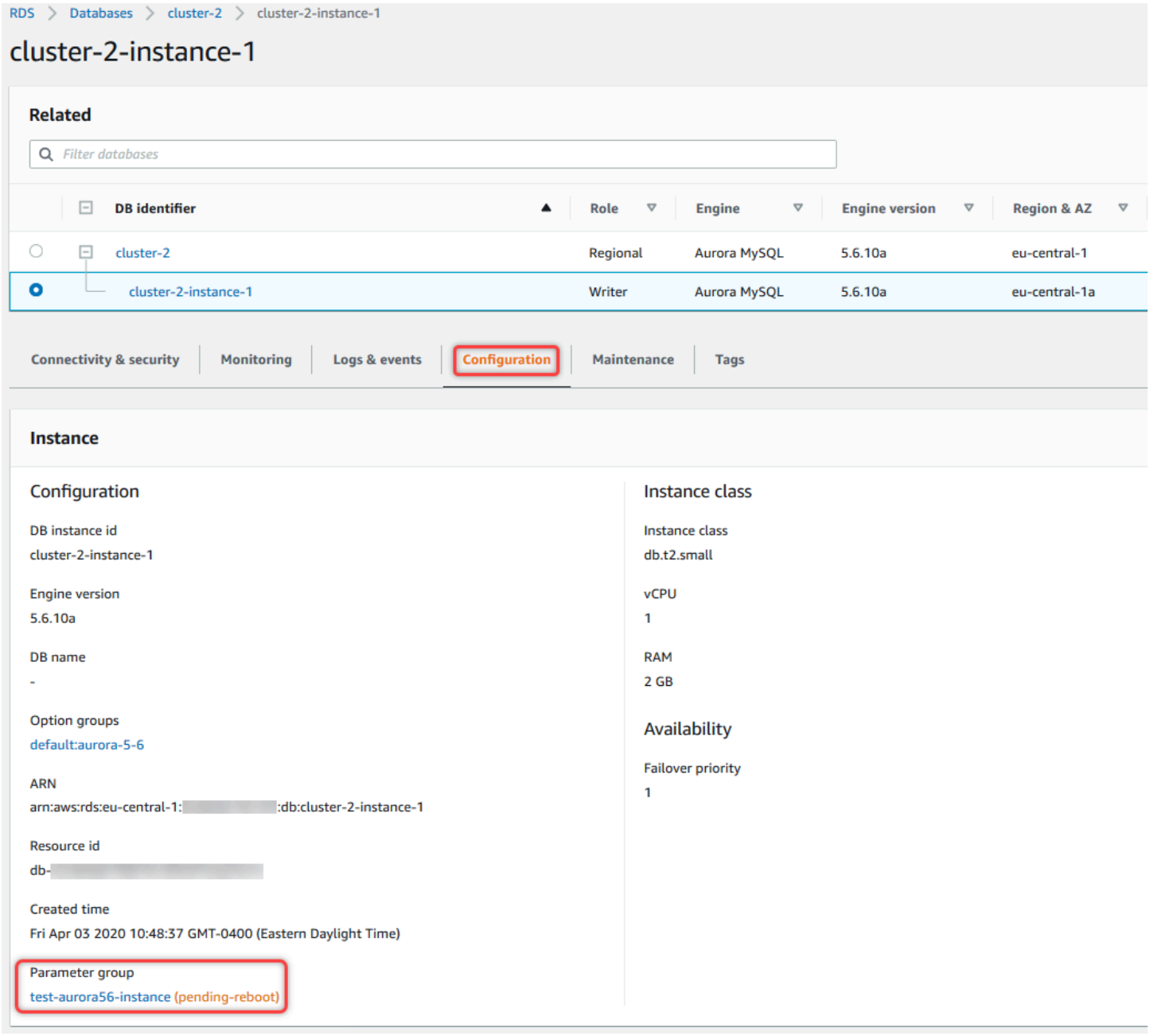
- `DBParameterGroupName`
- `Parameters`

在 Amazon Aurora 中将数据库参数组中的参数重置为其默认值

您可以将客户创建的数据库参数组中的参数值重置为默认值。对客户创建的数据库参数组中的参数所做的更改将应用于与此数据库参数组关联的所有数据库实例。

使用控制台时，可以将特定参数重置为其默认值。但是，您无法轻松地一次性重置数据库参数组中的所有参数。使用 AWS CLI 或 RDS API 时，可以将特定参数重置为其默认值。您还可以一次性重置数据库参数组中的所有参数。

对某些参数的更改将立即应用于数据库实例，而无需重新启动。而对其他一些参数进行的更改，只有在重新启动数据库实例之后，才会应用。RDS 控制台在 Configuration (配置) 选项卡上显示与数据库实例关联的数据库参数组的状态。例如，假设数据库实例未使用对其关联的数据库参数组所做的最新更改。如果是这样，RDS 控制台将显示状态为 pending-reboot（待重启）的数据库参数组。要向该数据库实例应用最新的参数更改，请手动重启数据库实例。



> **Note**
>
> 在默认数据库参数组中，参数始终设置为默认值。

控制台

将数据库参数组中的参数重置为默认值的方法

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择参数组。
3. 在列表中，选择参数组。
4. 对于 Parameter group actions (参数组操作)，选择 Edit (编辑)。
5. 选择要重置为默认值的参数。您可使用对话框右上方的箭头键滚动参数。

   您无法重置默认参数组中的值。
6. 选择重置，然后通过选择重置参数进行确认。

AWS CLI

要重置数据库参数组中的部分或所有参数，请使用带以下必需选项的 AWS CLI `reset-db-parameter-group` 命令：`--db-parameter-group-name`。

要重置数据库参数组中的所有参数，请指定 `--reset-all-parameters` 选项。要重置特定参数，请指定 `--parameters` 选项。

以下示例将名为 mydbparametergroup 的数据库参数组中的所有参数重置为其默认值。

Example

对于 Linux、macOS 或 Unix：

```
aws rds reset-db-parameter-group \
    --db-parameter-group-name mydbparametergroup \
    --reset-all-parameters
```

对于 Windows：

```
aws rds reset-db-parameter-group ^
    --db-parameter-group-name mydbparametergroup ^
    --reset-all-parameters
```

以下示例展示了在名为 mydbparametergroup 的数据库参数组中将 max_connections 和 max_allowed_packet 选项重置为各自的默认值。

Example

对于 Linux、macOS 或 Unix：

```
aws rds reset-db-parameter-group \
    --db-parameter-group-name mydbparametergroup \
    --parameters "ParameterName=max_connections,ApplyMethod=immediate" \
                 "ParameterName=max_allowed_packet,ApplyMethod=immediate"
```

对于 Windows：

```
aws rds reset-db-parameter-group ^
    --db-parameter-group-name mydbparametergroup ^
    --parameters "ParameterName=max_connections,ApplyMethod=immediate" ^
                 "ParameterName=max_allowed_packet,ApplyMethod=immediate"
```

该命令产生类似下面的输出：

```
DBParameterGroupName  mydbparametergroup
```

RDS API

要将数据库参数组中的参数重置为默认值，请使用带以下必需参数的 RDS API ResetDBParameterGroup 命令：DBParameterGroupName。

要重置数据库参数组中的所有参数，请将 ResetAllParameters 参数设置为 true。要重置特定参数，请指定 Parameters 参数。

在 Amazon Aurora 中复制数据库参数组

您可以复制您创建的自定义数据库参数组。复制参数组可能是一种方便的解决方案。例如，您创建了一个数据库参数组，并且想在新的数据库参数组中包含其大部分自定义参数和值。您可以使用 AWS Management Console复制数据库参数组。还可以使用 AWS CLI copy-db-parameter-group 命令或 RDS API CopyDBParameterGroup 操作。

复制数据库参数组之后，请至少等待 5 分钟，再创建使用该数据库参数组作为默认参数组的第一个数据库实例。这样，在使用参数组前，Amazon RDS 可以完成全部复制操作。这对于在为数据库实例创

建默认数据库时十分关键的参数非常重要。示例如 `character_set_database` 参数定义的默认数据库的字符集。请使用 Amazon RDS 控制台的参数组选项或使用 describe-db-parameters 命令来验证是否已创建数据库参数组。

> **Note**
>
> 您无法复制默认参数组。不过，您可以创建基于默认参数组的新参数组。
> 您无法将数据库参数组复制到其他 AWS 账户或 AWS 区域。

控制台

复制数据库参数组

1. 登录AWS Management Console并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择参数组。
3. 在列表中，选择要复制的自定义参数组。
4. 对于 Parameter group actions (参数组操作)，选择 Copy (复制)。
5. 在 New DB parameter group identifier (新数据库参数组标识符) 中，输入新参数组的名称。
6. 在 Description (描述) 中，输入新参数组的描述。
7. 选择 Copy (复制)。

AWS CLI

要复制数据库参数组，请使用 AWS CLI `copy-db-parameter-group` 命令及以下必要选项：

- `--source-db-parameter-group-identifier`
- `--target-db-parameter-group-identifier`
- `--target-db-parameter-group-description`

以下示例创建一个名为 `mygroup2` 的新数据库参数组，它是数据库参数组 `mygroup1` 的副本。

Example

对于 Linux、macOS 或 Unix：

```
aws rds copy-db-parameter-group \
    --source-db-parameter-group-identifier mygroup1 \
    --target-db-parameter-group-identifier mygroup2 \
    --target-db-parameter-group-description "DB parameter group 2"
```

对于 Windows：

```
aws rds copy-db-parameter-group ^
    --source-db-parameter-group-identifier mygroup1 ^
    --target-db-parameter-group-identifier mygroup2 ^
    --target-db-parameter-group-description "DB parameter group 2"
```

RDS API

要复制数据库参数组，请将 RDS API CopyDBParameterGroup 操作与下列必需参数配合使用：

- SourceDBParameterGroupIdentifier
- TargetDBParameterGroupIdentifier
- TargetDBParameterGroupDescription

在 Amazon Aurora 中列出数据库参数组

您可以列出为AWS 账户创建的数据库参数组。

> **Note**
>
> 当您为特定数据库引擎和版本创建数据库实例时，将自动从默认参数模板创建默认参数组。这些默认参数组包含首选参数设置，并且无法修改。当您创建自定义参数组时，可以修改参数设置。

控制台

列出 AWS 账户的所有数据库参数组

1. 登录AWS Management Console并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择参数组。

数据库参数组将显示在列表中。

AWS CLI

要列出 AWS 账户的所有数据库参数组，请使用 AWS CLI `describe-db-parameter-groups` 命令。

Example

下例列出了 AWS 账户的所有可用数据库参数组。

```
aws rds describe-db-parameter-groups
```

此命令会返回类似以下内容的响应：

```
DBPARAMETERGROUP  default.mysql8.0     mysql8.0  Default parameter group for MySQL8.0
DBPARAMETERGROUP  mydbparametergroup   mysql8.0  My new parameter group
```

以下示例描述了 mydbparamgroup1 参数组。

对于 Linux、macOS 或 Unix：

```
aws rds describe-db-parameter-groups \
    --db-parameter-group-name mydbparamgroup1
```

对于 Windows：

```
aws rds describe-db-parameter-groups ^
    --db-parameter-group-name mydbparamgroup1
```

此命令会返回类似以下内容的响应：

```
DBPARAMETERGROUP  mydbparametergroup1  mysql8.0  My new parameter group
```

RDS API

要列出 AWS 账户的所有数据库参数组，请使用 RDS API `DescribeDBParameterGroups` 操作。

在 Amazon Aurora 中查看数据库参数组的参数值

您可获得数据库参数组内所有参数的列表及它们的值。

控制台

查看数据库参数组的参数值

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择参数组。

   数据库参数组将显示在列表中。
3. 选择参数组名称以查看其参数列表。

AWS CLI

要查看数据库参数组的参数值，请使用带下列必需参数的 AWS CLI `describe-db-parameters` 命令。

- `--db-parameter-group-name`

Example

以下示例列出名为 mydbparametergroup 的数据库参数组的参数和参数值。

```
aws rds describe-db-parameters --db-parameter-group-name mydbparametergroup
```

此命令会返回类似以下内容的响应：

```
DBPARAMETER  Parameter Name            Parameter Value  Source           Data Type
 Apply Type  Is Modifiable
DBPARAMETER  allow-suspicious-udfs                      engine-default   boolean
 static      false
DBPARAMETER  auto_increment_increment                   engine-default   integer
 dynamic     true
DBPARAMETER  auto_increment_offset                      engine-default   integer
 dynamic     true
DBPARAMETER  binlog_cache_size         32768            system           integer
 dynamic     true
```

```
DBPARAMETER  socket                    /tmp/mysql.sock  system          string
 static      false
```

RDS API

要查看数据库参数组的参数值，请使用带下列所需参数的 RDS API DescribeDBParameters 命令。

- DBParameterGroupName

在 Amazon Aurora 中删除数据库参数组

您可以使用 AWS Management Console、AWS CLI 或 RDS API 删除数据库参数组。仅当参数组未与数据库实例关联时，才能将其删除。

控制台

删除数据库参数组

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择参数组。

   数据库参数组将显示在列表中。
3. 选择要删除的参数组的名称。
4. 选择操作，然后选择删除。
5. 查看参数组名称，然后选择删除。

AWS CLI

要删除数据库参数组，请使用带以下必需参数的 AWS CLI delete-db-parameter-group 命令。

- --db-parameter-group-name

Example

以下示例删除一个名为 mydbparametergroup 的数据库参数组。

```
aws rds delete-db-parameter-group --db-parameter-group-name mydbparametergroup
```

RDS API

要删除数据库参数组，请使用带以下必需参数的 RDS API DeleteDBParameterGroup 命令。

- DBParameterGroupName

比较数据库参数组

您可以使用 AWS Management Console查看两个数据库参数组之间的区别。

指定的参数组必须都是数据库参数组，或者必须都是数据库集群参数组。甚至当数据库引擎和版本相同时，也是如此。例如，您无法比较 aurora-mysql8.0（Aurora MySQL 版本 3）数据库参数组和 aurora-mysql8.0 数据库集群参数组。

您可以比较 Aurora MySQL 和 RDS for MySQL 数据库参数组（即使是不同版本），但您无法不能比较 Aurora PostgreSQL 和 RDS for PostgreSQL 数据库参数组。

比较两个数据库参数组

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择参数组。
3. 在此列表中，选择要比较的两个参数组。

   > **Note**
   >
   > 要将默认参数组与自定义参数组进行比较，首先请在默认选项卡上选择默认参数组，然后在自定义选项卡上选择自定义参数组。

4. 从操作中选择比较。

指定数据库参数

数据库参数类型包括：

- 整数
- Boolean
- String

- 长整型
- Double
- Timestamp
- 其他定义的数据类型的对象
- 整数、布尔值、字符串、long、double、时间戳或对象类型的值的数组

您还可以使用表达式、公式和函数指定整数和布尔参数。

目录

- 数据库参数公式
  - 数据库参数公式变量
  - 数据库参数公式运算符
- 数据库参数函数
- 数据库参数日志表达式
- 数据库参数值示例

数据库参数公式

数据库参数公式是一种可解析为整数值或布尔值的表达式。可以将表达式用大括号括起来：{}。可以为数据库参数值使用公式，也可以将公式用作数据库参数函数的参数。

语法

```
{FormulaVariable}
{FormulaVariable*Integer}
{FormulaVariable*Integer/Integer}
{FormulaVariable/Integer}
```

数据库参数公式变量

每个公式变量会返回一个整数或布尔值。变量的名称是区分大小写的。

AllocatedStorage

返回一个整数，它表示数据卷的大小（以字节为单位）。

DBInstanceClassMemory

该变量会返回一个整数，表示数据库进程可用的内存字节数。这个数字是以数据库实例类的内存总量开始在内部计算得出的。从此处，计算结果将减去为操作系统以及管理实例的 RDS 进程预留的内存。因此，该数字总是略低于 Amazon Aurora 数据库实例类 中实例类表所示的内存数字。确切的值取决于多种因素的组合。其中包括实例类、数据库引擎，以及其适用于 RDS 实例还是适用于属于 Aurora 集群一部分的实例。

DBInstanceVCPU

返回一个整数，它表示 Amazon RDS 用于管理实例的虚拟中央处理器 (vCPU) 的数量。

EndPointPort

返回一个整数，它表示连接到数据库实例时使用的端口。

TrueIfReplica

如果数据库实例是只读副本，返回 1，如果不是只读副本，则返回 0。这是 Aurora MySQL 中 `read_only` 参数的默认值。

数据库参数公式运算符

数据库参数公式支持两个运算符：除法和乘法。

除法运算符：/

用除数除以被除数，返回整数型商。商中的小数不四舍五入，直接截断。

语法

```
dividend / divisor
```

被除数和除数参数必须是整数型表达式。

乘法运算符：*

将表达式乘以表达式，返回表达式的乘积。表达式中的小数不四舍五入，直接截断。

语法

```
expression * expression
```

两个表达式必须都是整数型。

数据库参数函数

您可以将数据库参数函数的参数指定为整数或公式。每个函数必须具有至少一个参数。将多个参数指定为逗号分隔的列表。列表不能拥有任何空成员，如 argument1、argument3。函数名称区分大小写。

IF

返回一个参数。

语法

```
IF(argument1, argument2, argument3)
```

如果第一个参数的计算结果为 true，则返回第二个参数。否则返回第三个参数。

GREATEST

返回整数型或者参数公式列表中最大的值。

语法

```
GREATEST(argument1, argument2,...argumentn)
```

返回整数。

LEAST

返回整数型或者参数公式列表中最小的值。

语法

```
LEAST(argument1, argument2,...argumentn)
```

返回整数。

SUM

添加指定整数型或者参数公式的值。

语法

```
SUM(argument1, argument2,...argumentn)
```

返回整数。

数据库参数日志表达式

您可以为日志表达式设置整数数据库参数值。可以将表达式用大括号括起来：{}。例如：

```
{log(DBInstanceClassMemory/8187281418)*1000}
```

log 函数表示日志基数 2。此示例还使用了 DBInstanceClassMemory 公式变量。请参阅 数据库参数公式变量。

数据库参数值示例

这些示例展示了如何使用公式、函数和表达式来表达数据库参数的值。

> **Warning**
>
> 在数据库参数组内设置参数不当可能会产生意外的不利影响。可能包括性能下降和系统不稳定。修改数据库参数时应保持谨慎，且修改数据库参数组前要备份数据。将参数组更改应用到生产数据库实例前，请在使用时间点还原创建的测试数据库实例上试用这些参数组更改。

Example 使用数据库参数函数 LEAST

您可以指定 Aurora MySQL LEAST 参数值 中的 table_definition_cache 函数。使用它可以将可存储在定义缓存中的表定义的数量设置为 DBInstanceClassMemory/393040 或 20000 中的较小值。

```
LEAST({DBInstanceClassMemory/393040}, 20000)
```

将数据迁移到 Amazon Aurora 数据库集群

有多种方法可以将数据从现有数据库迁移到 Amazon Aurora 数据库群集，具体取决于数据库引擎兼容性。您的迁移选项还取决于您从中迁移数据的数据库和您迁移数据的规模。

将数据迁移到 Amazon Aurora MySQL 数据库集群

您可以将数据从以下来源之一迁移到 Amazon Aurora MySQL 数据库群集。

- RDS for MySQL 数据库实例
- 在 Amazon RDS 之外运行的 MySQL 数据库
- 不与 MySQL 兼容的数据库

有关更多信息，请参阅 将数据迁移到 Amazon Aurora MySQL 数据库集群。

将数据迁移到 Amazon Aurora PostgreSQL 数据库集群

您可以将数据从以下来源之一迁移到 Amazon Aurora PostgreSQL 数据库群集。

- Amazon RDS PostgreSQL 数据库实例
- 与 PostgreSQL 不兼容的数据库

有关更多信息，请参阅 将数据迁移到与 PostgreSQL 兼容的 Amazon Aurora。

使用 Aurora 数据库集群设置创建 Amazon ElastiCache 缓存

ElastiCache 是一项完全托管式内存缓存服务，提供的微秒级读写延迟可支持灵活的实时使用案例。ElastiCache 可以帮助提高应用程序和数据库的性能。您可以将 ElastiCache 用作不需要数据持久性的使用案例的主要数据存储，例如游戏排行榜、流媒体和数据分析。ElastiCache 可以帮助消除与部署和管理分布式计算环境相关的复杂性。有关更多信息，请参阅适用于 Memcached 的 Common ElastiCache Use Cases and How ElastiCache Can Help 和适用于 Redis OSS 的 Common ElastiCache Use Cases and How ElastiCache Can Help。您可以使用 Amazon RDS 控制台创建 ElastiCache 缓存。

您可以通过两种方式来使用 Amazon ElastiCache。您可以从无服务器缓存开始入手，也可以选择设计自己的缓存群集。如果您选择设计自己的缓存集群，ElastiCache 将同时使用 Redis OSS 和 Memcached 引擎。如果您不确定要使用哪个引擎，请参阅 Comparing Memcached and Redis OSS。有关 Amazon ElastiCache 的更多信息，请参阅 Amazon ElastiCache 用户指南。

主题

- 使用 Aurora 数据库集群设置创建 ElastiCache 缓存的概述
- 使用 Aurora 数据库集群中的设置创建 ElastiCache 缓存

使用 Aurora 数据库集群设置创建 ElastiCache 缓存的概述

您可以使用与新创建的或现有的 Aurora 数据库集群相同的配置设置，从 Amazon RDS 创建 ElastiCache 缓存。

将 ElastiCache 缓存与您的数据库集群关联的一些使用案例：

- 与单独在 RDS 上运行相比，将 ElastiCache 与 RDS 结合使用可以节省成本并提高性能。
- 您可以使用 ElastiCache 缓存作为不需要数据持久性的应用程序的主数据存储。使用 Redis OSS 或 Memcached 的应用程序几乎不需要任何修改就能使用 ElastiCache。

当您从 RDS 创建 ElastiCache 缓存时，ElastiCache 缓存会从关联的 Aurora 数据库集群继承以下设置：

- ElastiCache 连接设置
- ElastiCache 安全设置

您也可以根据要求设置缓存配置设置。

在应用程序中设置 ElastiCache

您的应用程序必须设置为使用 ElastiCache 缓存。还可以根据要求将应用程序设置为使用缓存策略，从而优化和提高缓存性能。

- 要访问并开始使用 ElastiCache 缓存，请参阅 Getting started with ElastiCache (Redis OSS) 和 Getting started with ElastiCache (Memcached)。
- 有关缓存策略的更多信息，请参阅适用于 Memcached 的 Caching strategies and best practices 和适用于 Redis 的 Caching strategies and best practices。
- 有关 ElastiCache（Redis OSS）集群中的高可用性的更多信息，请参阅 High availability using replication groups。
- 您可能会产生与备份存储、区域内或跨区域数据传输或使用 AWS Outposts 相关的成本。有关定价详细信息，请参阅 Amazon ElastiCache 定价。

使用 Aurora 数据库集群中的设置创建 ElastiCache 缓存

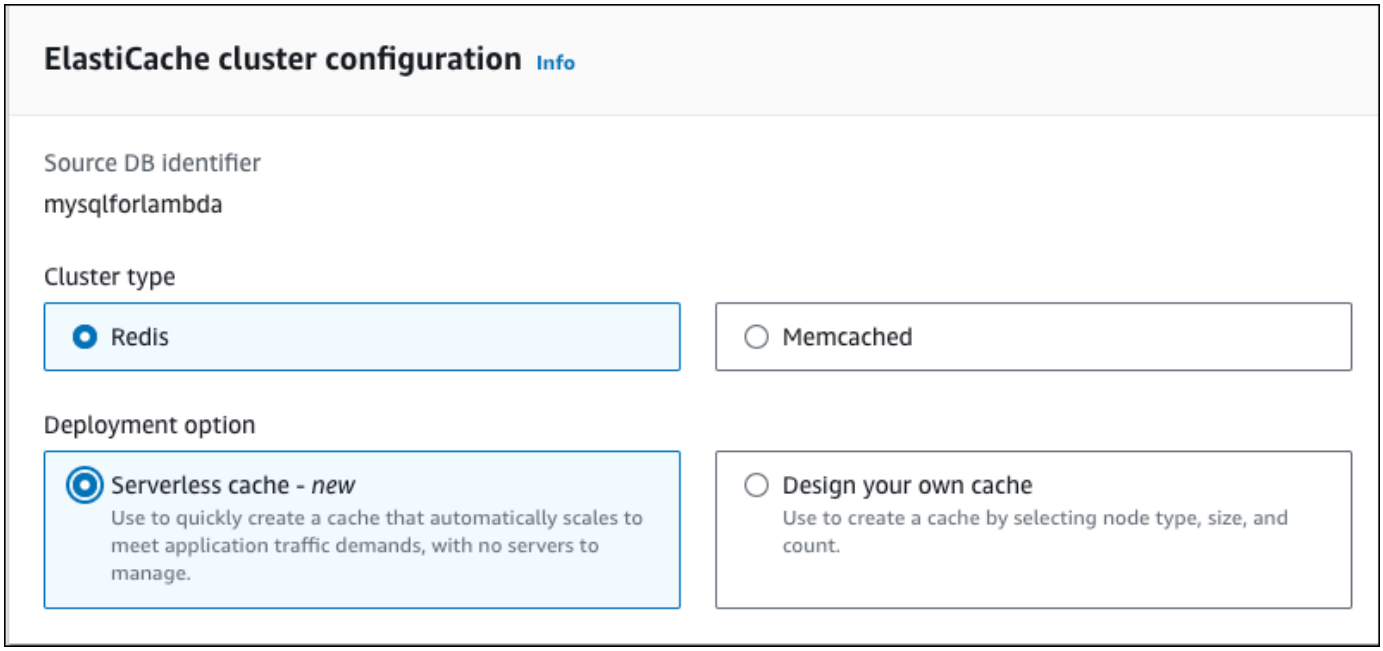
您可以使用继承自数据库集群的设置为 Aurora 数据库集群创建 ElastiCache 缓存。

使用数据库集群中的设置创建 ElastiCache 缓存

1. 要创建数据库集群，请按创建 Amazon Aurora 数据库集群中的说明操作。
2. 创建 Aurora 数据库集群后，控制台将显示建议的插件窗口。选择使用您的数据库设置从 RDS 创建 ElastiCache 集群。

   对于现有数据库，在数据库页面中，选择所需的数据库集群。在操作下拉菜单中，选择创建 ElastiCache 集群，以在 RDS 中创建与现有 Aurora 数据库集群具有相同设置的 ElastiCache 缓存。

   在 ElastiCache 配置部分，源数据库标识符显示 ElastiCache 缓存从哪个数据库集群继承设置。
3. 选择要创建 Redis OSS 集群还是 Memcached 集群。有关更多信息，请参阅 Comparing Memcached and Redis OSS。



4. 之后，选择是要创建无服务器缓存还是设计自己的缓存。有关更多信息，请参阅选择部署选项。

   如果您选择无服务器缓存：

   a. 在缓存设置中，输入名称和描述的值。

   b. 在查看默认设置下，保留默认设置以在缓存和数据库集群之间建立连接。

   c. 您也可以通过选择自定义默认设置来编辑默认设置。选择 ElastiCache 连接设置、ElastiCache 安全设置和最大使用限制。

5. 如果您选择设计自己的缓存：

   a. 如果您选择 Redis OSS 集群，请选择要将集群模式保持为启用还是禁用。有关更多信息，请参阅 Replication: Redis OSS (Cluster Mode Disabled) vs. Redis OSS (Cluster Mode Enabled)。

   b. 输入名称、描述和引擎版本的值。

      对于引擎版本，建议的原定设置值是最新的引擎版本。您也可以为 ElastiCache 缓存选择最符合您要求的引擎版本。

   c. 在节点类型选项中选择节点类型。有关更多信息，请参阅管理节点。

      如果您选择在集群模式设置为启用的情况下创建 Redis OSS 集群，请在分片数量选项中输入分片数（分区/节点组）。

      在副本数量中输入每个分片的副本数量。

> **Note**
>
> 所选节点类型、分片数量和副本数量都会影响您的缓存性能和资源成本。确保这些设置符合您的数据库需求。有关定价信息，请参阅 Amazon ElastiCache 定价。

d. 选择 ElastiCache 连接设置和 ElastiCache 安全设置。您可以保留默认设置或根据需要自定义这些设置。

6. 验证 ElastiCache 缓存的默认设置和继承设置。某些设置在创建后无法更改。

> **Note**
>
> RDS 可能会调整您的 ElastiCache 缓存的备份时段，以满足 60 分钟的最低时段要求。源数据库的备份时段保持不变。

7. 准备就绪后，请选择创建 ElastiCache 缓存。

控制台显示创建 ElastiCache 缓存的确认横幅。点击横幅中指向 ElastiCache 控制台的链接可查看缓存的详细信息。ElastiCache 控制台显示新创建的 ElastiCache 缓存。

使用 AWS Database Migration Service 将 EC2 数据库自动迁移到 Amazon Aurora

您可以使用 Aurora 控制台将 EC2 数据库迁移到 Aurora。Aurora 使用 AWS Database Migration Service（AWS DMS）迁移您的源 EC2 数据库。AWS DMS 允许您将关系数据库迁移到 AWS 云中。有关 AWS Database Migration Service 的更多信息，请参阅《AWS Database Migration Service 用户指南》中的什么是 AWS Database Migration Service？。

要开始迁移，您必须创建等效的 Aurora 数据库集群，以将数据迁移到其中。创建目标数据库后，您可以将 EC2 数据库导入其中。对于小于 1 TiB 的源数据库，此迁移操作可减少将数据迁移到 Aurora 所需的时间和资源。

概览

Aurora 控制台允许您将 EC2 数据库迁移到等效的 Aurora 数据库中。您必须创建 Aurora 数据库，才能从控制台进行迁移。

您可以为以下数据库引擎迁移 EC2 数据库：

- MySQL
- PostgreSQL

该迁移过程涉及到以下步骤：

- 在 Aurora 中创建等效的数据库。为了使这些数据库具有等效性，它们必须具有相同的数据库引擎和兼容的引擎版本。它们还必须位于同一 VPC 中。有关创建数据库的说明，请参阅 创建 Amazon Aurora 数据库集群 。
- 选择数据库的复制类型：
  - 完全加载迁移 – Aurora 将完整的源数据库复制到目标数据库，并在必要时在目标中创建新表。

    > **Note**
    >
    > 此选项会导致您的 Aurora 数据库中断。

  - 完全加载和更改数据捕获（CDC）迁移 – 与完全加载迁移类似，使用此选项，Aurora 会将完整的源数据库复制到目标数据库。但是，在完全加载迁移之后，Aurora 会将源中捕获的任何更改应用于目标数据库。更改数据捕获通过使用数据库引擎的本机 API 来收集对数据库日志的更改。

> **Note**
>
> 此选项会导致您的 Aurora 数据库中断。

- 更改数据捕获（CDC）– 使用此选项可使您的目标数据库在迁移过程中保持可用。Aurora 会将源数据库中正在进行的更改迁移到目标数据库。

- Aurora 会创建必要的联网资源来促进迁移。Aurora 在创建所需资源后，会通知您已创建的资源并允许您启动数据传输。

  完成迁移所需的时间取决于复制的类型和源数据库的大小。

先决条件

MySQL

在开始将 MySQL 数据库用作源数据库之前，请确保满足以下先决条件。这些先决条件适用于 AWS 托管的源。

您必须拥有具有复制管理员角色的 AWS DMS 的账户。该角色需要以下权限：

- REPLICATION CLIENT – 仅 CDC 任务需要此权限。换而言之，full-load-only 任务不需要此权限。
- REPLICATION SLAVE – 仅 CDC 任务需要此权限。换而言之，full-load-only 任务不需要此权限。

AWS DMS 用户还必须具有为复制指定的源表的 SELECT 权限。

如果您使用 MySQL 特定的迁移前评测功能，请授予以下权限。

```
grant select on mysql.user to <dms_user>;
grant select on mysql.db to <dms_user>;
grant select on mysql.tables_priv to <dms_user>;
grant select on mysql.role_edges to <dms_user>  #only for MySQL version 8.0.11 and
 higher
```

PostgreSQL

在从 AWS 托管的 PostgreSQL 源数据库迁移数据之前，请执行以下操作：

- 我们建议您使用具有 PostgreSQL 数据库实例所需最低权限的 AWS 用户账户作为 AWS DMS 的 PostgreSQL 源端点的用户账户。建议不要使用主账户。该账户必须具有 `rds_superuser` 和 `rds_replication` 角色。`rds_replication` 角色可授予管理逻辑时隙和使用逻辑时隙流式处理数据的权限。

> **Note**
>
> 一些 AWS DMS 事务在被 DMS 引擎再次使用之前会空闲一段时间。在 PostgreSQL 版本 9.6 及更高版本中使用参数 `idle_in_transaction_session_timeout`，能够导致空闲事务超时并失败。

限制

以下限制适用于自动迁移过程：

- 您的目标数据库状态必须为可用才能开始源数据库迁移。
- 当从 MySQL 源数据库迁移时，您的 Aurora 账户必须具有复制管理员角色。您还必须为该角色应用适当的权限。
- EC2 实例和目标数据库必须位于同一 VPC 中。
- 使用从 EC2 数据库迁移数据操作时，您无法将 EC2 数据库迁移到以下目标数据库：
  - Aurora global database
  - Aurora Limitless database
  - Aurora Serverless v1
  - MySQL 版本低于 5.7 的数据库
  - PostgreSQL 版本低于 10.4 的数据库

为同构迁移创建 IAM 资源

Aurora 使用 AWS DMS 迁移数据。为了访问数据库并迁移数据，AWS DMS 会为同构数据迁移创建无服务器环境。在此环境中，AWS DMS 需要访问 VPC 对等连接、路由表、安全组和其他 AWS 资源。此外，AWS DMS 将每个数据迁移的日志、指标和进度存储在 Amazon CloudWatch 中。要创建数据迁移项目，AWS DMS 需要访问这些服务。

此外，AWS DMS 需要访问代表一组用户凭证的密钥，以验证源和目标连接的数据库连接。

> **Note**
>
> 通过使用从 EC2 实例迁移数据操作，您可以使用 Aurora 控制台生成这些 IAM 资源。如果您使用控制台生成的 IAM 资源，则跳过此步骤。

此过程需要以下 IAM 资源：

主题

- 为同构数据迁移创建 IAM 策略
- 为同构数据迁移创建 IAM 角色
- 创建密钥访问策略和角色
- 为 AWS DMS 创建 IAM 角色以管理 Amazon VPC

为同构数据迁移创建 IAM 策略

在此步骤中，创建 IAM policy，向 AWS DMS 提供 Amazon EC2 和 CloudWatch 资源的访问权限。接下来，创建 IAM 角色并附加此策略。

为数据迁移创建 IAM 策略

1. 登录到 AWS Management Console，然后通过以下网址打开 IAM 控制台：https://console.aws.amazon.com/iam/。
2. 在导航窗格中，选择策略。
3. 选择创建策略。
4. 在创建策略页面中，选择 JSON 选项卡。
5. 将下面的 JSON 粘贴到编辑器中。

   JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeRouteTables",
```

```
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeVpcPeeringConnections",
                "ec2:DescribeVpcs",
                "ec2:DescribePrefixLists",
                "logs:DescribeLogGroups"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "servicequotas:GetServiceQuota"
            ],
            "Resource": "arn:aws:servicequotas:*:*:vpc/L-0EA8095F"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:DescribeLogStreams"
            ],
            "Resource": "arn:aws:logs:*:*:log-group:dms-data-migration-*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": "arn:aws:logs:*:*:log-group:dms-data-migration-*:log-
stream:dms-data-migration-*"
        },
        {
            "Effect": "Allow",
            "Action": "cloudwatch:PutMetricData",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateRoute",
                "ec2:DeleteRoute"
            ],
            "Resource": "arn:aws:ec2:*:*:route-table/*"
```

```
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateTags"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:security-group/*",
                "arn:aws:ec2:*:*:security-group-rule/*",
                "arn:aws:ec2:*:*:route-table/*",
                "arn:aws:ec2:*:*:vpc-peering-connection/*",
                "arn:aws:ec2:*:*:vpc/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:AuthorizeSecurityGroupEgress",
                "ec2:AuthorizeSecurityGroupIngress"
            ],
            "Resource": "arn:aws:ec2:*:*:security-group-rule/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:AuthorizeSecurityGroupEgress",
                "ec2:AuthorizeSecurityGroupIngress",
                "ec2:RevokeSecurityGroupEgress",
                "ec2:RevokeSecurityGroupIngress"
            ],
            "Resource": "arn:aws:ec2:*:*:security-group/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:AcceptVpcPeeringConnection",
                "ec2:ModifyVpcPeeringConnectionOptions"
            ],
            "Resource": "arn:aws:ec2:*:*:vpc-peering-connection/*"
        },
        {
            "Effect": "Allow",
            "Action": "ec2:AcceptVpcPeeringConnection",
            "Resource": "arn:aws:ec2:*:*:vpc/*"
```

```
        }
    ]
}
```

6. 选择下一步：标签和下一步：审核。
7. 在名称*中输入 **HomogeneousDataMigrationsPolicy**，然后选择创建策略。

为同构数据迁移创建 IAM 角色

在此步骤中，您将创建 IAM 角色，该角色提供对 AWS Secrets Manager、Amazon EC2 和 CloudWatch 的访问权限。

为数据迁移创建 IAM 角色

1. 登录 AWS Management Console，然后通过以下网址打开 IAM 控制台：https://console.aws.amazon.com/iam/。
2. 在导航窗格中，选择角色。
3. 选择创建角色。
4. 在选择可信实体页面中，在可信实体类型下选择 AWS 服务。对于其他 AWS 服务的应用场景，选择 DMS。
5. 选中 DMS 复选框，然后选择下一步。
6. 在添加权限页面上，选择之前创建的 HomogeneousDataMigrationsPolicy。选择下一步。
7. 在命名、检查并创建页面上，在角色名称中输入 **HomogeneousDataMigrationsRole**，然后选择创建角色。
8. 在角色页面上，在角色名称中输入 **HomogeneousDataMigrationsRole**。选择 HomogeneousDataMigrationsRole。
9. 在 HomogeneousDataMigrationsRole 页面上，选择信任关系选项卡。选择编辑信任策略。
10. 在编辑信任策略页面上，将以下 JSON 粘贴到编辑器中，替换现有文本。

    JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
```

```
            "Principal": {
                "Service": [
                    "dms-data-migrations.amazonaws.com",
                    "dms.your_region.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

在以上示例中，请用您的 AWS 区域名称替换 *your_region*。

上述基于资源的策略为 AWS DMS 服务主体提供了根据客户管理型 HomogeneousDataMigrationsPolicy 策略执行任务的权限。

11. 选择更新策略。

创建密钥访问策略和角色

按照以下步骤创建您的密钥访问策略和角色，允许 DMS 访问源数据库和目标数据库的用户凭证。

创建密钥访问策略和角色，以允许 Amazon RDS 访问 AWS Secrets Manager，从而访问相应的密钥

1. 登录到 AWS Management Console，然后通过以下网址打开 AWS Identity and Access Management（IAM）控制台：https://console.aws.amazon.com/iam/。
2. 选择策略，然后选择创建策略。
3. 选择 JSON 并输入以下策略，以启用对密钥的访问和解密。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "secretsmanager:GetSecretValue",
            "Resource": secret_arn,
        },
        {
             "Effect": "Allow",
             "Action": [
                        "kms:Decrypt",
```

```
                        "kms:DescribeKey"
                    ],
            "Resource": kms_key_arn,
        }
    ]
}
```

此处，*secret_arn* 是密钥的 ARN，您可以根据需要从任一 SecretsManagerSecretId 中获取，*kms_key_arn* 是您用于加密密钥的 AWS KMS 密钥的 ARN，如以下示例所示。

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "secretsmanager:GetSecretValue",
            "Resource": "arn:aws:secretsmanager:us-
east-2:123456789012:secret:MySQLTestSecret-qeHamH"
        },
        {
             "Effect": "Allow",
             "Action": [
                        "kms:Decrypt",
                        "kms:DescribeKey"
                    ],
             "Resource": "arn:aws:kms:us-
east-2:123456789012:key/761138dc-0542-4e58-947f-4a3a8458d0fd"
        }
    ]
}
```

> **Note**
>
> 如果您使用 AWS Secrets Manager 创建的默认加密密钥，则无需为 *kms_key_arn* 指定 AWS KMS 权限。
> 如果您希望策略提供对两个密钥的访问权限，只需为另一个 *secret_arn* 指定一个额外的 JSON 资源对象。

4. 查看并创建具友好名称和描述（可选）的策略。
5. 选择角色，然后选择创建角色。
6. 选择 AWS 服务作为可信实体的类型。
7. 从服务列表中选择 DMS 作为可信服务，然后选择下一步：权限。
8. 查找并附加您在步骤 4 中创建的策略，然后继续添加所有标签并查看角色。此刻，编辑角色的信任关系，以使用 Amazon RDS 区域服务主体作为可信实体。此主体采用以下格式。

```
dms.region-name.amazonaws.com
```

此处，*region-name* 是区域的名称，如 `us-east-1`。因此，此区域的 Amazon RDS 区域服务主体如下。

```
dms.us-east-1.amazonaws.com
dms-data-migrations.amazonaws.com
```

为 AWS DMS 创建 IAM 角色以管理 Amazon VPC

您必须为 AWS DMS 创建 IAM 角色，才能管理资源的 VPC 设置。要成功迁移，此角色必须可用。

为数据库迁移创建 **`dms-vpc-role`**

<result>

这样将为 DMS 创建角色来管理用于迁移的 VPC 设置。
</result>

1. 登录 AWS Management Console，然后打开 IAM 控制台：https://console.aws.amazon.com/iam/。
2. 在控制台的导航窗格中，选择 Roles，然后选择 Create role。
3. 对于选择可信实体选项，选择 AWS 服务选项。

   对于使用案例，选择 DMS。
4. 对于添加权限步骤，选择 `AmazonDMSVPCManagementRole` 并选择下一步。
5. 在命名、查看和创建页面中，将角色名称设置为 `dms-vpc-role`，并选择创建角色。

为 EC2 数据库设置数据迁移

要从 EC2 源数据库开始迁移数据，您必须创建一个等效的 Aurora 数据库。有关创建数据库的说明，请参阅 创建 Amazon Aurora 数据库集群 。

创建目标数据库后，使用以下步骤设置数据迁移：

设置数据迁移项目

1. 在 RDS 控制台的数据库页面上选择目标数据库。
2. 选择操作下拉列表并选择从 EC2 数据库迁移数据选项。要查看支持的目标数据库，请参阅 限制。
3. 在选择源 EC2 数据库部分下：

    1. 检查引擎类型并确保它与源数据库相同。

        另外，请检查引擎版本是否兼容。
    2. 对于 EC2 实例，请选择您的源数据库所驻留的 EC2 实例。
    3. 在端口中，输入源数据库允许流量的端口。
    4. 对于密钥，如果您没有现有密钥，请选择创建和使用新密钥。为您的源数据库输入用户名和密码。此外，选择要用于加密该密钥的 KMS 密钥。

        如果您有现有密钥，请选择使用现有密钥，然后从下拉列表中选择密钥。
    5. 对于密钥的 IAM 角色，如果您有现有的 IAM 角色，请选择使用现有的 IAM 角色，然后从下拉列表中选择一个可以访问上一步中的密钥 ID 的 IAM 角色。

        如果您没有现有的 IAM 角色，请选择创建和使用新的 IAM 角色。为 IAM 角色名称输入您的角色的新名称。您可以在下面的链接中查看与此角色关联的权限。
4. 在查看目标 RDS 数据库部分下：

    1. 在该部分顶部，确认目标数据库的设置。
    2. 对于密钥，如果您没有保存目标数据库凭证的现有密钥，请选择创建和使用新密钥。

        如果您有现有密钥，请从下拉列表中选择该密钥。
    3. 对于密钥的 IAM 角色，选择可以访问上一步中密钥的 IAM 角色。如果您没有现有 IAM 角色，还可以创建一个新 IAM 角色。

        如果下拉列表未填充 IAM 角色，请以 `arn:aws:iam:`*`account_id`*`:role/`*`roleName`* 格式指定 IAM 角色 ARN。

5. 在配置数据迁移部分下：

   1. 从完全加载、完全加载和更改数据捕获（CDC）或更改数据捕获（CDC）中选择数据迁移的类型。有关这些选项的详细信息，请参阅概览。

      迁移开始后，您无法修改迁移类型。

   2. 对于数据迁移的 IAM 角色，如果您有现有的 IAM 角色，请选择使用现有的 IAM 角色，然后从下拉列表中选择一个 IAM 角色，该角色将授予 DMS 创建迁移所需资源的权限。如果您没有现有的 IAM 角色，请选择创建和使用新的 IAM 角色。
6. 确认查看迁移设置选项卡显示了成功设置数据迁移所需的设置。
7. 选择迁移以完成迁移设置。

完成这些步骤后，您可以通过在控制台的进度横幅中选择查看详细信息来查看正在为数据迁移设置的资源。设置好所需资源后，迁移将自动开始。如果您创建

要将多个数据库迁移到目标数据库，请使用有关新 EC2 数据库的详细信息重新开始此过程。

管理数据迁移

从 RDS 控制台使用从 EC2 数据库迁移数据操作后，Aurora 会自动启动迁移。

如果您使用 AWS DMS 控制台创建迁移资源，则可以启动迁移过程。

启动数据迁移

请按照以下步骤启动数据迁移：

启动数据迁移

1. 在 RDS 控制台的数据库页面上选择目标数据库。
2. 在数据库详细信息页面上，选择数据迁移选项卡。
3. 在数据迁移选项卡下，关联的数据迁移列出了可用的数据迁移。

   设置所需资源后，使用 Aurora 控制台设置的迁移会自动启动。

   使用 DMS 控制台设置的迁移将设置为就绪。

   要开始这些迁移，请选择操作下拉列表并选择启动。
4. 这样将开始 EC2 数据库的数据迁移。

停止数据迁移

对于复制类型为完全加载的数据迁移，停止迁移会导致迁移过程停止且无法恢复。停止迁移后，必须重新启动迁移。

对于复制类型设置为更改数据捕获（CDC）或完全加载和 CDC 的迁移，您可以停止持续复制过程，稍后再恢复该过程。

停止数据迁移

1. 在 RDS 控制台的数据库页面上选择目标数据库。
2. 在数据库详细信息页面上，选择数据迁移选项卡。
3. 在数据迁移选项卡下，关联的数据迁移列出了正在进行的数据迁移。

   要停止迁移，请选择数据迁移，然后在操作下拉列表中选择停止。
4. 这样将停止 EC2 数据库的数据迁移。

恢复数据迁移

对于复制类型为完全加载和更改数据捕获（CDC）或更改数据捕获（CDC）迁移的数据迁移，您可以从最后一个停止点恢复 CDC 流程。

恢复数据迁移

1. 在 RDS 控制台的数据库页面上选择目标数据库。
2. 在数据库详细信息页面上，选择数据迁移选项卡。
3. 在数据迁移选项卡下，关联的数据迁移列出了已停止的数据迁移。

   要恢复迁移，请选择数据迁移，然后在操作下拉列表中选择恢复处理。
4. 这样将恢复 EC2 数据库的数据迁移。

删除数据迁移

要删除关联的数据迁移，请使用以下说明：

删除数据迁移

1. 在 RDS 控制台的数据库页面上选择目标数据库。

2. 在数据库详细信息页面上，选择数据迁移选项卡。
3. 要删除迁移，请选择数据迁移，然后在操作下拉列表中选择删除。
4. 这样将删除相应的数据迁移。

删除正在进行的数据迁移不会影响已加载到目标数据库的任何数据。

重新启动数据迁移

要从 CDC 起点重新启动关联的数据迁移，请按照以下说明进行操作。

重新启动数据迁移

1. 在 RDS 控制台的数据库页面上选择目标数据库。
2. 在数据库详细信息页面上，选择数据迁移选项卡。
3. 要重新启动迁移，请选择数据迁移，然后在操作下拉列表中选择重新启动。
4. 这样将从 CDC 起点重新启动数据迁移。

重新启动正在进行的数据迁移不会影响已加载到目标数据库的任何数据。

监控数据迁移

数据迁移开始后，您可以监控其状态和进度。大型数据集的数据迁移需要数小时才能完成。为了保持数据迁移的可靠性、可用性和高性能，请定期监控迁移进度。

检查数据迁移的状态和进度

1. 在 RDS 控制台的数据库页面上选择目标数据库。
2. 在数据库详细信息页面上，选择数据迁移选项卡。
3. 关联的数据迁移部分列出了您的数据迁移。检查状态列。
4. 对于正在运行的数据迁移，迁移过程列显示了已迁移数据的百分比。
5. 要在 CloudWatch 中监控进程，请使用 CloudWatch 列中的链接。

迁移状态

对于您运行的每个数据迁移，Aurora 控制台都会显示状态。以下列表包括状态：

- Ready：数据迁移已准备好开始。

- `Starting`：Aurora 正在为您的数据迁移创建无服务器环境。
- `Load running`：Aurora 正在执行完全加载迁移。
- `Load complete, replication ongoing`：Aurora 已完成完全加载，现在复制正在进行的更改。此状态仅适用于完全加载和 CDC 类型的迁移。
- `Replication ongoing`：Aurora 正在复制正在进行的更改。此状态仅适用于 CDC 类型的迁移。
- `Stopping`：Aurora 正在停止数据迁移。当您从操作菜单中选择停止数据迁移时，此状态适用。
- `Stopped`：Aurora 已停止数据迁移。
- `Failed`：数据迁移已失败。有关更多信息，请参阅日志文件。
- `Restarting`：数据迁移已从 CDC 起点重新启动了正在进行的数据复制。

教程：使用自定义参数组创建 MySQL 数据库集群

在本教程中，您将使用自定义参数组，来创建 MySQL 数据库集群。有关参数组的更多信息，请参阅 Amazon Aurora 数据库集群的数据库集群参数组。

> **Important**
>
> 创建 AWS 账户并不会收费；但是，在完成本教程过程中使用 AWS 资源可能会产生费用。完成本教程后，如果不再需要这些资源，可以将其删除。

要使用自定义配置和设置创建数据库集群，您可以使用自定义参数组。如果您使用多个数据库并希望为实例集统一配置设置，则自定义参数组特别有用。

通过完成这些步骤，您将学习：

- 如何使用 Amazon Aurora，通过自定义参数组创建 MySQL 数据库集群。
- 如何为 MySQL 数据库集群使用特定的参数。

要完成本教程，请执行以下任务：

1. 使用 MySQL 参数 `default_password_lifetime` 创建一个数据库集群参数组。
2. 使用您创建的自定义数据库集群参数组创建 MySQL 数据库集群。

主题

- 先决条件
- 创建 Amazon Aurora 数据库集群参数组
- 修改自定义参数组中的参数值
- 使用数据库集群参数组创建 MySQL 数据库集群

先决条件

本教程要求您拥有一个 AWS 账户和一个具有管理权限的用户。如果您还没有设置这些内容，请完成以下各节中的步骤：

- 注册 AWS 账户

- 创建具有管理访问权限的用户

创建 Amazon Aurora 数据库集群参数组

在本教程中，您将了解如何在控制台中为 MySQL 数据库集群创建一个具有 default_password_lifetime 的自定义参数组。default_password_lifetime 参数控制客户端密码自动过期之前的天数。有关可用于 MySQL 数据库集群的其它参数的更多信息，请参阅 Aurora MySQL 配置参数。

创建参数组

1. 打开 Amazon RDS 控制台，然后选择参数组。
2. 对于自定义参数组，选择创建参数组。
3. 设置参数组详细信息。
    1. 输入参数组的名称。
    2. 输入参数组的描述。
    3. 对于引擎类型，选择 。Aurora MySQL 支持蓝绿部署。
    4. 对于参数组系列，选择 aurora-mysql8.0。
    5. 对于类型，选择数据库集群参数组。
4. 选择创建。

新的数据库集群参数组将显示在 Amazon RDS 控制台的参数组页面上。以下步骤说明如何修改参数值以自定义参数组。

修改自定义参数组中的参数值

使用以下步骤修改您在创建 Amazon Aurora 数据库集群参数组中创建的参数组中的参数值。

修改参数组中的参数值

1. 打开 Amazon RDS 控制台，然后选择参数组。
2. 对于自定义参数组，请选择您创建的数据库集群参数组的名称。
3. 选择编辑。
4. 在筛选参数搜索框中，搜索自定义参数 default_password_lifetime。
5. 选中参数旁边的复选框，并输入要为此密码生命周期参数设置的天数值。
6. 选择保存更改。

自定义参数组现在可以与 Amazon Aurora for MySQL 8.0 数据库集群相关联。

使用数据库集群参数组创建 MySQL 数据库集群

最后，使用您在前面的步骤中创建的自定义参数组，创建 MySQL 数据库集群。以下步骤显示了如何使用自定义参数组创建 MySQL 数据库集群。

使用自定义参数和新选项组创建数据库集群

1. 打开 Amazon RDS 控制台并选择数据库。
2. 选择创建数据库。
3. 对于选择数据库创建方法，选择标准创建。
4. 对于引擎选项，选择 Aurora（MySQL 兼容）。
5. 选择其它配置。
   - 对于初始数据库名称，为数据库集群选择名称。
   - 在数据库集群参数组下拉列表中，选择您之前创建的数据库集群参数组的名称。
6. 对于本教程，可以保留任何其它数据库设置的默认设置，也可以根据需要对其进行修改。
7. 选择创建数据库。

RDS 使用自定义参数组创建新的 MySQL 数据库集群。要查看有关此数据库的更多信息，请参阅 Amazon RDS 控制台的数据库页面。

在本教程中，您使用自定义参数组，通过定制的设置配置了 MySQL 数据库集群。这个新创建的 MySQL 数据库集群使用 `default_password_lifetime` 参数来管理用户密码生命周期。要优化数据库，可以在自定义参数组中应用其它设置并添加选项。

创建完自定义数据库集群后，应删除资源，以免产生不必要的费用。要删除数据库集群，请按照删除 Aurora 数据库集群和数据库实例中的说明操作。

管理 Amazon Aurora 数据库集群

本节说明如何管理和维护 Aurora 数据库集群。Aurora 设计在以复制拓扑形式连接的数据库服务器的集群。因此，管理 Aurora 通常涉及将更改部署到多个服务器，以及确保所有 Aurora 副本与源服务器保持同步。由于 Aurora 会随着数据增加透明地扩展底层存储，因此管理 Aurora 需要相对较少的磁盘存储管理。同样，由于 Aurora 自动执行持续备份，因此，Aurora 集群无需进行详细的计划或考虑执行备份的停机时间。

主题

- 停止和启动 Amazon Aurora 数据库集群
- 自动连接 EC2 实例和 Aurora 数据库集群
- 自动连接 Lambda 函数和 Aurora 数据库集群
- 修改 Amazon Aurora 数据库集群
- 将 Aurora 副本添加到数据库集群
- 管理 Aurora 数据库集群的性能和扩展
- 克隆 Amazon Aurora 数据库集群卷
- 将 Aurora 与其他AWS服务集成
- 维护 Amazon Aurora 数据库集群
- 重启 Amazon Aurora 数据库集群或 Amazon Aurora 数据库实例
- 对 Amazon Aurora 数据库集群进行失效转移
- 删除 Aurora 数据库集群和数据库实例
- 为 Amazon Aurora 和 Amazon RDS 资源添加标签
- Amazon RDS 中的 Amazon 资源名称（ARN）
- Amazon Aurora 更新

停止和启动 Amazon Aurora 数据库集群

停止和启动 Aurora 数据库集群有助于您控制开发和测试环境的成本。您可以暂时停止集群中的所有数据库实例，而不是每次使用集群时设置和停用所有数据库实例。

主题

- 停止和启动 Aurora 数据库集群概述
- 停止和启动 Aurora 数据库集群的限制
- 停止 Aurora 数据库集群
- 停止 Aurora 数据库集群后可以执行的操作
- 启动 Aurora 数据库集群

停止和启动 Aurora 数据库集群概述

在不需要使用 Aurora 数据库集群期间，您可以同时停止该集群中的所有实例。您可以在需要使用时再次启动集群。启动和停止简化了用于开发、测试或不需要持续可用性的类似活动的集群的设置和停用过程。您可以仅使用单个操作执行涉及的所有 AWS Management Console 过程，而无论在集群中具有多少个实例。

在停止数据库集群后，您只需在指定的保留时段内为集群存储、手动快照和自动备份存储付费。您无需为任何数据库实例小时数付费。

> **Important**
>
> 您可以停止数据库集群最多 7 天。如果您在七天后未手动启动数据库集群，则数据库集群将自动启动，这样它就不会落后于任何所需的维护更新。

为了最大限度减少具有较少负载的 Aurora 集群的费用，您可以停止集群而不是删除它的 Aurora 所有副本。对于具有超过一个或两个实例的集群，则只能使用 AWS CLI 或 Amazon RDS API 频繁删除和重新创建数据库实例。也可能很难按正确的顺序执行此类操作序列，例如，在删除主实例之前删除所有 Aurora 副本以避免激活故障转移机制。

如果您需要将数据库集群保持运行状态，但具有的容量超过所需的容量，请不要使用启动和停止。如果您的集群成本太高或不太繁忙，请删除一个或多个数据库实例，或者将所有数据库实例更改为较小的实例类。您无法停止单个 Aurora 数据库实例。

停止数据库集群的时间因数据库实例类、网络状态、数据库引擎类型和数据库状态等因素而异。此过程可能需要几分钟时间。Amazon RDS 服务执行以下操作：

- 关闭数据库引擎进程。
- 关闭 RDS 平台进程。
- 终止底层 Amazon EC2 实例。

重启数据库集群的时间因数据库大小、数据库实例类、网络状态、数据库引擎类型和集群关闭时的数据库状态等因素而异。启动过程可能需要几分钟到几小时，但通常是几分钟时间。我们建议您在创建可用性计划时考虑启动时间的可变性。

要启动数据库集群，该服务会执行如下操作：

- 预置底层 Amazon EC2 实例。
- 启动 RDS 平台进程。
- 启动数据库引擎进程。
- 恢复数据库实例（即使在正常关闭后也会发生恢复）。

停止和启动 Aurora 数据库集群的限制

有些 Aurora 集群无法停止和启动。

- 只有当属于 Aurora Global Database 的集群是全球数据库中的唯一集群时，您才可以启动和停止该集群。
- 您无法停止和启动具有跨区域只读副本的集群。
- 您无法停止和启动属于蓝绿部署一部分的集群。
- 您无法停止和启动 Aurora Serverless v1 集群。使用 Aurora Serverless v2，您可以停止和启动该集群。

停止 Aurora 数据库集群

要使用 Aurora 数据库集群或执行管理，应始终先运行 Aurora 数据库集群，停止该集群，然后再次启动该集群。在停止集群后，您需要在指定的保留时段内为集群存储、手动快照和自动备份存储付费，但不需要为数据库实例小时数付费。

停止操作先停止 Aurora 副本实例，然后停止主实例以避免激活故障转移机制。

控制台

停止 Aurora 集群

1. 登录AWS Management Console并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Databases (数据库)，然后选择集群。您可以从该页面中执行停止操作，或者导航到要停止的数据库集群的详细信息页面。
3. 对于 Actions（操作），选择 Stop temporarily（临时停止）。
4. 在 Stop DB cluster temporarily（临时停止数据库集群）窗口中，选择确认数据库集群将在 7 天后自动重新启动。
5. 选择 Stop temporarily（临时停止）以停止数据库集群，或选择 Cancel（取消）以取消该操作。

AWS CLI

要使用 AWS CLI 停止数据库实例，请使用以下参数调用 stop-db-cluster 命令：

- `--db-cluster-identifier` – Aurora 集群的名称。

Example

```
aws rds stop-db-cluster --db-cluster-identifier mydbcluster
```

RDS API

要使用 Amazon RDS API 停止数据库实例，请使用以下参数调用 StopDBCluster 操作：

- `DBClusterIdentifier` – Aurora 集群的名称。

停止 Aurora 数据库集群后可以执行的操作

在停止 Aurora 集群后，您可以执行时间点还原以还原到指定的自动备份保留时段中的任意时间点。有关执行时间点还原的详细信息，请参阅还原数据。

在停止 Aurora 数据库集群后，您无法修改该集群或其任何数据库实例的配置。您也无法在该集群中添加或删除数据库实例，或者，如果仍具有任何关联的数据库实例，则无法删除该集群。您必须在执行任何此类管理操作之前启动该集群。

停止数据库集群会移除待处理操作，但数据库集群参数组或数据库集群实例的数据库参数组除外。

在再次启动后，Aurora 将任何计划的维护应用于停止的集群。请记住，在 7 天后，Aurora 自动启动任何停止的集群，以使其维护状态不会落后太多。

Aurora 也不会执行任何自动备份，因为停止集群后无法更改基础数据。Aurora 不会延长数据库集群停止期间的备份保留期。

启动 Aurora 数据库集群

在启动 Aurora 数据库集群时，您始终从已处于停止状态的 Aurora 集群开始。在启动集群时，它的所有数据库实例将再次变得可用。集群保留其配置设置，例如，终端节点、参数组和 VPC 安全组。

启动数据库集群通常需要几分钟时间。

控制台

启动 Aurora 集群

1. 登录AWS Management Console并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Databases (数据库)，然后选择集群。您可以从该页面中执行启动操作，或者导航到要启动的数据库集群的详细信息页面。
3. 对于 Actions (操作)，选择 Start (开始)。

AWS CLI

要使用 AWS CLI 启动数据库集群，请使用以下参数调用 start-db-cluster 命令：

- `--db-cluster-identifier` – Aurora 集群的名称。该名称是在创建集群时选择的特定集群标识符，或者是选择的数据库实例标识符加上在末尾附加的 `-cluster`。

Example

```
aws rds start-db-cluster --db-cluster-identifier mydbcluster
```

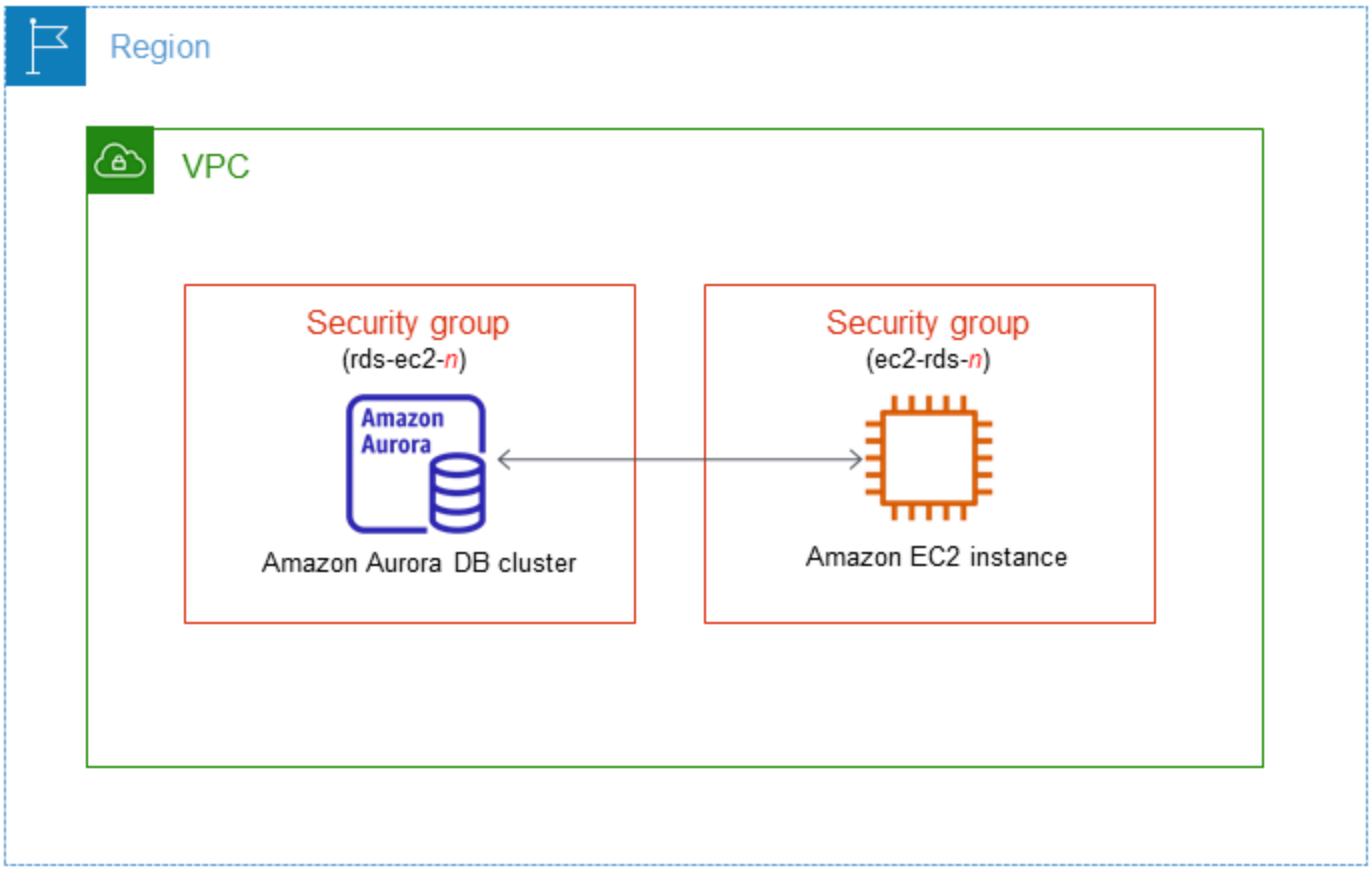
RDS API

要使用 Amazon RDS API 启动 Aurora 数据库集群，请使用以下参数调用 StartDBCluster 操作：

- DBCluster – Aurora 集群的名称。该名称是在创建集群时选择的特定集群标识符，或者是选择的数据库实例标识符加上在末尾附加的 -cluster。

自动连接 EC2 实例和 Aurora 数据库集群

您可以使用 RDS 控制台来简化在 Amazon Elastic Compute Cloud（Amazon EC2）实例和 Aurora 数据库集群之间设置连接的过程。通常，数据库集群位于私有子网中，而 EC2 实例位于 VPC 内的公有子网中。您可以在 EC2 实例上使用 SQL 客户端连接到数据库集群。EC2 实例还可以运行访问私有数据库集群的 Web 服务器或应用程序。



如果您想连接到与 Aurora 数据库集群不在同一 VPC 中的 EC2 实例，请参阅在 VPC 中访问数据库集群的场景中的相应方案。

主题

- 与 EC2 实例的自动连接概述
- 自动连接 EC2 实例和 Aurora 数据库集群
- 查看连接的计算资源
- 连接到运行特定数据库引擎的数据库实例

与 EC2 实例的自动连接概述

当您在 EC2 实例和 Aurora 数据库集群之间设置连接时，Amazon RDS 会自动为您的 EC2 实例和数据库集群配置 VPC 安全组。

以下是将 EC2 实例与 Aurora 数据库集群连接的要求：

- EC2 实例必须与 数据库集群存在于同一 VPC 中。

  如果同一 VPC 中不存在任何 EC2 实例，则控制台将提供创建一个此类实例的链接。
- 目前，数据库集群不能是 Aurora Serverless 数据库集群，也不能是 Aurora 全局数据库的一部分。
- 设置连接的用户必须具有执行以下 Amazon EC2 操作的权限：
  - `ec2:AuthorizeSecurityGroupEgress`
  - `ec2:AuthorizeSecurityGroupIngress`
  - `ec2:CreateSecurityGroup`
  - `ec2:DescribeInstances`
  - `ec2:DescribeNetworkInterfaces`
  - `ec2:DescribeSecurityGroups`
  - `ec2:ModifyNetworkInterfaceAttribute`
  - `ec2:RevokeSecurityGroupEgress`

如果数据库实例和 EC2 实例位于不同的可用区，则您的账户可能会产生跨可用区成本。

当您设置与 EC2 实例的连接时，Amazon RDS 会根据与数据库集群和 EC2 实例关联的安全组的当前配置采取操作，如下表所述。

当前 RDS 安全组配置	当前 EC2 安全组配置	RDS 操作
有一个或多个安全组与 数据库集群 [其名称与模式 `rds-ec2-`*n*（其中 *n* 是数字）相匹配] 关联。尚未修改与此模式匹配的安全组。该安全组只具有一条以 EC2 实例的 VPC 安全组作为源的入站规则。	有一个或多个安全组与 EC2 实例关联，此实例的名称与模式 `ec2-rds-`*n*（其中 *n* 是数字）相匹配。尚未修改与此模式匹配的安全组。该安全组只具有一条以 数据库集群的 VPC 安全组作为源的出站规则。	RDS 不执行任何操作。 已在 EC2 实例和 数据库集群之间自动配置了连接。由于 EC2 实例和 RDS 数据库之间已经存在连接，因此不会修改安全组。

当前 RDS 安全组配置	当前 EC2 安全组配置	RDS 操作
以下任一条件适用： • 没有与 数据库集群（其名称与模式 `rds-ec2-`*n* 匹配）关联的安全组。 • 有一个或多个安全组与 数据库集群（其名称与模式 `rds-ec2-`*n* 相匹配）关联。但是，Amazon RDS 不能使用这些安全组中的任何一个来连接 EC2 实例。如果安全组没有一条以 EC2 实例的 VPC 安全组作为源的入站规则，则 Amazon RDS 无法使用该安全组。Amazon RDS 也无法使用经过修改的安全组。修改示例包括添加规则或更改现有规则的端口。	以下任一条件适用： • 没有与 EC2 实例（其名称与模式 `ec2-rds-`*n* 匹配）关联的安全组。 • 有一个或多个安全组与 EC2 实例关联，此实例的名称与模式 `ec2-rds-`*n* 相匹配。但是，Amazon RDS 不能将其中任何安全组用于连接数据库集群。如果安全组没有一条以数据库集群的 VPC 安全组作为源的出站规则，则 Amazon RDS 无法使用该安全组。Amazon RDS 也无法使用经过修改的安全组。	RDS action: create new security groups
有一个或多个安全组与 数据库集群（其名称与模式 `rds-ec2-`*n* 相匹配）关联。尚未修改与此模式匹配的安全组。该安全组只具有一条以 EC2 实例的 VPC 安全组作为源的入站规则。	有一个或多个安全组与 EC2 实例关联，此实例的名称与模式 `ec2-rds-`*n* 相匹配。但是，Amazon RDS 不能将其中任何安全组用于连接数据库集群。如果安全组没有一条以数据库集群的 VPC 安全组作为源的出站规则，则 Amazon RDS 无法使用该安全组。Amazon RDS 也无法使用经过修改的安全组。	RDS action: create new security groups

当前 RDS 安全组配置	当前 EC2 安全组配置	RDS 操作
有一个或多个安全组与 数据库集群（其名称与模式 `rds-ec2-`*n* 相匹配）关联。尚未修改与此模式匹配的安全组。该安全组只具有一条以 EC2 实例的 VPC 安全组作为源的入站规则。	存在用于连接的有效 EC2 安全组，但它与 EC2 实例不关联。此安全组的名称与模式 `ec2-rds-`*n* 相匹配。尚未修改它。它只具有一条以 数据库集群的 VPC 安全组作为源的出站规则。	RDS action: associate EC2 security group
以下任一条件适用： • 没有与 数据库集群（其名称与模式 `rds-ec2-`*n* 匹配）关联的安全组。 • 有一个或多个安全组与 数据库集群（其名称与模式 `rds-ec2-`*n* 相匹配）关联。但是，Amazon RDS 不能使用这些安全组中的任何一个来连接 EC2 实例。如果安全组没有一条以 EC2 实例的 VPC 安全组作为源的入站规则，则 Amazon RDS 无法使用该安全组。Amazon RDS 也无法使用经过修改的安全组。	有一个或多个安全组与 EC2 实例关联，此实例的名称与模式 `ec2-rds-`*n* 相匹配。尚未修改与此模式匹配的安全组。该安全组只具有一条以 数据库集群的 VPC 安全组作为源的出站规则。	RDS action: create new security groups

RDS 操作：创建新的安全组

Amazon RDS 执行以下操作：

- 创建与模式 `rds-ec2-`*n* 匹配的新安全组。该安全组具有一条以 EC2 实例的 VPC 安全组作为源的入站规则。此安全组与 数据库集群关联，并允许 EC2 实例访问 数据库集群。
- 创建与模式 `ec2-rds-`*n* 匹配的新安全组。该安全组具有一条以数据库集群的 VPC 安全组作为目标的出站规则。该安全组与 EC2 实例相关联，并允许 EC2 实例向 数据库集群发送流量。

RDS 操作：关联 EC2 安全组

Amazon RDS 将有效的现有 EC2 安全组与 EC2 实例关联。该安全组允许 EC2 实例向 数据库集群发送流量。

自动连接 EC2 实例和 Aurora 数据库集群

在 EC2 实例与 Aurora 数据库集群之间设置连接之前，请确保满足与 EC2 实例的自动连接概述中所述的要求。

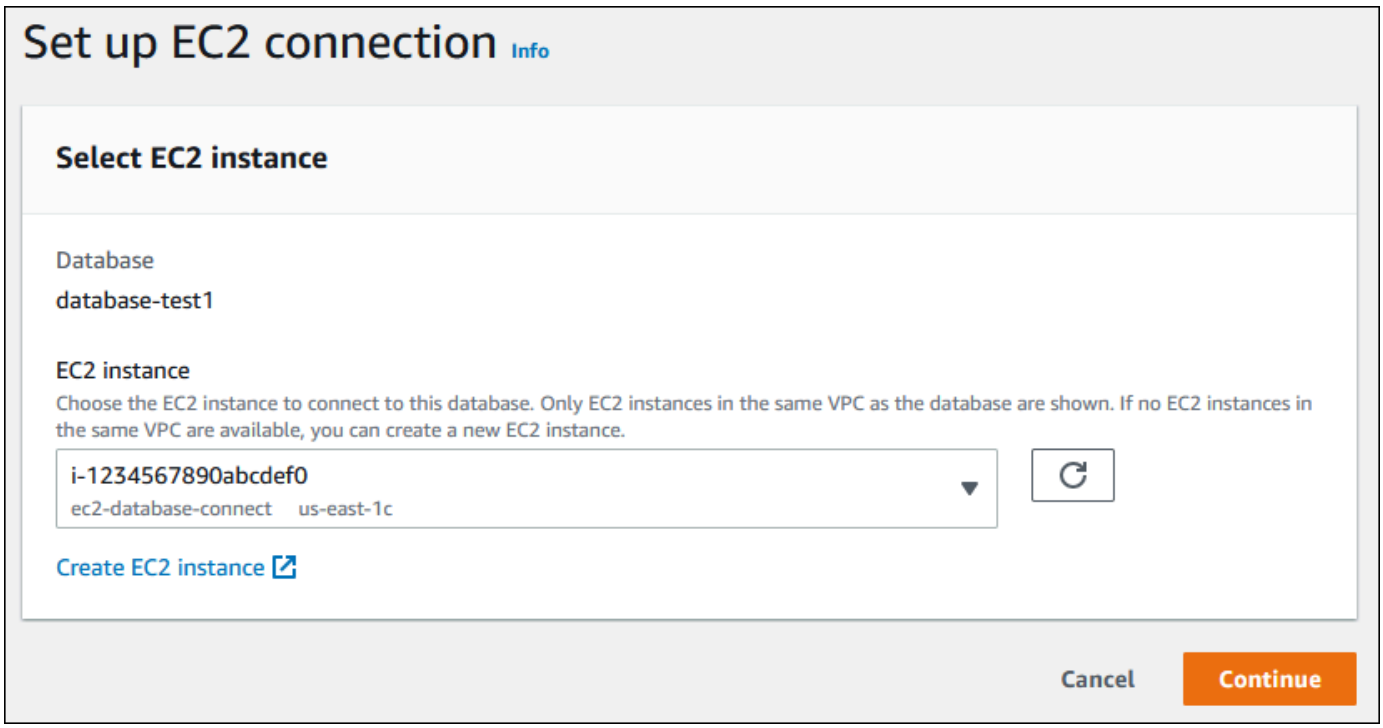
如果您在配置连接后更改安全组，则这些更改可能会影响 EC2 实例与 Aurora 数据库集群之间的连接。

> **Note**
>
> 您只能使用 AWS Management Console自动在 EC2 实例与 Aurora 数据库集群之间设置连接。您无法使用 AWS CLI 或 RDS API 自动设置连接。

自动连接 EC2 实例与 Aurora 数据库集群

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Databases（数据库），然后选择 数据库集群。
3. 从操作中，选择设置 EC2 连接。

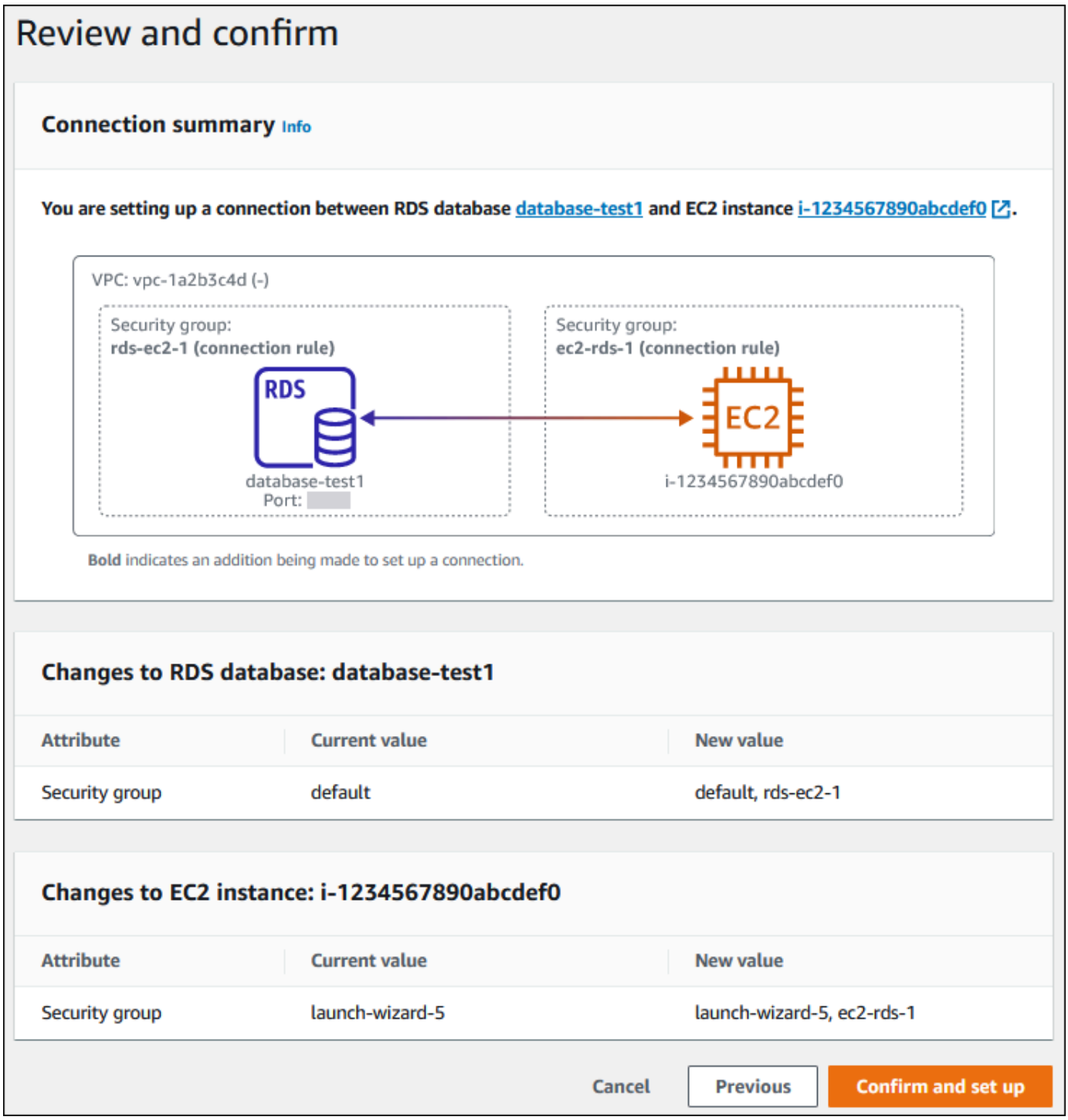
   将出现 Set up EC2 connection（设置 EC2 连接）页面。
4. 在 Set up EC2 connection（设置 EC2 连接）页上，选择 EC2 实例。



如果同一 VPC 中不存在任何 EC2 实例，请选择 Create EC2 instance（创建 EC2 实例）来创建一个此类实例。在这种情况下，请确保新的 EC2 实例与 数据库集群位于同一 VPC 中。

5. 选择继续。

   将出现 Review and confirm（检查并确认）页面。



6. 在 Review and confirm（检查并确认）页面上，检查 RDS 为设置与 EC2 实例的连接而将进行的更改。

   如果更改正确，请选择确认并设置。

   如果更改不正确，请选择 Previous（上一步）或 Cancel（取消）。

查看连接的计算资源

您可以使用 AWS Management Console查看连接到 Aurora 数据库集群的计算资源。显示的资源包括自动设置的计算资源连接。您可以通过以下方式自动设置与计算资源的连接：

- 您可以在创建数据库时选择计算资源。

  有关更多信息，请参阅 创建 Amazon Aurora 数据库集群。
- 您可以在现有数据库和计算资源之间设置连接。

  有关更多信息，请参阅 自动连接 EC2 实例和 Aurora 数据库集群。

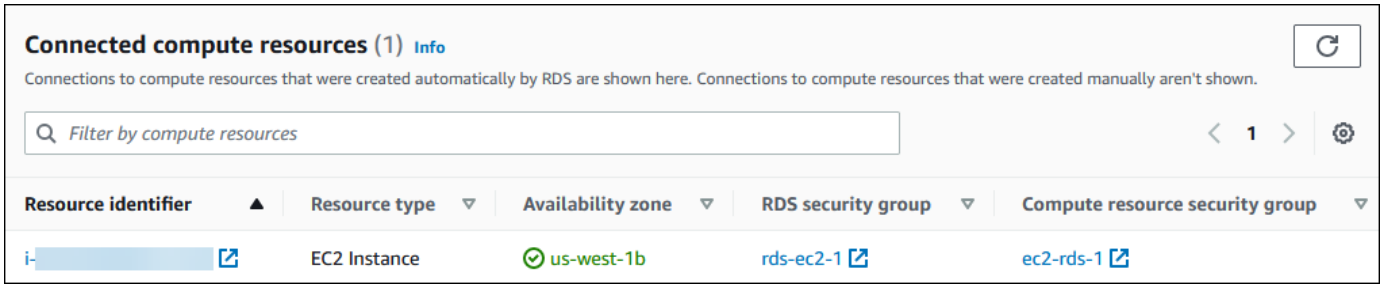
列出的计算资源不包括手动连接到数据库的计算资源。例如，您可以通过向与数据库关联的 VPC 安全组添加规则来允许计算资源手动访问数据库。

要列出计算资源，必须满足以下条件：

- 与计算资源关联的安全组的名称与模式 `ec2-rds-n` 相匹配（其中 *n* 是数字）。
- 与计算资源关联的安全组具有出站规则，其端口范围设置为数据库集群使用的端口。
- 与计算资源关联的安全组具有出站规则，源设置为与 数据库集群关联的安全组。
- 与 数据库集群关联的安全组的名称与模式 `rds-ec2-n`（其中 *n* 是数字）相匹配。
- 与数据库集群关联的安全组具有入站规则，其端口范围设置为数据库集群使用的端口。
- 与 数据库集群关联的安全组有一条入站规则，其源设置为与计算资源关联的安全组。

查看连接到 Aurora 数据库集群的计算资源

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Databases（数据库），然后选择 数据库集群的名称。
3. 在 Connectivity & security（连接和安全）选项卡上，在 Connected compute resources（连接的计算资源）中查看计算资源。



连接到运行特定数据库引擎的数据库实例

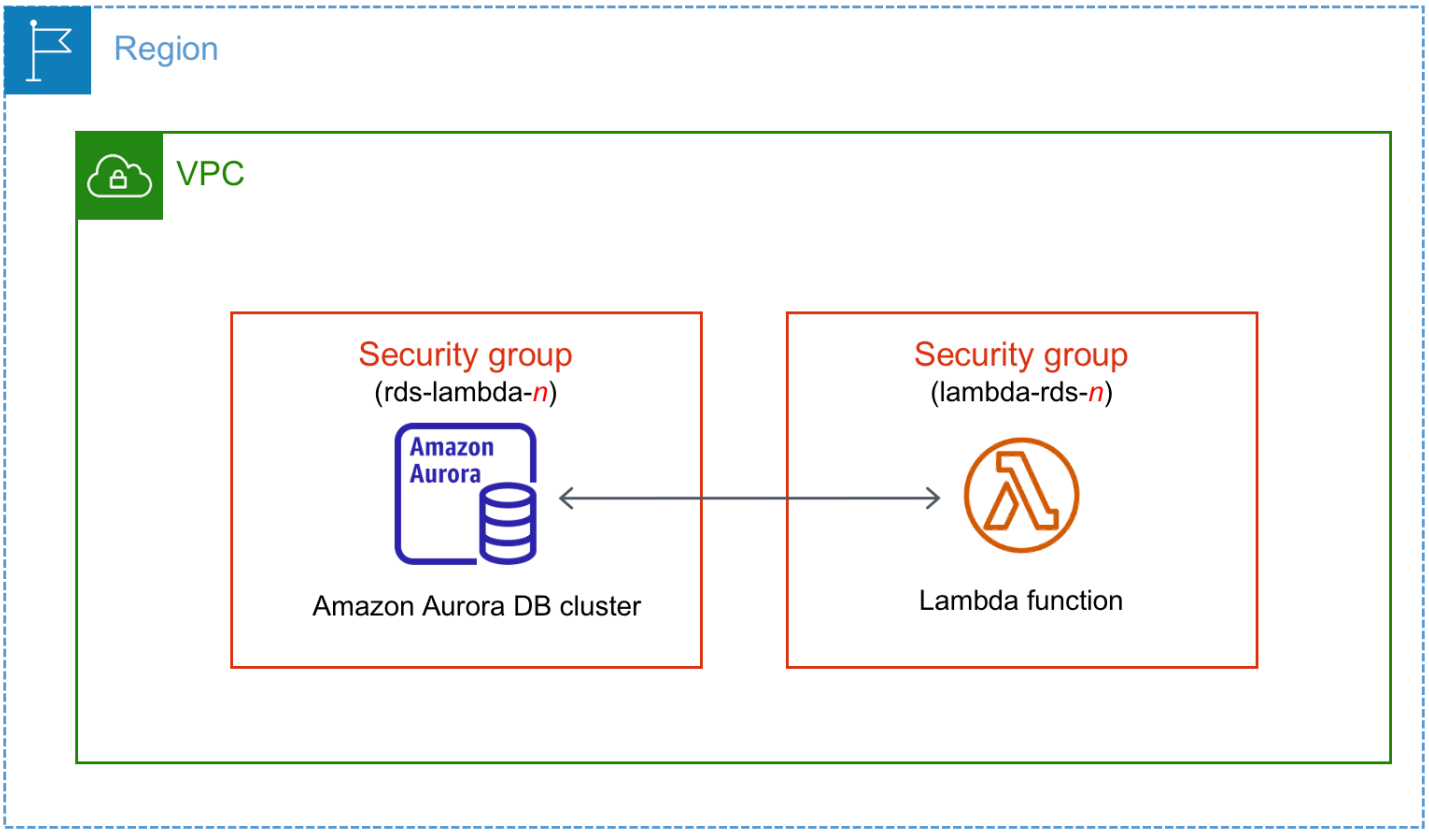
有关连接到运行特定数据库引擎的数据库实例的信息，请按照数据库引擎的说明操作：

- 连接到 Amazon Aurora MySQL 数据库集群
- 连接到 Amazon Aurora PostgreSQL 数据库集群

自动连接 Lambda 函数和 Aurora 数据库集群

您可以使用 Amazon RDS 控制台来简化在 Lambda 函数和 Aurora 数据库集群之间设置连接的过程。通常，您的数据库集群位于 VPC 内的私有子网中。应用程序可以使用 Lambda 函数访问您的私有数据库集群。

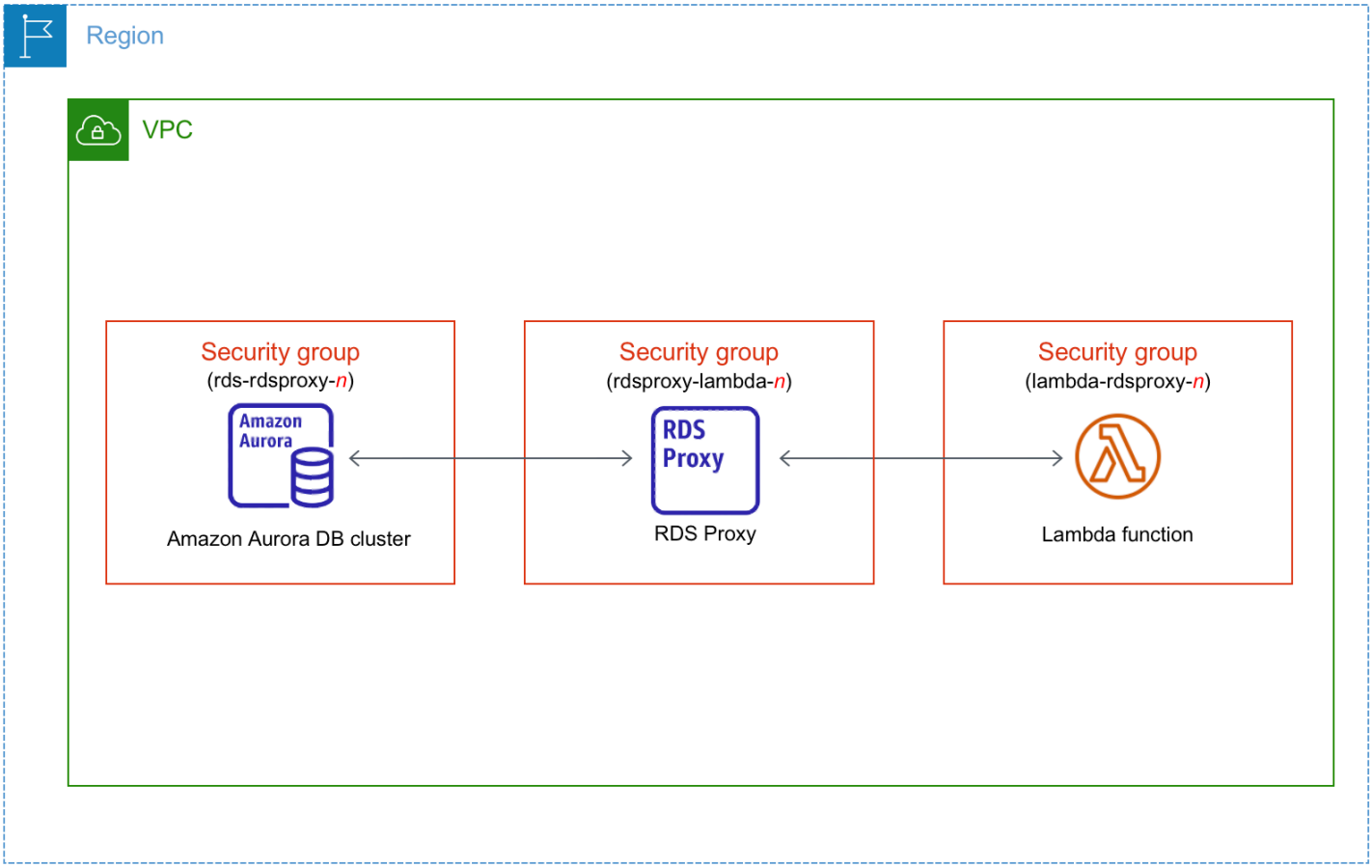
下图显示了数据库集群与您的 Lambda 函数之间的直接连接。



您可以通过 RDS 代理在 Lambda 函数和数据库集群之间设置连接，以提高数据库性能和弹性。通常，Lambda 函数会进行频繁的短期数据库连接，此类连接受益于 RDS 代理提供的连接池。您可以利用已为 Lambda 函数提供的任何 AWS Identity and Access Management（IAM）身份验证，而不是在 Lambda 应用程序代码中管理数据库凭证。有关更多信息，请参阅 适用于 Aurora 的 Amazon RDS 代理。

当您使用控制台通过现有代理进行连接时，Amazon RDS 会更新代理安全组，以允许来自数据库集群和 Lambda 函数的连接。

您也可以从同一个控制台页面创建新的代理。当您在控制台中创建代理时，要访问数据库集群，必须输入数据库凭证或选择 AWS Secrets Manager 密钥。



Tip

要将 Lambda 函数快速连接到Aurora 数据库集群，还可以使用控制台内的引导式向导。要打开该向导，请执行以下操作：

1. 打开 Lamba 控制台的函数页面。
2. 选择要将数据库连接到其中的函数。
3. 在配置选项卡上，选择 RDS 数据库。
4. 选择连接到 RDS 数据库。

将函数连接到数据库后，您可以通过选择添加代理来创建代理。

主题

- 与 Lambda 函数的自动连接概述
- 自动连接 Lambda 函数和 Aurora 数据库集群

- 查看连接的计算资源

与 Lambda 函数的自动连接概述

以下是将 Lambda 函数与 Aurora 数据库集群连接的要求：

- Lambda 函数必须与数据库集群位于同一 VPC 中。
- 目前，数据库集群不能是 Aurora Serverless 数据库集群，也不能是 Aurora Global Database 的一部分。
- 设置连接的用户必须具有执行以下 Amazon RDS、Amazon EC2、Lambda、Secrets Manager 和 IAM 操作的权限：
  - Amazon RDS
    - `rds:CreateDBProxies`
    - `rds:DescribeDBClusters`
    - `rds:DescribeDBProxies`
    - `rds:ModifyDBCluster`
    - `rds:ModifyDBProxy`
    - `rds:RegisterProxyTargets`
  - Amazon EC2
    - `ec2:AuthorizeSecurityGroupEgress`
    - `ec2:AuthorizeSecurityGroupIngress`
    - `ec2:CreateSecurityGroup`
    - `ec2:DeleteSecurityGroup`
    - `ec2:DescribeSecurityGroups`
    - `ec2:RevokeSecurityGroupEgress`
    - `ec2:RevokeSecurityGroupIngress`
  - Lambda
    - `lambda:CreateFunctions`
    - `lambda:ListFunctions`
    - `lambda:UpdateFunctionConfiguration`
  - Secrets Manager
    - `secretsmanager:CreateSecret`

    - `secretsmanager:DescribeSecret`
- IAM
    - `iam:AttachPolicy`
    - `iam:CreateRole`
    - `iam:CreatePolicy`
- AWS KMS
    - `kms:describeKey`

> **Note**
>
> 如果数据库集群与 Lambda 函数位于不同的可用区，您的账户可能会产生跨可用区成本。

当您在 Lambda 函数和 Aurora 数据库集群之间设置连接时，Amazon RDS 会为您的函数和数据库集群配置 VPC 安全组。如果您使用 RDS 代理，则 Amazon RDS 还会为代理配置 VPC 安全组。Amazon RDS 根据与数据库集群、Lambda 函数和代理关联的安全组的当前配置执行操作，如下表所述。

当前 RDS 安全组配置	当前 Lambda 安全组配置	当前代理安全组配置	RDS 操作
有一个或多个安全组与数据库集群关联，其名称与模式 `rds-lambda-` *n* 匹配，或者，如果代理已连接到您的数据库集群，则 RDS 将检查关联代理的 `TargetHealth` 是否为 `AVAILABLE` 。 尚未修改与此模式匹配的安全组。该安全组只具有一条以	有一个或多个安全组与 Lambda 函数关联，此函数的名称与模式 `lambda-rds-` *n* 或 `lambda-rdsproxy-` *n*（其中 *n* 是数字）相匹配。 尚未修改与此模式匹配的安全组。此安全组只有一条出站规则，此规则以数据库集群或代理的 VPC 安全组作为目标。	有一个或多个安全组与代理关联，此代理的名称与模式 `rdsproxy-lambda-` *n*（其中 *n* 是数字）相匹配。 尚未修改与此模式匹配的安全组。此安全组具有入站和出站规则，这些规则具有 Lambda 函数和数据库集群的 VPC 安全组。	Amazon RDS 不执行任何操作。 已在 Lambda 函数、代理（可选）和数据库集群之间自动配置了连接。由于函数、代理和数据库之间已经存在连接，因此不会修改安全组。

当前 RDS 安全组配置	当前 Lambda 安全组配置	当前代理安全组配置	RDS 操作
Lambda 函数或代理的 VPC 安全组作为源的入站规则。			

当前 RDS 安全组配置	当前 Lambda 安全组配置	当前代理安全组配置	RDS 操作
以下任一条件适用： • 没有与数据库集群（名称与模式 `rds-lambda-` *n* 匹配或关联代理的 `TargetHealth` 为 `AVAILABLE`）相关联的安全组。 • 有一个或多个与数据库集群（名称与模式 `rds-lambda-` *n* 匹配或关联代理的 `TargetHealth` 为 `AVAILABLE`）相关联的安全组。但是，这些安全组都不能用于连接 Lambda 函数。 如果安全组没有一条以 Lambda 函数或代理的 VPC 安全组作为源的入站规则，则 Amazon RDS 无法使用该安全组。Amazon RDS 也无法使用经过修改的安全组。修改示例包括添加规则或更改现有规则的端口。	以下任一条件适用： • 没有与 Lambda 函数（名称与模式 `lambda-rds-` *n* 或 `lambda-rdsproxy-` *n* 匹配）关联的安全组。 • 有一个或多个安全组与 Lambda 函数关联，此函数的名称与模式 `lambda-rds-` *n* 或 `lambda-rdsproxy-` *n* 相匹配。但是，Amazon RDS 不能使用这些安全组中的任何一个来连接数据库集群。 如果安全组没有一条以数据库集群或代理的 VPC 安全组作为目标的出站规则，则 Amazon RDS 无法使用该安全组。Amazon RDS 也无法使用经过修改的安全组。	以下任一条件适用： • 没有与代理（名称与模式 `rdsproxy-lambda-` *n* 匹配）关联的安全组。 • 有一个或多个安全组与代理关联，此代理的名称与 `rdsproxy-lambda-` *n* 相匹配。但是，Amazon RDS 不能使用这些安全组中的任何一个来连接数据库集群或 Lambda 函数。 如果安全组没有入站和出站规则（这些规则具有数据库集群和 Lambda 函数的 VPC 安全组），则 Amazon RDS 无法使用该安全组。Amazon RDS 也无法使用经过修改的安全组。	RDS action: create new security groups

当前 RDS 安全组配置	当前 Lambda 安全组配置	当前代理安全组配置	RDS 操作
有一个或多个与数据库集群（名称与模式 `rds-lambda-` *n* 匹配或关联代理的 `TargetHealth` 为 `AVAILABLE`）相关联的安全组。 尚未修改与此模式匹配的安全组。该安全组只具有一条以 Lambda 函数或代理的 VPC 安全组作为源的入站规则。	有一个或多个安全组与 Lambda 函数关联，此函数的名称与模式 `lambda-rds-` *n* 或 `lambda-rdsproxy-` *n* 相匹配。 但是，Amazon RDS 不能使用这些安全组中的任何一个来连接数据库集群。如果安全组没有一条以数据库集群或代理的 VPC 安全组作为目标的出站规则，则 Amazon RDS 无法使用该安全组。Amazon RDS 也无法使用经过修改的安全组。	有一个或多个安全组与代理关联，此代理的名称与模式 `rdsproxy-lambda-` *n* 相匹配。 但是，Amazon RDS 不能使用这些安全组中的任何一个来连接数据库集群或 Lambda 函数。如果安全组没有入站和出站规则（这些规则具有数据库集群和 Lambda 函数的 VPC 安全组），则 Amazon RDS 无法使用该安全组。Amazon RDS 也无法使用经过修改的安全组。	RDS action: create new security groups
有一个或多个与数据库集群（名称与模式 `rds-lambda-` *n* 匹配或关联代理的 `TargetHealth` 为 `AVAILABLE`）相关联的安全组。 尚未修改与此模式匹配的安全组。该安全组只具有一条以 Lambda 函数或代理的 VPC 安全组作为源的入站规则。	存在用于连接的有效 Lambda 安全组，但它与 Lambda 函数不关联。此安全组的名称与模式 `lambda-rds-` *n* 或 `lambda-rdsproxy-` *n* 相匹配。尚未修改它。它只有一条出站规则，此规则以数据库集群或代理的 VPC 安全组作为目标。	存在用于连接的有效代理安全组，但它与代理不关联。此安全组的名称与模式 `rdsproxy-lambda-` *n* 相匹配。尚未修改它。它具有入站和出站规则，这些规则具有数据库集群和 Lambda 函数的 VPC 安全组。	RDS action: associate Lambda security group

当前 RDS 安全组配置	当前 Lambda 安全组配置	当前代理安全组配置	RDS 操作
以下任一条件适用： • 没有与数据库集群（名称与模式 `rds-lambda-` *n* 匹配或关联代理的 `TargetHealth` 为 `AVAILABLE`）相关联的安全组。 • 有一个或多个与数据库集群（名称与模式 `rds-lambda-` *n* 匹配或关联代理的 `TargetHealth` 为 `AVAILABLE`）相关联的安全组。但是，Amazon RDS 不能使用这些安全组中的任何一个来连接 Lambda 函数或代理。 如果安全组没有一条以 Lambda 函数或代理的 VPC 安全组作为源的入站规则，则 Amazon RDS 无法使用该安全组。Amazon RDS 也无法使用经过修改的安全组。	有一个或多个安全组与 Lambda 函数关联，此函数的名称与模式 `lambda-rds-` *n* 或 `lambda-rdsproxy-` *n* 相匹配。 尚未修改与此模式匹配的安全组。此安全组只有一条出站规则，此规则以数据库集群或代理的 VPC 安全组作为目标。	有一个或多个安全组与代理关联，此代理的名称与模式 `rdsproxy-lambda-` *n* 相匹配。 尚未修改与此模式匹配的安全组。此安全组具有入站和出站规则，这些规则具有数据库集群和 Lambda 函数的 VPC 安全组。	RDS action: create new security groups

当前 RDS 安全组配置	当前 Lambda 安全组配置	当前代理安全组配置	RDS 操作
以下任一条件适用： • 没有与数据库集群（名称与模式 `rds-lambda-` *n* 匹配或关联代理的 `TargetHealth` 为 `AVAILABLE` ）相关联的安全组。 • 有一个或多个与数据库集群（名称与模式 `rds-lambda-` *n* 匹配或关联代理的 `TargetHealth` 为 `AVAILABLE` ）相关联的安全组。但是，Amazon RDS 不能使用这些安全组中的任何一个来连接 Lambda 函数或代理。 如果安全组没有一条以 Lambda 函数或代理的 VPC 安全组作为源的入站规则，则 Amazon RDS 无法使用该安全组。Amazon RDS 也无法使用经过修改的安全组。	以下任一条件适用： • 没有与 Lambda 函数（名称与模式 `lambda-rds-` *n* 或 `lambda-rdsproxy-` *n* 匹配）关联的安全组。 • 有一个或多个安全组与 Lambda 函数关联，此函数的名称与模式 `lambda-rds-` *n* 或 `lambda-rdsproxy-` *n* 相匹配。但是，Amazon RDS 不能使用这些安全组中的任何一个来连接数据库集群。 如果安全组没有一条以数据库集群或代理的 VPC 安全组作为源的出站规则，则 Amazon RDS 无法使用该安全组。Amazon RDS 也无法使用经过修改的安全组。	以下任一条件适用： • 没有与代理（名称与模式 `rdsproxy-lambda-` *n* 匹配）关联的安全组。 • 有一个或多个安全组与代理关联，此代理的名称与 `rdsproxy-lambda-` *n* 相匹配。但是，Amazon RDS 不能使用这些安全组中的任何一个来连接数据库集群或 Lambda 函数。 如果安全组没有入站和出站规则（这些规则具有数据库集群和 Lambda 函数的 VPC 安全组），则 Amazon RDS 无法使用该安全组。Amazon RDS 也无法使用经过修改的安全组。	RDS action: create new security groups

RDS 操作：创建新的安全组

Amazon RDS 执行以下操作：

- 创建与模式 `rds-lambda-`*n* 或 `rds-rdsproxy-`*n* 匹配的新安全组（如果您选择使用 RDS 代理）。该安全组具有一条以 Lambda 函数或代理的 VPC 安全组作为源的入站规则。该安全组与数据库集群相关联，并允许函数或代理访问数据库集群。
- 创建与模式 `lambda-rds-`*n* 或 `lambda-rdsproxy-`*n* 匹配的新安全组。此安全组具有一条出站规则，此规则以数据库集群或代理的 VPC 安全组作为目标。此安全组与 Lambda 函数相关联，并允许该函数将流量发送到数据库集群或者通过代理发送流量。
- 创建与模式 `rdsproxy-lambda-`*n* 匹配的新安全组。此安全组具有入站和出站规则，这些规则具有数据库集群和 Lambda 函数的 VPC 安全组。

RDS 操作：关联 Lambda 安全组

Amazon RDS 将有效的现有的 Lambda 安全组与 Lambda 函数关联。此安全组允许该函数将流量发送到数据库集群或者通过代理发送流量。

自动连接 Lambda 函数和 Aurora 数据库集群

您可以使用 Amazon RDS 控制台自动将 Lambda 函数连接到您的数据库集群。这简化了在这些资源之间设置连接的过程。

您也可以使用 RDS 代理在连接中包含代理。Lambda 函数会进行频繁的短期数据库连接，此类连接受益于 RDS 代理提供的连接池。您还可以使用已经为 Lambda 函数设置的任何 IAM 身份验证，而不是在 Lambda 应用程序代码中管理数据库凭证。

您可以使用设置 Lambda 连接页面将现有的数据库集群连接到新的和现有的 Lambda 函数。设置过程会自动为您设置所需的安全组。

在 Lambda 函数和数据库集群之间设置连接之前，请确保：

- 您的 Lambda 函数和数据库集群位于同一 VPC 中。
- 您拥有用户账户的相应权限。有关要求的更多信息，请参阅与 Lambda 函数的自动连接概述。

如果您在配置连接后更改安全组，则这些更改可能会影响 Lambda 函数与数据库集群之间的连接。

> **Note**
>
> 您只能在 AWS Management Console中自动设置数据库集群和 Lambda 函数之间的连接。要连接 Lambda 函数，数据库集群中的所有实例必须处于可用状态。

自动连接 Lambda 函数和数据库集群

<result>

确认设置后，Amazon RDS 开始连接您的 Lambda 函数、RDS 代理（如果您使用了代理）和数据库集群的过程。控制台显示连接详细信息对话框，其中列出了允许在您的资源之间进行连接的安全组更改。
</result>

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择数据库，然后选择要连接到 Lambda 函数的数据库集群。
3. 对于操作，选择设置 Lambda 连接。
4. 在设置 Lambda 连接页面上，在选择 Lambda 函数下方，执行以下任一操作：
   - 如果您在与数据库集群相同的 VPC 中有一个现有的 Lambda 函数，请选择选择现有函数，然后选择该函数。
   - 如果您在同一 VPC 中没有 Lambda 函数，请选择创建新函数，然后输入函数名称。默认运行时系统设置为 Nodejs.18。完成连接设置后，您可以在 Lambda 控制台中修改新 Lambda 函数的设置。
5. （可选）在 RDS 代理下，选择使用 RDS 代理进行连接，然后执行以下任一操作：
   - 如果您有要使用的现有代理，请选择选择现有代理，然后选择此代理。
   - 如果您没有代理，并且希望 Amazon RDS 自动为您创建代理，请选择创建新代理。然后，对于数据库凭证，执行以下任一操作：

     a. 选择数据库用户名和密码，然后为您的数据库集群输入用户名和密码。

     b. 选择 Secrets Manager 密钥。然后，对于选择密钥，选择一个 AWS Secrets Manager 密钥。如果您没有 Secrets Manager 密钥，请选择创建新的 Secrets Manager 密钥以创建新密钥。创建密钥后，对于选择密钥，选择新密钥。

     创建新代理后，选择选择现有代理，然后选择该代理。请注意，您的代理可能需要一些时间才能用于连接。

6. （可选）扩展连接摘要并验证您的资源的突出显示更新。
7. 选择 Set up (设置)。

查看连接的计算资源

您可以使用 AWS Management Console查看连接到数据库集群的 Lambda 函数。显示的资源包括 Amazon RDS 自动设置的计算资源连接。

列出的计算资源不包括手动连接到数据库集群的计算资源。例如，您可以通过向与数据库关联的 VPC 安全组添加规则，来允许计算资源手动访问数据库集群。

要使控制台列出 Lambda 函数，必须满足以下条件：

- 与计算资源关联的安全组的名称与模式 `lambda-rds-`*n* 或 `lambda-rdsproxy-`*n*（其中 *n* 是数字）相匹配。
- 与计算资源关联的安全组具有出站规则，其端口范围设置为数据库集群或关联代理的端口。出站规则的目标必须设置为与数据库集群或关联代理关联的安全组。
- 如果配置包含代理，则连接到与您数据库关联的代理的安全组名称与模式 `rdsproxy-lambda-`*n*（其中 *n* 是一个数字）相匹配。
- 与函数关联的安全组具有出站规则，其端口设置为数据库集群或关联代理使用的端口。目标必须设置为与数据库集群或关联代理关联的安全组。

查看自动连接到数据库集群的计算资源

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择数据库，然后选择数据库实例。
3. 在连接和安全选项卡上，在连接的计算资源下查看计算资源。

修改 Amazon Aurora 数据库集群

您可以更改数据库集群的设置来完成更改其备份保留期或其数据库端口等任务。还可以修改数据库集群中的数据库实例来完成更改其数据库实例类或为其启用 Performance Insights 等任务。本主题指导您修改 Aurora 数据库集群及其数据库实例，并介绍每个实例的设置。

我们建议您在修改生产数据库集群或数据库实例之前在测试数据库集群或数据库实例上测试所有更改，以便完全了解每项更改的影响。在升级数据库版本时，这尤其重要。

主题

- 使用控制台、CLI 和 API 修改数据库集群
- 修改数据库集群中的数据库实例
- 更改数据库主用户的密码
- Amazon Aurora 设置
- 不适用于 Amazon Aurora 数据库集群的设置
- 不适用于 Amazon Aurora 数据库实例的设置

使用控制台、CLI 和 API 修改数据库集群

您可以使用 AWS Management Console、AWS CLI 或 RDS API 修改数据库集群。

> **Note**
>
> 大多数修改可以立即应用，也可以在下一个计划的维护时段内应用。无论您选择何时应用，某些修改（例如开启删除保护）都会立即生效。
> 在 AWS Management Console中更改主密码始终立即生效。但是，在使用 AWS CLI 或 RDS API 时，您可以选择是立即应用此更改，还是在下一个计划维护时段内应用此更改。
> 如果您使用 SSL 端点并更改数据库集群标识符，请停止并重新启动数据库集群以更新 SSL 端点。有关更多信息，请参阅 停止和启动 Amazon Aurora 数据库集群。

控制台

修改数据库集群

1. 登录AWS Management Console并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。

2. 在导航窗格中，选择 Databases (数据库)，然后选择您希望修改的数据库集群。
3. 选择修改。此时会显示修改数据库集群页面。
4. 根据需要更改任意设置。有关每项设置的信息，请参阅 Amazon Aurora 设置。

> **Note**
>
> 在 AWS Management Console中，某些实例级别更改仅适用于当前数据库实例，而其他更改适用于整个数据库集群。有关设置是适用于数据库实例还是数据库集群的信息，请参阅Amazon Aurora 设置中的设置范围。要在 AWS Management Console中更改在实例级别修改整个数据库集群的设置，请按照修改数据库集群中的数据库实例中的说明操作。

5. 当所有更改都达到您的要求时，选择继续并查看修改摘要。
6. 要立即应用更改，请选择立即应用。
7. 在确认页面上，检查您的更改。如果更改正确无误，请选择修改集群以保存更改。

   或者，选择 Back 编辑您的更改，或者选择 Cancel 取消更改。

AWS CLI

要使用 AWS CLI 修改数据库集群，请调用 modify-db-cluster 命令。指定数据库集群标识符以及要修改的设置值。有关每项设置的信息，请参阅 Amazon Aurora 设置。

> **Note**
>
> 某些设置仅适用于数据库实例。要更改这些设置，请按照修改数据库集群中的数据库实例中的说明操作。

Example

以下命令将备份保留期设置为 1 周（7 天）以修改 `mydbcluster`。

对于 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --backup-retention-period 7
```

对于 Windows：

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --backup-retention-period 7
```

RDS API

要使用 Amazon RDS API 修改数据库集群，请调用 ModifyDBCluster 操作。指定数据库集群标识符以及要修改的设置值。有关每个参数的信息，请参阅Amazon Aurora 设置。

> **Note**
>
> 某些设置仅适用于数据库实例。要更改这些设置，请按照修改数据库集群中的数据库实例中的说明操作。

修改数据库集群中的数据库实例

您可以使用 AWS Management Console、AWS CLI 或 RDS API 修改数据库集群中的数据库实例。

修改数据库实例后可以立即应用更改。要立即应用更改，可以在 AWS Management Console 中选择 Apply Immediately（立即应用）选项，还可以在调用 AWS CLI 时使用 `--apply-immediately` 参数，或在使用 Amazon RDS API 时将 `ApplyImmediately` 参数设为 `true`。

如果不选择立即应用更改，则更改将推迟到下一个维护时段。在下一个维护时段中，将应用这些延迟的更改中的任何一个。如果选择立即应用更改，将应用您的新更改以及任何先前延迟的更改。

要查看下一个维护时段待处理的修改，请使用 describe-db-clusters AWS CLI 命令并选中 `PendingModifiedValues` 字段。

> **Important**
>
> 如果任何延迟的修改需要停机，选择 Apply immediately (立即应用) 可能会对该数据库实例导致意外停机。数据库集群中的其他数据库实例则无停机。`describe-pending-maintenance-actions` CLI 命令的输出中未列出您推迟的修改。维护操作仅包括您计划在下一个维护时段进行的系统升级。

控制台

修改数据库集群中的数据库实例

1. 登录AWS Management Console并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择数据库，然后选择您希望修改的数据库实例。
3. 对于 Actions (操作)，选择 Modify (修改)。将显示修改数据库实例页面。
4. 根据需要更改任意设置。有关每项设置的信息，请参阅 Amazon Aurora 设置。

   > **Note**
   >
   > 某些设置应用于整个数据库集群，并且必须在集群级别进行更改。要更改这些设置，请按照使用控制台、CLI 和 API 修改数据库集群中的说明操作。
   > 在 AWS Management Console中，某些实例级别更改仅适用于当前数据库实例，而其他更改适用于整个数据库集群。有关设置是适用于数据库实例还是数据库集群的信息，请参阅Amazon Aurora 设置中的设置范围。

5. 当所有更改都达到您的要求时，选择继续并查看修改摘要。
6. 要立即应用更改，请选择立即应用。
7. 在确认页面上，检查您的更改。如果更改正确无误，请选择 Modify DB Instance（修改数据库实例）保存更改。

   或者，选择 Back 编辑您的更改，或者选择 Cancel 取消更改。

AWS CLI

要使用 AWS CLI 修改数据库集群中的数据库实例，请调用 modify-db-instance 命令。指定数据库实例标识符以及要修改的设置值。有关每个参数的信息，请参阅Amazon Aurora 设置。

> **Note**
>
> 某些设置适用于整个数据库集群。要更改这些设置，请按照使用控制台、CLI 和 API 修改数据库集群中的说明操作。

Example

以下代码将数据库实例类设置为 mydbinstance 以修改 db.r4.xlarge。将在下一维护时段使用 --no-apply-immediately 应用这些更改。使用 --apply-immediately 可立即应用更改。

对于 Linux、macOS 或 Unix：

```
aws rds modify-db-instance \
    --db-instance-identifier mydbinstance \
    --db-instance-class db.r4.xlarge \
    --no-apply-immediately
```

对于 Windows：

```
aws rds modify-db-instance ^
    --db-instance-identifier mydbinstance ^
    --db-instance-class db.r4.xlarge ^
    --no-apply-immediately
```

RDS API

要使用 Amazon RDS API 修改数据库实例，请调用 ModifyDBInstance 操作。指定数据库实例标识符以及要修改的设置值。有关每个参数的信息，请参阅Amazon Aurora 设置。

> **Note**
>
> 某些设置适用于整个数据库集群。要更改这些设置，请按照使用控制台、CLI 和 API 修改数据库集群中的说明操作。

更改数据库主用户的密码

您可以使用 AWS Management Console或 AWS CLI 更改主用户密码。

控制台

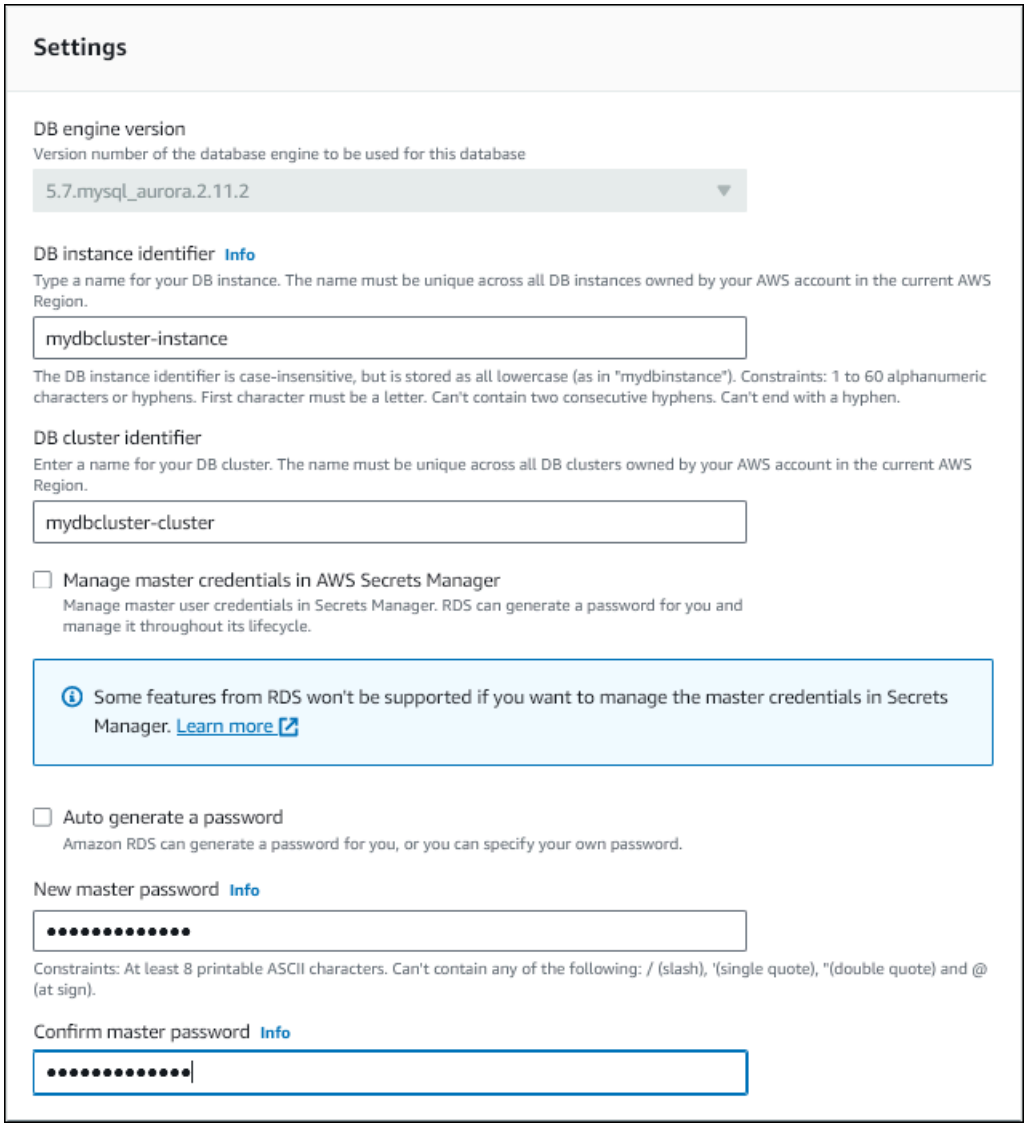
您可以使用 AWS Management Console修改写入器数据库实例以更改主用户密码。

更改主用户密码

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。

2. 在导航窗格中，选择数据库，然后选择您希望修改的数据库实例。
3. 对于 Actions (操作)，选择 Modify (修改)。

   将显示修改数据库实例页面。
4. 输入新的主密码。
5. 在确认主密码中，输入相同的新密码。



6. 选择继续，查看修改摘要。

> **Note**
>
> 密码更改将始终立即生效。

7. 在确认页面上，选择修改数据库实例。

CLI

您可以通过 AWS CLI 调用 modify-db-cluster 命令来更改主用户密码。指定数据库集群标识符和新密码，如以下示例中所示。

您无需指定 `--apply-immediately|--no-apply-immediately`，因为密码更改始终会立即生效。

对于 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --master-user-password mynewpassword
```

对于 Windows：

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --master-user-password mynewpassword
```

Amazon Aurora 设置

下表包含有关可修改的设置、修改设置的方法以及设置范围的详细信息。范围确定设置是适用于整个数据库集群，还是只能为特定数据库实例设定该设置。

> **Note**
>
> 如果要修改 Aurora Serverless v1 或 Aurora Serverless v2 数据库集群，则可以使用其他设置。有关这些设置的更多信息，请参阅修改 Aurora Serverless v1 数据库集群和管理 Aurora Serverless v2 数据库集群。
> 由于相关限制，某些设置不可用于 Aurora Serverless v1 和 Aurora Serverless v2。有关更多信息，请参阅Aurora Serverless v1 的限制 和Aurora Serverless v2 的要求和限制。

设置和说明	方法	范围	停机说明
自动次要版本升级 是否希望数据库实例在首选次要引擎版本	**Note** 默认情况下，此设置处于启	整个数据库集群	在此更改期间，不会出现中断。如果 Aurora 应用自动升

设置和说明	方法	范围	停机说明
升级可用时自动接收该升级。只有在计划的维护时段内，才会安装升级。 有关引擎更新的更多信息，请参阅 Amazon Aurora PostgreSQL 的数据库引擎更新和 Amazon Aurora MySQL 的数据库引擎更新。有关 Aurora MySQL 的自动次要版本升级设置的更多信息，请参阅 启用 Aurora MySQL 次要版本之间的自动升级。	用状态。对于每个新集群，根据其重要性、预期生命周期以及每次升级后执行的验证测试量为此设置选择适当的值。 更改此设置时，请对 Aurora 集群中的每个数据库实例执行此修改。如果对集群中的任何数据库实例禁用了此设置，则不会自动升级集群。 通过使用 AWS Management Console，修改数据库集群中的数据库实例。 使用 AWS CLI，运行 modify-db-instance 并设置 `--auto-minor-version-upgrade|--no-auto-minor-version-upgrade` 选项。 使用 RDS API，调用 ModifyDBInstance		级，则会在未来的维护时段内出现中断。

设置和说明	方法	范围	停机说明
	并设置 `AutoMinorVersionUpgrade` 参数。		
备份保留期 自动备份的保留天数。最小值为 1。 有关更多信息，请参阅“备份”。	通过使用 AWS Management Console，使用控制台、CLI 和 API 修改数据库集群。 使用 AWS CLI，运行 modify-db-cluster 并设置 `--backup-retention-period` 选项。 使用 RDS API，调用 ModifyDBCluster 并设置 `BackupRetentionPeriod` 参数。	整个数据库集群	在此更改期间，不会出现中断。

设置和说明	方法	范围	停机说明
备份时段（开始时间） 发生数据库自动备份的时间范围。备份时段是开始时间 (采用通用协调时间 (UTC)) 和持续时间 (以小时为单位)。 Aurora 备份是连续的增量备份，但备份窗口用于创建在备份保留期内保留的每日系统备份。您可以复制它以便在保留期之外保留。 数据库集群的维护时段不能与备份时段重叠。 有关更多信息，请参阅“备份时段”。	通过使用 AWS Management Console，使用控制台、CLI 和 API 修改数据库集群。 使用 AWS CLI，运行 modify-db-cluster 并设置 `--preferred-backup-window` 选项。 使用 RDS API，调用 ModifyDBCluster 并设置 `PreferredBackupWindow` 参数。	整个数据库集群。	在此更改期间，不会出现中断。

设置和说明	方法	范围	停机说明
容量设置 Aurora Serverless v1 数据库集群的扩展属性。您只能在 `serverless` 数据库引擎模式下修改数据库集群的扩展属性。 有关 Aurora Serverless v1 的信息，请参阅使用 Amazon Aurora Serverless v1。	通过使用 AWS Management Console，使用控制台、CLI 和 API 修改数据库集群。 使用 AWS CLI，运行 modify-db-cluster 并设置 `--scaling-configuration` 选项。 使用 RDS API，调用 ModifyDBCluster 并设置 `ScalingConfiguration` 参数。	整个数据库集群	在此更改期间，不会出现中断。 更改立即发生。此设置忽略立即应用设置。
证书颁发机构 数据库实例使用的服务器证书的证书颁发机构（CA）。	通过使用 AWS Management Console，修改数据库集群中的数据库实例。 使用 AWS CLI，运行 modify-db-instance 并设置 `--ca-certificate-identifier` 选项。 使用 RDS API，调用 ModifyDBInstance 并设置 `CACertificateIdentifier` 参数。	仅指定的数据库实例	仅当数据库引擎不支持在不重新启动的情况下轮换时，才会发生中断。您可以使用 describe-db-engine-versions AWS CLI 命令来确定数据库引擎是否支持在不重新启动的情况下轮换。

设置和说明	方法	范围	停机说明
集群存储配置 数据库集群的存储类型：Aurora I/O-Optimized 或 Aurora Standard。 有关更多信息，请参阅 Amazon Aurora 数据库集群的存储配置。	通过使用 AWS Management Console，使用控制台、CLI 和 API 修改数据库集群。 使用 AWS CLI，运行 modify-db-cluster 并设置 `--storage-type` 选项。 使用 RDS API，调用 ModifyDBCluster 并设置 `StorageType` 参数。	整个数据库集群	对于采用优化型读取功能实例类的 Aurora PostgreSQL 数据库集群，更改其存储类型会导致中断。而对于采用其他实例类类型的集群，在更改存储类型时不会导致中断。有关数据库实例类类型的更多信息，请参阅数据库实例类类型。

设置和说明	方法	范围	停机说明
将标签复制到快照 选择该项可指定将为该数据库集群定义的标签复制到从该数据库集群创建的数据库快照。有关更多信息，请参阅“为 Amazon Aurora 和 Amazon RDS 资源添加标签”。	通过使用 AWS Management Console，使用控制台、CLI 和 API 修改数据库集群。 通过使用 AWS CLI 来运行 https://docs.aws.amazon.com/cli/latest/reference/rds/modify-db-cluster.html 并设置 `--copy-tags-to-snapshot` 或 `--no-copy-tags-to-snapshot` 选项。 通过使用 RDS API 来调用 https://docs.aws.amazon.com/AmazonRDS/latest/APIReference/API_ModifyDBCluster.html 并设置 `CopyTagsToSnapshot` 参数。	整个数据库集群	在此更改期间，不会出现中断。

设置和说明	方法	范围	停机说明
数据 API 您可以使用基于 Web 服务的应用程序（包括 AWS Lambda 和 AWS AppSync）访问 Aurora Serverless v1。 此设置仅适用于 Aurora Serverless v1 数据库集群。 有关更多信息，请参阅“使用 Amazon RDS 数据 API”。	通过使用 AWS Management Console，使用控制台、CLI 和 API 修改数据库集群。 通过使用 AWS CLI 来运行 `modify-db-cluster` 并设置 `--enable-http-endpoint` 选项。 通过使用 RDS API 来调用 `ModifyDBCluster` 并设置 `EnableHttpEndpoint` 参数。	整个数据库集群	在此更改期间，不会出现中断。

设置和说明	方法	范围	停机说明
数据库身份验证 您想要使用的数据库身份验证。 适用于 MySQL： • 选择密码身份验证以仅使用数据库密码验证数据库用户的身份。 • 选择密码和 IAM 数据库身份验证以使用 IAM 用户和角色的数据库密码和用户凭证验证数据库用户的身份。有关更多信息，请参阅“IAM 数据库身份验证”。 适用于 PostgreSQL： • 选择 IAM database authentication（IAM 数据库身份验证），以通过用户和角色使用数据库密码和用户凭证验证数据库用户的身份。有关更多信息，请参阅 IAM 数据库身份验证。	通过使用 AWS Management Console，使用控制台、CLI 和 API 修改数据库集群。 使用 AWS CLI，运行 modify-db-cluster 并设置以下选项： • 对于 IAM 身份验证，请设置 `--enable-iam-database-authentication|--no-enable-iam-database-authentication` 选项。 • 对于 Kerberos 身份验证，请设置 `--domain` 和 `--domain-iam-role-name` 选项。 使用 RDS API，调用 ModifyDBCluster 并设置以下参数： • 对于 IAM 身份验证，请设置 `EnableIAMDatabaseA`	整个数据库集群	在此更改期间，不会出现中断。

设置和说明	方法	范围	停机说明
• 选择 Kerberos 身份验证以使用 Kerberos 身份验证来验证数据库密码和用户凭证。有关更多信息，请参阅“在 Aurora PostgreSQL 中使用 Kerberos 身份验证”。	uthentica tion 参数。 • 对于 Kerberos 身份验证，请设置 Domain 和 DomainIAM RoleName 参数。		
数据库端口 要用于访问数据库集群的端口。	通过使用 AWS Management Console，使用控制台、CLI 和 API 修改数据库集群。 通过使用 AWS CLI 来运行 modify-db-cluster 并设置 --port 选项。 通过使用 RDS API 来调用 ModifyDBCluster 并设置 Port 参数。	整个数据库集群	在此更改期间发生服务中断。将立即重新引导数据库集群中的所有数据库实例。

设置和说明	方法	范围	停机说明
数据库集群标识符 数据库集群标识符。此值以一个小写字符串存储。 更改数据库集群标识符后，数据库集群端点发生变化。数据库集群中的数据库实例的端点不会发生变化。	通过使用 AWS Management Console，使用控制台、CLI 和 API 修改数据库集群。 通过使用 AWS CLI 来运行 `modify-db-cluster` 并设置 `--new-db-cluster-identifier` 选项。 通过使用 RDS API 来调用 `ModifyDBCluster` 并设置 `NewDBClusterIdentifier` 参数。	整个数据库集群	在此更改期间，不会出现中断。

设置和说明	方法	范围	停机说明
数据库集群参数组 要与数据库集群关联的数据库集群参数组。 有关更多信息，请参阅“Amazon Aurora 的参数组”。	通过使用 AWS Management Console，使用控制台、CLI 和 API 修改数据库集群。 通过使用 AWS CLI 来运行 `modify-db-cluster` 并设置 `--db-cluster-parameter-group-name` 选项。 通过使用 RDS API 来调用 `ModifyDBCluster` 并设置 `DBClusterParameterGroupName` 参数。	整个数据库集群	在此更改期间，不会出现中断。当您更改参数组时，对某些参数的更改将立即应用于数据库集群中的数据库实例，而无需重新启动。而对其他一些参数进行的更改，只有在重新启动数据库集群中的数据库实例之后，才会应用。

设置和说明	方法	范围	停机说明
数据库实例类 您要使用的数据库实例类。 有关更多信息，请参阅“Amazon Aurora 数据库实例类”。	通过使用 AWS Management Console，修改数据库集群中的数据库实例。 通过使用 AWS CLI 来运行 `modify-db-instance` 并设置 `--db-instance-class` 选项。 通过使用 RDS API 来调用 `ModifyDBInstance` 并设置 `DBInstanceClass` 参数。	仅指定的数据库实例	在此更改期间发生服务中断。

设置和说明	方法	范围	停机说明
数据库实例标识符 数据库实例标识符。此值以一个小写字符串存储。	通过使用 AWS Management Console，修改数据库集群中的数据库实例。 通过使用 AWS CLI 来运行 `modify-db-instance` 并设置 `--new-db-instance-identifier` 选项。 通过使用 RDS API 来调用 `ModifyDBInstance` 并设置 `NewDBInstanceIdentifier` 参数。	仅指定的数据库实例	在此更改期间发生停机。 RDS 会重新启动数据库实例来更新以下项： • Aurora MySQL – `information_schema.replica_host_status` 表中的 `SERVER_ID` 列 • Aurora PostgreSQL – `aurora_replica_status()` 函数中的 `server_id` 列

设置和说明	方法	范围	停机说明
数据库参数组 要与数据库实例关联的数据库参数组。 有关更多信息，请参阅“Amazon Aurora 的参数组”。	通过使用 AWS Management Console，修改数据库集群中的数据库实例。 通过使用 AWS CLI 来运行 `modify-db-instance` 并设置 `--db-parameter-group-name` 选项。 通过使用 RDS API 来调用 `ModifyDBInstance` 并设置 `DBParameterGroupName` 参数。	仅指定的数据库实例	在此更改期间，不会出现中断。 将新数据库参数组与数据库实例关联时，修改后的静态和动态参数仅在数据库实例重新启动后得到应用。但是，如果在将数据库参数组与数据库实例关联之后修改数据库参数组中的动态参数，这些更改将立即得到应用，而无需重启。 有关更多信息，请参阅Amazon Aurora 的参数组 和重启 Amazon Aurora 数据库集群或 Amazon Aurora 数据库实例。

设置和说明	方法	范围	停机说明
删除保护 启用删除保护以禁止删除数据库集群。有关更多信息，请参阅“Aurora 集群的删除保护”。	通过使用 AWS Management Console，使用控制台、CLI 和 API 修改数据库集群。 通过使用 AWS CLI 来运行 `modify-db-cluster` 并设置 `--deletion-protection|--no-deletion-protection` 选项。 通过使用 RDS API 来调用 `ModifyDBCluster` 并设置 `DeletionProtection` 参数。	整个数据库集群	在此更改期间，不会出现中断。

设置和说明	方法	范围	停机说明
引擎版本 要使用的数据库引擎的版本。在升级生产数据库集群之前，我们建议您在测试数据库集群上测试升级过程以验证其持续时间以及您的应用程序。	通过使用 AWS Management Console，使用控制台、CLI 和 API 修改数据库集群。 通过使用 AWS CLI 来运行 `modify-db-cluster` 并设置 `--engine-version` 选项。 通过使用 RDS API 来调用 `ModifyDBCluster` 并设置 `EngineVersion` 参数。	整个数据库集群	在此更改期间发生服务中断。

设置和说明	方法	范围	停机说明
增强监控 启用增强监控以允许为运行数据库实例的操作系统实时收集指标。 有关更多信息，请参阅“使用增强监控来监控操作系统指标”。	通过使用 AWS Management Console，修改数据库集群中的数据库实例。 通过使用 AWS CLI 来运行 `modify-db-instance` 并设置 `--monitoring-role-arn` 和 `--monitoring-interval` 选项。 通过使用 RDS API 来调用 `ModifyDBInstance` 并设置 `MonitoringRoleArn` 和 `MonitoringInterval` 参数。	仅指定的数据库实例	在此更改期间，不会出现中断。

设置和说明	方法	范围	停机说明
日志导出 选择要发布到 Amazon CloudWatch Logs 的日志类型。 有关更多信息，请参阅“Aurora MySQL 数据库日志文件”。	通过使用 AWS Management Console，使用控制台、CLI 和 API 修改数据库集群。 通过使用 AWS CLI 来运行 `modify-db-cluster` 并设置 `--cloudwatch-logs-export-configuration` 选项。 通过使用 RDS API 来调用 `ModifyDBCluster` 并设置 `CloudwatchLogsExportConfiguration` 参数。	整个数据库集群	在此更改期间，不会出现中断。

设置和说明	方法	范围	停机说明
维护窗口 进行系统维护的时间范围。维护系统包括升级 (如果适用)。维护时段是开始时间 (采用通用协调时间 (UTC)) 和持续时间 (以小时为单位)。 如果将时段设置为当前时间，则当前时间与该时段结束之间必须相隔至少 30 分钟以确保应用所有待处理的更改。 您可以为数据库集群以及数据库集群中的每个数据库实例单独设置维护时段。在修改范围是整个数据库集群时，将在数据库集群的维护时段执行修改。在修改范围是数据库实例时，将在该数据库实例的维护时段执行修改。 数据库集群的维护时段不能与备份时段重叠。 有关更多信息，请参阅“Amazon RDS 维护时段”。	要使用 AWS Management Console 更改数据库集群的维护时段，请参阅使用控制台、CLI 和 API 修改数据库集群。 要使用 AWS Management Console 更改数据库实例的维护时段，请参阅修改数据库集群中的数据库实例。 要使用 AWS CLI 更改数据库集群的维护时段，请运行 `modify-db-cluster` 并设置 `--preferred-maintenance-window` 选项。 要使用 AWS CLI 更改数据库实例的维护时段，请运行 `modify-db-instance` 并设置 `--preferred-maintenance-window` 选项。 要使用 RDS API 更改数据库集群的维护时段，请调用 `ModifyDBCluster` 并设置 `Preferred`	整个数据库集群或单个数据库实例	如果有一个或多个待处理的操作导致服务中断，并且维护时段经过更改，加入了当前时间，则立即应用这些待处理的操作并会出现中断。

设置和说明	方法	范围	停机说明
	Maintenan ceWindow 参数。 要使用 RDS API 更改数据库实例的维护时段，请调用 ModifyDBI nstance 并设置 Preferred Maintenan ceWindow 参数。		

设置和说明	方法	范围	停机说明
在 AWS Secrets Manager 中管理主凭证 选择在 AWS Secrets Manager 中管理主凭证，以在 Secrets Manager 的密钥中管理主用户密码。 （可选）选择用于保护密钥的 KMS 密钥。请从您的账户的 KMS 密钥中进行选择，或输入来自其他账户的密钥。 有关更多信息，请参阅 使用 Amazon Aurora 和 AWS Secrets Manager 管理密码。 如果 Aurora 已在管理数据库集群的主用户密码，则可以通过选择 Rotate secret immediately（立即轮换密钥）以轮换主用户密码。 有关更多信息，请参阅 使用 Amazon Aurora 和 AWS Secrets Manager 管理密码。	通过使用 AWS Management Console，修改数据库集群中的数据库实例。 通过使用 AWS CLI 来运行 `modify-db-cluster` 并设置 `--manage-master-user-password | --no-manage-master-user-password` 和 `--master-user-secret-kms-key-id` 选项。要立即轮换主用户密码，请设置 `--rotate-master-user-password` 选项。 通过使用 RDS API 来调用 `ModifyDBCluster` 并设置 `ManageMasterUserPassword` 和 `MasterUserSecretKmsKeyId` 参数。要立即轮换主用户密码，请将 `RotateMasterUserPa`	整个数据库集群	在此更改期间，不会出现中断。

设置和说明	方法	范围	停机说明
	ssword 参数设置为 true。		
网络类型 数据库集群支持的 IP 寻址协议。 IPv4，指定资源只能通过 IPv4 寻址协议与数据库集群通信。 Dual-stack mode（双堆栈模式），指定资源可通过 IPv4 和/或 IPv6 与数据库集群通信。如果您有任何必须通过 IPv6 寻址协议与数据库集群通信的资源，请使用双堆栈模式。要使用双堆栈模式，请确保至少有两个子网跨越两个同时支持 IPv4 和 IPv6 网络协议的可用区。此外，请确保将 IPv6 CIDR 块与您指定的数据库子网组中的子网进行关联。 有关更多信息，请参阅 Amazon Aurora IP 寻址。	通过使用 AWS Management Console，使用控制台、CLI 和 API 修改数据库集群。 通过使用 AWS CLI 来运行 modify-db-cluster 并设置 --network-type 选项。 通过使用 RDS API 来调用 ModifyDBCluster 并设置 NetworkType 参数。	整个数据库集群	在此更改期间，不会出现中断。

设置和说明	方法	范围	停机说明
新建主密码 您的主用户密码。 • 对于 Aurora MySQL，密码必须包含 8–41 个可打印 ASCII 字符。 • 对于 Aurora PostgreSQL，它必须包含 8–99 个可打印的 ASCII 字符。 • 它不能包含 /、"、@ 或空格。	通过使用 AWS Management Console，修改数据库集群中的数据库实例。 通过使用 AWS CLI 来运行 `modify-db-cluster` 并设置 `--master-user-password` 选项。 通过使用 RDS API 来调用 `ModifyDBCluster` 并设置 `MasterUserPassword` 参数。	整个数据库集群	在此更改期间，不会出现中断。

设置和说明	方法	范围	停机说明
Performance Insights 是否启用 Performance Insights，该工具监控数据库实例负载，以便您可以分析数据库性能以及解决数据库性能问题。 有关更多信息，请参阅“在 Amazon Aurora 上使用性能详情监控数据库负载”。	通过使用 AWS Management Console，修改数据库集群中的数据库实例。 通过使用 AWS CLI 来运行 `modify-db-instance` 并设置 `--enable-performance-insights|--no-enable-performance-insights` 选项。 通过使用 RDS API 来调用 `ModifyDBInstance` 并设置 `EnablePerformanceInsights` 参数。	仅指定的数据库实例	在此更改期间，不会出现中断。

设置和说明	方法	范围	停机说明
Performance Insights AWS KMS key 用于加密 Performance Insights 数据的 AWS KMS key 标识符。KMS 密钥标识符是 Amazon Resource Name (ARN)、密钥标识符或 KMS 密钥的别名。 有关更多信息，请参阅 为 Aurora 打开和关闭性能详情。	通过使用 AWS Management Console，修改数据库集群中的数据库实例。 通过使用 AWS CLI 来运行 `modify-db-instance` 并设置 `--performance-insights-kms-key-id` 选项。 通过使用 RDS API 来调用 `ModifyDBInstance` 并设置 `PerformanceInsightsKMSKeyId` 参数。	仅指定的数据库实例	在此更改期间，不会出现中断。

设置和说明	方法	范围	停机说明
Performance Insights 保留期 保留 Performance Insights 数据的时间（天）。免费套餐中的保留设置为 Default (7 days) [原定设置（7 天）]。要将性能数据保留更长时间，请指定 1–24 个月。有关保留期的更多信息，请参阅性能详情的定价和数据留存。 有关更多信息，请参阅 为 Aurora 打开和关闭性能详情。	通过使用 AWS Management Console，修改数据库集群中的数据库实例。 通过使用 AWS CLI 来运行 `modify-db-instance` 并设置 `--performance-insights-retention-period` 选项。 通过使用 RDS API 来调用 `ModifyDBInstance` 并设置 `PerformanceInsightsRetentionPeriod` 参数。	仅指定的数据库实例	在此更改期间，不会出现中断。

设置和说明	方法	范围	停机说明
提升层 该值指定在现有主实例发生故障后，将 Aurora 副本提升为数据库集群中的主实例的顺序。 有关更多信息，请参阅 Aurora 数据库集群的容错能力。	通过使用 AWS Management Console，修改数据库集群中的数据库实例。 通过使用 AWS CLI 来运行 `modify-db-instance` 并设置 `--promotion-tier` 选项。 通过使用 RDS API 来调用 `ModifyDBInstance` 并设置 `PromotionTier` 参数。	仅指定的数据库实例	在此更改期间，不会出现中断。

设置和说明	方法	范围	停机说明
公有访问权限 可公开访问：为数据库实例提供公有 IP 地址，这表示可以在 VPC 外部访问该实例。要可供公开访问，数据库实例还必须在 VPC 的公有子网中。 不可公开访问：只能从 VPC 内部访问数据库实例。 有关更多信息，请参阅“对互联网隐藏 VPC 中的数据库集群”。 要从 Amazon VPC 外部连接到数据库实例，数据库实例必须可公开访问，必须使用数据库实例安全组的入站规则授予访问权限，并且必须满足其他要求。有关更多信息，请参阅“无法连接到 Amazon RDS 数据库实例”。 如果您的数据库实例不可公开访问，则您还可以使用AWS Site-to-Site VPN 连接或 AWS Direct Connect	通过使用 AWS Management Console，修改数据库集群中的数据库实例。 通过使用 AWS CLI 来运行 `modify-db-instance` 并设置 `--publicly-accessible|--no-publicly-accessible` 选项。 通过使用 RDS API 来调用 `ModifyDBInstance` 并设置 `PubliclyAccessible` 参数。	仅指定的数据库实例	在此更改期间，不会出现中断。

设置和说明	方法	范围	停机说明
连接从专用网络访问该实例。有关更多信息，请参阅“互联网络流量隐私”。			
Serverless v2 容量设置 以 Aurora 容量单位（ACU）为单位衡量的 Aurora Serverless v2 数据库集群的数据库容量。 有关更多信息，请参阅 为集群设置 Aurora Serverless v2 容量范围。	通过使用 AWS Management Console，使用控制台、CLI 和 API 修改数据库集群。 通过使用 AWS CLI 来运行 `modify-db-cluster` 并设置 `--serverless-v2-scaling-configuration` 选项。 通过使用 RDS API 来调用 `ModifyDBCluster` 并设置 `ServerlessV2ScalingConfiguration` 参数。	整个数据库集群	在此更改期间，不会出现中断。 更改立即发生。此设置忽略立即应用设置。

设置和说明	方法	范围	停机说明
安全组 要与数据库集群关联的安全组。 有关更多信息，请参阅“使用安全组控制访问权限”。	通过使用 AWS Management Console，使用控制台、CLI 和 API 修改数据库集群。 通过使用 AWS CLI 来运行 `modify-db-cluster` 并设置 `--vpc-security-group-ids` 选项。 通过使用 RDS API 来调用 `ModifyDBCluster` 并设置 `VpcSecurityGroupIds` 参数。	整个数据库集群	在此更改期间，不会出现中断。
目标回溯时段 您希望能够回溯数据库集群的时间长度（秒）。该设置仅适用于 Aurora MySQL；只有在启用了回溯的情况下创建数据库集群时才可用。	通过使用 AWS Management Console，使用控制台、CLI 和 API 修改数据库集群。 通过使用 AWS CLI 来运行 `modify-db-cluster` 并设置 `--backtrack-window` 选项。 通过使用 RDS API 来调用 `ModifyDBCluster` 并设置 `BacktrackWindow` 参数。	整个数据库集群	在此更改期间，不会出现中断。

不适用于 Amazon Aurora 数据库集群的设置

AWS CLI 命令 `modify-db-cluster` 和 RDS API 操作 `ModifyDBCluster` 中的以下设置不适用于 Amazon Aurora 数据库集群。

> **Note**
>
> 您不能使用 AWS Management Console 修改 Aurora 数据库集群的这些设置。

AWS CLI 设置	RDS API 设置
`--allocated-storage`	`AllocatedStorage`
`--auto-minor-version-upgrade | --no-auto-minor-version-upgrade`	`AutoMinorVersionUpgrade`
`--db-cluster-instance-class`	`DBClusterInstanceClass`
`--enable-performance-insights | --no-enable-performance-insights`	`EnablePerformanceInsights`
`--iops`	`Iops`
`--monitoring-interval`	`MonitoringInterval`
`--monitoring-role-arn`	`MonitoringRoleArn`
`--option-group-name`	`OptionGroupName`
`--performance-insights-kms-key-id`	`PerformanceInsightsKMSKeyId`
`--performance-insights-retention-period`	`PerformanceInsightsRetentionPeriod`

不适用于 Amazon Aurora 数据库实例的设置

AWS CLI 命令 modify-db-instance 和 RDS API 操作 ModifyDBInstance 中的以下设置不适用于 Amazon Aurora 数据库实例。

> **Note**
>
> 您不能使用 AWS Management Console 修改 Aurora 数据库实例的这些设置。

AWS CLI 设置	RDS API 设置
`--allocated-storage`	`AllocatedStorage`
`--allow-major-version-upgrade|--no-allow-major-version-upgrade`	`AllowMajorVersionUpgrade`
`--copy-tags-to-snapshot|--no-copy-tags-to-snapshot`	`CopyTagsToSnapshot`
`--domain`	`Domain`
`--db-security-groups`	`DBSecurityGroups`
`--db-subnet-group-name`	`DBSubnetGroupName`
`--domain-iam-role-name`	`DomainIAMRoleName`
`--multi-az|--no-multi-az`	`MultiAZ`
`--iops`	`Iops`
`--license-model`	`LicenseModel`
`--network-type`	`NetworkType`
`--option-group-name`	`OptionGroupName`
`--processor-features`	`ProcessorFeatures`

AWS CLI 设置	RDS API 设置
`--storage-type`	`StorageType`
`--tde-credential-arn`	`TdeCredentialArn`
`--tde-credential-password`	`TdeCredentialPassword`
`--use-default-processor-features|--no-use-default-processor-features`	`UseDefaultProcessorFeatures`

将 Aurora 副本添加到数据库集群

在使用复制的 Aurora 数据库集群中，具有一个主数据库实例和最多 15 个 Aurora 副本。主数据库实例支持读取和写入操作，并执行针对集群卷的所有数据修改。Aurora 副本连接到同一存储卷作为主数据库实例并仅支持读取操作。您可以使用 Aurora 副本从主数据库实例中分载读取工作负载。有关更多信息，请参阅 Aurora 副本。

Amazon Aurora 副本具有以下限制：

- 您无法为 Aurora Serverless v1 数据库集群创建 Aurora 副本。Aurora Serverless v1 有一个自动纵向扩展和缩减的数据库实例以支持所有数据库读写操作。

  不过，您可以将读取器实例添加到 Aurora Serverless v2 数据库集群。有关更多信息，请参阅 添加 Aurora Serverless v2 读取器。

我们建议您将 Aurora 数据库集群中的主实例和 Aurora 副本分配到多个可用区，以提高数据库集群的可用性。有关更多信息，请参阅“区域可用性”。

要从 Aurora 数据库集群中删除 Aurora 副本，请按照 从 Aurora 数据库集群中删除数据库实例 中的说明删除 Aurora 副本。

> **Note**
>
> Amazon Aurora 还支持对外部数据库的复制，例如 RDS 数据库实例。RDS 数据库实例必须与 Amazon Aurora 在同一个AWS区域。有关更多信息，请参阅 使用 Amazon Aurora 进行复制。

您可以使用AWS Management Console、AWS CLI 或 RDS API 将 Aurora 副本添加到数据库集群中。

控制台

将 Aurora 副本添加到数据库集群

1. 登录AWS Management Console并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Databases (数据库)，然后选择要在其中添加新数据库实例的数据库集群。
3. 确保集群和主实例都处于可用状态。如果数据库集群或主实例处于过渡状态（如 正在创建），则无法添加副本。

如果集群没有主实例，请使用 create-db-instance AWS CLI 命令创建一个主实例。如果您使用 CLI 还原了数据库集群快照，然后在AWS Management Console中查看集群，则可能会出现此情况。

4. 对于 Actions (操作)，选择 Add reader (添加读取器)。

   Add reader (添加读取器) 页面随即出现。

5. 在 Add reader (添加读取器) 页面上，为 Aurora 副本指定选项。下表显示 Aurora 副本的设置。

对于该选项	请执行该操作
可用区	确定您是否希望指定特定的可用区。该列表仅包括映射到您在创建数据库集群时选择的数据库子网组的那些可用区。有关可用区的更多信息，请参阅区域及可用区。
公开访问	选择 Yes 可向 Aurora 副本提供公有 IP 地址；否则，请选择 No。有关隐藏 Aurora 副本以防止公开访问的更多信息，请参阅对互联网隐藏 VPC 中的数据库集群。
加密	选择 Enable encryption 以对此 Aurora 副本启用静态加密。有关更多信息，请参阅“加密 Amazon Aurora 资源”。
数据库实例类	选择定义 Aurora 副本的处理和内存要求的数据库实例类。有关数据库实例类选项的更多信息，请参阅Amazon Aurora 数据库实例类。
Aurora 副本源	选择要为其创建 Aurora 副本的主实例的标识符。
数据库实例标识符	为该实例输入一个名称，该名称在您所选的 AWS 区域中对于您的账户是唯一的。您可选择对该名称进行一些巧妙处理，例如将所选的 AWS 区域和数据库引擎包括在名称中（如 **aurora-read-instance1** ）。
优先级	选择实例的故障转移优先级。如果您未选择值，则默认值为 tier-1。此优先级决定从主实例故障恢复时提升 Aurora 副本的顺序。有关更多信息，请参阅“Aurora 数据库集群的容错能力”。

对于该选项	请执行该操作
数据库端口	Aurora 副本的端口与数据库集群的端口相同。
数据库参数组	选择参数组。Aurora 具有一个可使用的默认参数组，您也可以创建自己的参数组。有关参数组的更多信息，请参阅 Amazon Aurora 的参数组。
Performance Insights	默认情况下，Turn on Performance Insights（开启 Performance Insights）复选框处于选中状态。该值不从写入器实例继承。有关更多信息，请参阅 在 Amazon Aurora 上使用性能详情监控数据库负载。
增强监控	选择启用增强监控可启用您的数据库集群在其上运行的操作系统的实时指标收集。有关更多信息，请参阅“使用增强监控来监控操作系统指标”。
监控角色	仅当增强监控设置为启用增强监控时可用。选择您创建的 IAM 角色以允许 Amazon RDS 与 Amazon CloudWatch Logs 通信，或选择默认以让 RDS 为您创建一个名为 `rds-monitoring-role` 的角色。有关更多信息，请参阅“使用增强监控来监控操作系统指标”。
粒度	仅当增强监控设置为启用增强监控时可用。设置为数据库集群收集指标的时间间隔（以秒为单位）。

对于该选项	请执行该操作
自动次要版本升级	如果要在次要数据库引擎版本升级可用时让 Aurora 数据库集群自动接收这些升级，请选择启用自动次要版本升级。 自动次要版本升级设置同时适用于 Aurora PostgreSQL 和 Aurora MySQL 数据库集群。对于 Aurora MySQL 2.x 集群，此设置将集群升级到最高版本 2.07.2。 有关 Aurora PostgreSQL 引擎更新的更多信息，请参阅 Amazon Aurora PostgreSQL 的数据库引擎更新。 有关 Aurora MySQL 引擎更新的更多信息，请参阅Amazon Aurora MySQL 的数据库引擎更新。

6. 选择 Add reader (添加读取器) 以创建 Aurora 副本。

AWS CLI

要在数据库集群中创建 Aurora 副本，请运行 create-db-instance AWS CLI 命令。包括数据库集群名称以作为 `--db-cluster-identifier` 选项。您可以选择使用 `--availability-zone` 参数为 Aurora 副本指定可用区，如以下示例中所示。

例如，以下命令创建一个名为 `sample-instance-us-west-2a` 的新 MySQL 5.7 兼容 Aurora 副本。

对于 Linux、macOS 或 Unix：

```
aws rds create-db-instance --db-instance-identifier sample-instance-us-west-2a \
    --db-cluster-identifier sample-cluster --engine aurora-mysql --db-instance-class
 db.r5.large \
    --availability-zone us-west-2a
```

对于 Windows：

```
aws rds create-db-instance --db-instance-identifier sample-instance-us-west-2a ^
    --db-cluster-identifier sample-cluster --engine aurora-mysql --db-instance-class
 db.r5.large ^
    --availability-zone us-west-2a
```

以下命令创建一个名为 sample-instance-us-west-2a 的新 MySQL 5.7 兼容 Aurora 副本。

对于 Linux、macOS 或 Unix：

```
aws rds create-db-instance --db-instance-identifier sample-instance-us-west-2a \
    --db-cluster-identifier sample-cluster --engine aurora-mysql --db-instance-class
 db.r5.large \
    --availability-zone us-west-2a
```

对于 Windows：

```
aws rds create-db-instance --db-instance-identifier sample-instance-us-west-2a ^
    --db-cluster-identifier sample-cluster --engine aurora --db-instance-class
 db.r5.large ^
    --availability-zone us-west-2a
```

以下命令创建名为 sample-instance-us-west-2a 的与 PostgreSQL 兼容的新 Aurora 副本。

对于 Linux、macOS 或 Unix：

```
aws rds create-db-instance --db-instance-identifier sample-instance-us-west-2a \
    --db-cluster-identifier sample-cluster --engine aurora-postgresql --db-instance-
class db.r5.large \
    --availability-zone us-west-2a
```

对于 Windows：

```
aws rds create-db-instance --db-instance-identifier sample-instance-us-west-2a ^
    --db-cluster-identifier sample-cluster --engine aurora-postgresql --db-instance-
class db.r5.large ^
    --availability-zone us-west-2a
```

RDS API

要在数据库集群中创建 Aurora 副本，请调用 CreateDBInstance 操作。包括数据库集群名称以作为 DBClusterIdentifier 参数。您可以选择使用 AvailabilityZone 参数为 Aurora 副本指定可用区。

有关使用 Aurora 副本自动扩缩 Amazon Aurora 的信息，请参阅以下各节。

主题

- Amazon Aurora Auto Scaling 与 Aurora 副本结合使用
- 向 Amazon Aurora 数据库集群添加自动扩缩策略
- 编辑 Amazon Aurora 数据库集群的自动扩缩策略
- 从 Amazon Aurora 数据库集群删除自动扩缩策略

Amazon Aurora Auto Scaling 与 Aurora 副本结合使用

为了满足您的连接和工作负载要求，Aurora Auto Scaling 动态调整为 Aurora 数据库集群预调配的 Aurora 副本数（读取器数据库实例）。Aurora Auto Scaling 适用于 Aurora MySQL 和 Aurora PostgreSQL。通过使用 Aurora Auto Scaling，Aurora 数据库集群可以处理连接或工作负载突然增加的情况。在连接或工作负载减少时，Aurora Auto Scaling 删除不需要的 Aurora 副本，以便您无需为未使用的配置数据库实例付费。

您可以定义一个扩展策略并将其应用于 Aurora 数据库集群。扩展策略 定义了 Aurora Auto Scaling 可以管理的最小和最大 Aurora 副本数。根据该策略，Aurora Auto Scaling 向上或向下调整 Aurora 副本数以响应实际工作负载，该负载是使用 Amazon CloudWatch 指标和目标值确定的。

> **Note**
>
> Aurora Auto Scaling 不适用于写入器数据库实例上的工作负载。Aurora Auto Scaling 只能协助处理读取器实例上的工作负载。

您可以使用 AWS Management Console根据预定义的指标应用扩展策略。或者，您也可以使用 AWS CLI 或 Aurora Auto Scaling API 根据预定义或自定义指标应用扩展策略。

主题

- 开始前的准备工作
- Aurora Auto Scaling 策略
- 数据库实例 ID 和标记
- Aurora Auto Scaling 和 Performance Insights

开始前的准备工作

您必须先创建一个带有主（写入器）数据库实例的 Aurora 数据库集群，然后才可以将 Aurora Auto Scaling 与 Aurora 数据库集群结合使用。有关创建 Aurora 数据库集群的更多信息，请参阅创建 Amazon Aurora 数据库集群。

仅当数据库集群处于可用状态时，Aurora Auto Scaling 才会扩缩数据库集群。

在 Aurora Auto Scaling 添加新的 Aurora 副本时，新 Aurora 副本是与主实例使用的数据库实例类相同的数据库实例类。有关数据库实例类的更多信息，请参阅Amazon Aurora 数据库实例类。而且，新

Aurora 副本的提升层设置为最低优先级，默认为 15。这意味着在故障转移期间，更高优先级的副本（如手动创建的副本）会先提升。有关更多信息，请参阅“Aurora 数据库集群的容错能力”。

Aurora Auto Scaling 仅删除它创建的 Aurora 副本。

要利用 Aurora Auto Scaling，应用程序必须支持连接到新的 Aurora 副本。为此，建议您使用 Aurora 读取器端点。您可以使用驱动程序，如 AWS JDBC 驱动程序。有关更多信息，请参阅 连接到 Amazon Aurora 数据库集群。

> **Note**
>
> Aurora 全局数据库目前不支持备用数据库集群的 Aurora Auto Scaling。

Aurora Auto Scaling 策略

Aurora Auto Scaling 使用扩展策略调整 Aurora 数据库集群中的 Aurora 副本数。Aurora Auto Scaling 具有以下组件：

- 服务相关角色
- 目标指标
- 最小和最大容量
- 冷却时间

主题

- 服务相关角色
- 目标指标
- 最小和最大容量
- 冷却时间
- 启用或禁用缩减活动
- 添加、编辑或删除自动扩缩策略

服务相关角色

Aurora Auto Scaling 使用 `AWSServiceRoleForApplicationAutoScaling_RDSCluster` 服务相关角色。有关更多信息，请参阅 Application Auto Scaling 用户指南 中的 Application Auto Scaling 服务相关角色。

目标指标

在这种类型的策略中，预定义或自定义指标以及指标目标值是在目标跟踪扩展策略配置中指定的。Aurora Auto Scaling 创建和管理触发扩展策略的 CloudWatch 警报，并根据指标和目标值计算扩展调整。扩展策略根据需要添加或删除 Aurora 副本，以便将指标保持在指定的目标值或该值附近。除了将指标保持在目标值附近以外，目标跟踪扩展策略还会根据由于工作负载变化而造成的指标波动进行调整。这种策略还会最大限度减少数据库集群的可用 Aurora 副本数的快速波动。

例如，使用具有预定义的平均 CPU 使用率指标的扩展策略。这种策略可以将 CPU 使用率保持在指定的使用率百分比 (如 40%) 或该值附近。

> **Note**
>
> 对于每个 Aurora 数据库集群，您只能为每个目标指标创建一个 Auto Scaling 策略。

配置 Aurora Auto Scaling 时，将根据集群中所有读取器实例的平均值计算目标指标值。此计算的执行方式如下：

- 包括 Aurora 集群中的所有读取器实例，无论这些实例是由 Auto Scaling 管理还是手动添加的。
- 包括与自定义端点关联的实例。自定义端点不会影响目标指标的计算。
- 不包括集群的写入器实例。

这些指标从 CloudWatch 中使用以下维度派生：

- `DBClusterIdentifier`
- `Role=READER`

例如，考虑一个具有以下设置的 Aurora MySQL 集群：

- 手动实例（不受 Auto Scaling 控制）：
  - 写入器的 CPU 利用率为 50%
  - 读取器 1（自定义端点：`custom-reader-1`）的 CPU 利用率为 90%
  - 读取器 2（自定义端点：`custom-reader-2`）的 CPU 利用率为 90%
- Auto Scaling 实例：
  - 读取器 3（使用 Auto Scaling 添加）的 CPU 利用率为 10%

在这种情况下，Auto Scaling 策略的目标指标的计算方法如下：

```
Target metric = (CPU utilization of reader 1 + reader 2 + reader 3) / total number of
 readers

Target metric = (90 + 90 + 10) / 3 = 63.33%
```

Auto Scaling 策略使用此值，根据定义的阈值来评估是横向缩减还是横向扩展。

请考虑以下事项：

- 尽管自定义端点决定如何将流量路由到特定读取器，但它们不会将读取器排除在指标计算之外。
- 手动实例始终包含在目标指标计算中。
- 为避免意外的扩展行为，请确保 Auto Scaling 配置考虑到了集群中的所有读取器实例。
- 如果集群没有读取器，则不会计算该指标，并且 Auto Scaling 策略警报将保持不活动状态。为使 Auto Scaling 策略有效运行，必须始终至少有一个读取器。

最小和最大容量

您可以指定 Application Auto Scaling 管理的最大 Aurora 副本数。必须将该值设置为 0–15，并且必须大于或等于为最小 Aurora 副本数指定的值。

您还可以指定 Application Auto Scaling 管理的最小 Aurora 副本数。必须将该值设置为 0–15，并且必须小于或等于为最大 Aurora 副本数指定的值。

必须至少有一个读取器数据库实例，Aurora Auto Scaling 才能正常工作。如果数据库集群没有读取器实例，并且您将最低容量设置为 0，则 Aurora Auto Scaling 将无法工作。

> **Note**
>
> 最小和最大容量是为 Aurora 数据库集群设置的。指定的值适用于与 Aurora 数据库集群关联的所有策略。

冷却时间

您可以添加影响 Aurora 数据库集群扩展和缩减的冷却时间，以调整目标跟踪扩展策略的响应速度。冷却时间阻止后续扩展或缩减请求，直至冷却时间到期。这种阻止降低了为缩减请求删除 Aurora 数据库集群中的 Aurora 副本的速度，并降低了为扩展请求创建 Aurora 副本的速度。

您可以指定以下冷却时间：

- 缩减活动减少 Aurora 数据库集群中的 Aurora 副本数。缩减冷却时间指定在完成一个缩减活动后开始另一个缩减活动之前等待的时间 (秒)。
- 扩展活动增加 Aurora 数据库集群中的 Aurora 副本数。扩展冷却时间指定在完成一个扩展活动后开始另一个扩展活动之前等待的时间 (秒)。

> **Note**
>
> 如果后续的横向扩展请求需要的 Aurora 副本数量大于第一个请求的数量，则忽略横向扩展冷却时间。

如果您未设置横向缩减或横向扩展冷却时间，则冷却时间的默认值为 300 秒。

启用或禁用缩减活动

您可以为策略启用或禁用缩减活动。启用缩减活动允许扩展策略删除 Aurora 副本。在启用缩减活动时，扩展策略中的缩减冷却时间将应用于缩减活动。禁用缩减活动将禁止扩展策略删除 Aurora 副本。

> **Note**
>
> 扩展活动始终处于启用状态，以便扩展策略可以根据需要创建 Aurora 副本。

添加、编辑或删除自动扩缩策略

可以使用 AWS Management Console、AWS CLI 或 Application Auto Scaling API 添加、编辑或删除自动扩缩策略。有关添加、编辑或删除自动扩缩策略的更多信息，请参阅以下各节。

- 向 Amazon Aurora 数据库集群添加自动扩缩策略
- 编辑 Amazon Aurora 数据库集群的自动扩缩策略
- 从 Amazon Aurora 数据库集群删除自动扩缩策略

数据库实例 ID 和标记

当 Aurora Auto Scaling 添加副本时，其数据库实例 ID 将以 `application-autoscaling-` 作为前缀，例如 `application-autoscaling-61aabbcc-4e2f-4c65-b620-ab7421abc123`。

以下标签将自动添加到数据库实例。您可以在数据库实例详细信息页面的 Tags（标签）选项卡上查看它。

标签	值
application-autoscaling:resourceId	cluster:mynewcluster-cluster

有关 Amazon RDS 资源标签的更多信息，请参阅 为 Amazon Aurora 和 Amazon RDS 资源添加标签。

Aurora Auto Scaling 和 Performance Insights

您可以使用 Performance Insights 来监控 Aurora Auto Scaling 已添加的副本，就像监视任何 Aurora 读取器数据库实例一样。

有关使用 Performance Insights 监控 Aurora 数据库集群的更多信息，请参阅在 Amazon Aurora 上使用性能详情监控数据库负载。

向 Amazon Aurora 数据库集群添加自动扩缩策略

您可以使用AWS Management Console、AWS CLI 或 Application Auto Scaling API 添加扩展策略。

> **Note**
>
> 有关使用 AWS CloudFormation 添加扩展策略的示例，请参阅 AWS CloudFormation 用户指南中声明 Aurora 数据库集群的扩展策略。

控制台

您可以使用AWS Management Console将扩展策略添加到 Aurora 数据库集群中。

在 Aurora 数据库集群中添加 Auto Scaling 策略

1. 登录AWS Management Console并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Databases (数据库)。
3. 选择要添加策略的 Aurora 数据库集群。

4. 选择 Logs & events (日志和事件) 选项卡。
5. 在 Auto Scaling 策略部分中，选择添加。

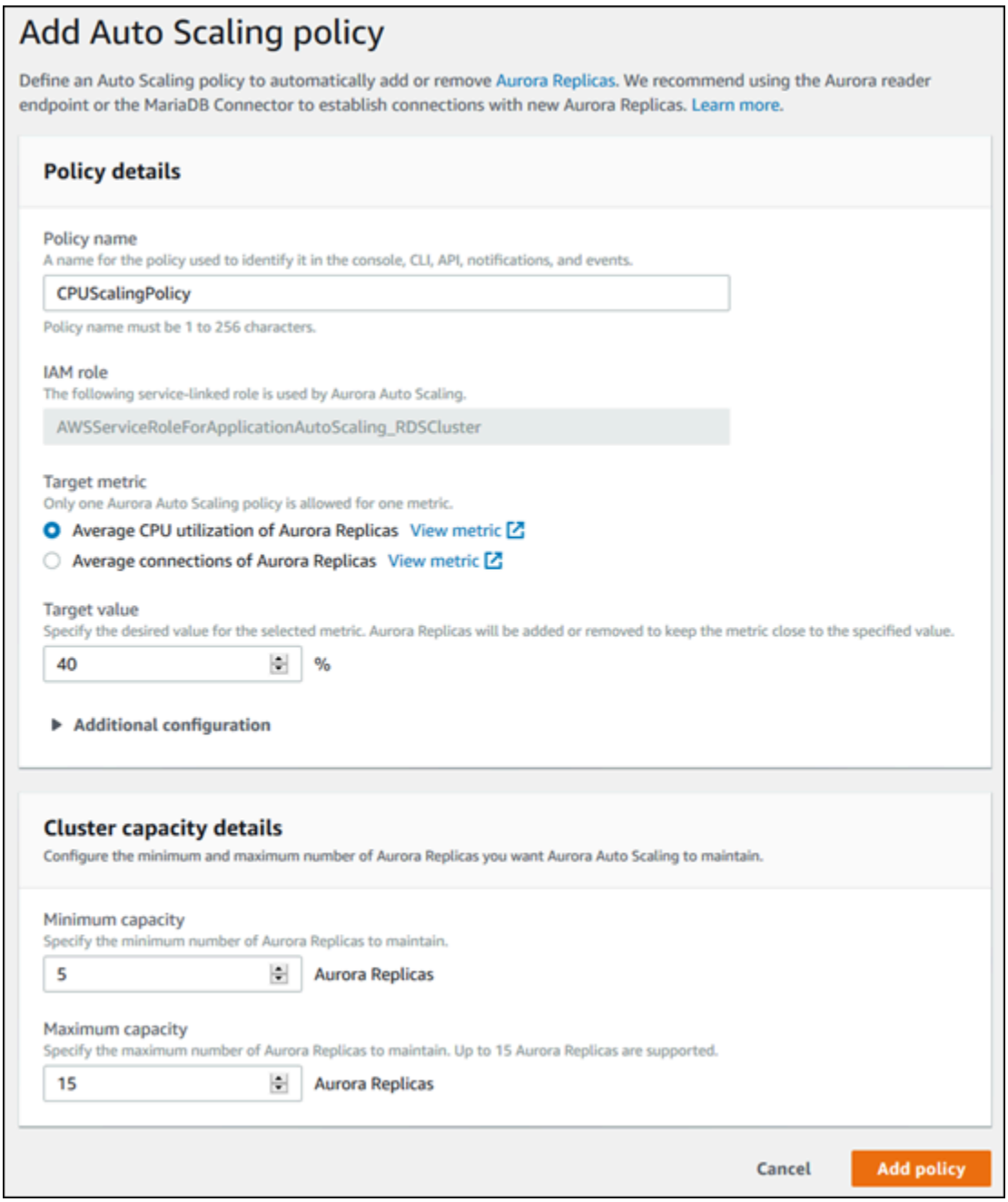
   将显示添加 Auto Scaling 策略对话框。
6. 对于 Policy Name (策略名称)，键入策略名称。
7. 对于目标指标，请选择以下选项之一：

   - Aurora 副本的平均 CPU 使用率，用于创建基于平均 CPU 使用率的策略。
   - Aurora 副本的平均连接数，用于创建基于 Aurora 副本的平均连接数的策略。
8. 对于目标值，请键入以下值之一：

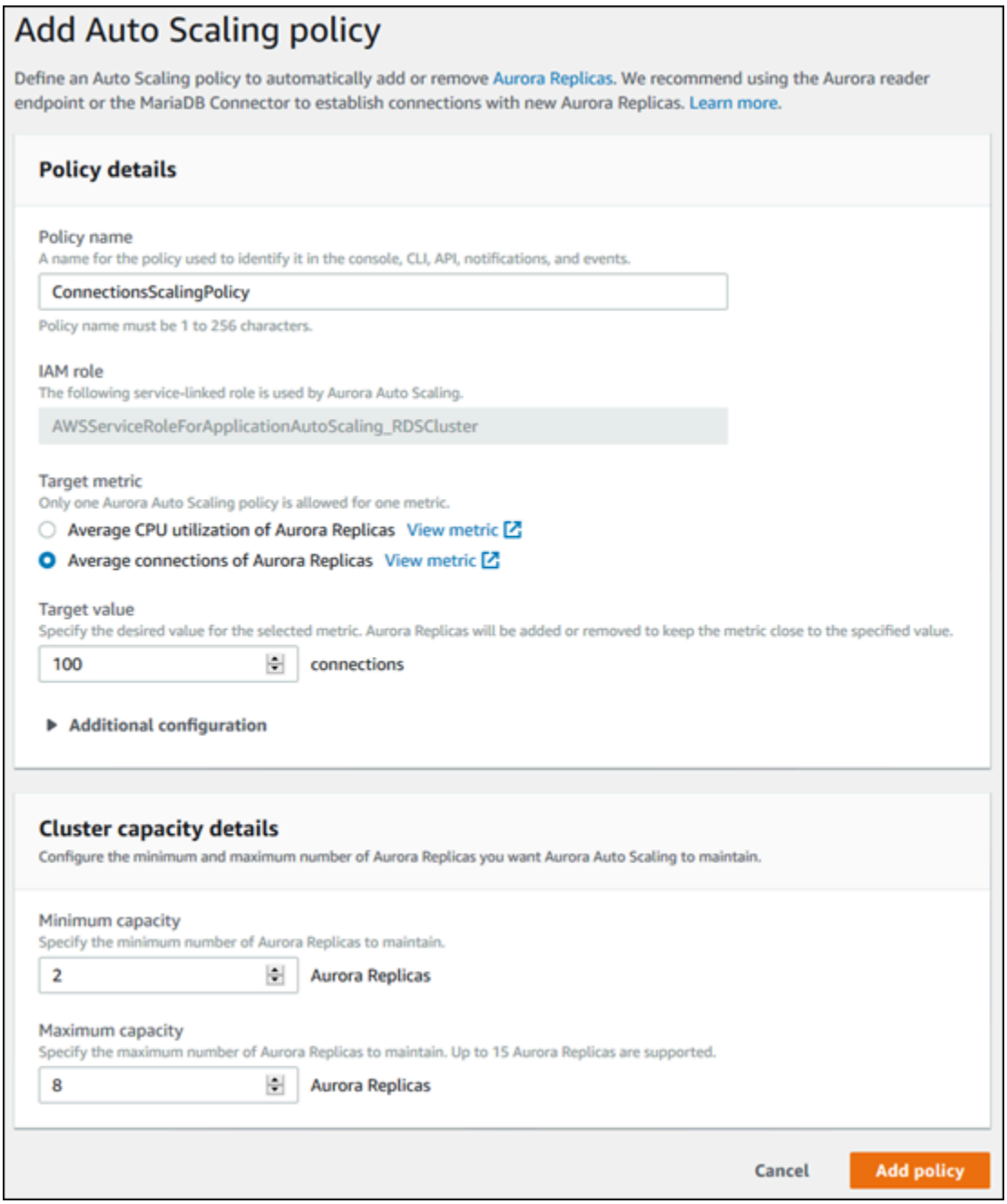
   - 如果在上一步中选择 Aurora 副本的平均 CPU 使用率，请键入要在 Aurora 副本上保持的 CPU 使用率百分比。
   - 如果在上一步中选择 Aurora 副本的平均连接数，请键入要保持的连接数。

   将添加或删除 Aurora 副本以使指标接近于指定的值。
9. （可选）展开 Additional Configuration（附加配置）以创建横向缩减或横向扩展冷却时间。
10. 对于最小容量，请键入 Aurora Auto Scaling 策略需要保持的最小 Aurora 副本数。
11. 对于最大容量，请键入 Aurora Auto Scaling 策略需要保持的最大 Aurora 副本数。
12. 选择添加策略。

以下对话框创建一个基于平均 CPU 使用率 40% 的 Auto Scaling 策略。该策略指定最少 5 个 Aurora 副本，最多 15 个 Aurora 副本。



以下对话框创建一个基于平均连接数 100 的 Auto Scaling 策略。该策略指定最少 2 个 Aurora 副本，最多 8 个 Aurora 副本。



AWS CLI 或 Application Auto Scaling API

您可以应用基于预定义或自定义指标的扩展策略。为此，您可以使用 AWS CLI 或 Application Auto Scaling API。第一步是在 Application Auto Scaling 中注册 Aurora 数据库集群。

注册 Aurora 数据库集群

您必须先在 Application Auto Scaling 中注册 Aurora 数据库集群，然后才能在 Aurora 数据库集群中使用 Aurora Auto Scaling。这样做是为了定义应用于该集群的扩展维度和限制。Application Auto Scaling

沿 rds:cluster:ReadReplicaCount 可扩展维度动态扩展 Aurora 数据库集群，它表示 Aurora 副本的数量。

要注册 Aurora 数据库集群，您可以使用 AWS CLI 或 Application Auto Scaling API。

AWS CLI

要注册 Aurora 数据库集群，请使用具有以下参数的 register-scalable-target AWS CLI 命令：

- --service-namespace – 将该值设置为 rds。
- --resource-id – Aurora 数据库集群的资源标识符。对于该参数，资源类型为 cluster，唯一标识符为 Aurora 数据库集群的名称，例如，cluster:myscalablecluster。
- --scalable-dimension – 将该值设置为 rds:cluster:ReadReplicaCount。
- --min-capacity – 由 Application Auto Scaling 管理的最小读取器数据库实例数。有关集群中 --min-capacity、--max-capacity 和数据库实例数之间关系的信息，请参阅 最小和最大容量。
- --max-capacity – 由 Application Auto Scaling 管理的最大读取器数据库实例数。有关集群中 --min-capacity、--max-capacity 和数据库实例数之间关系的信息，请参阅 最小和最大容量。

Example

在以下示例中，您注册一个名为 myscalablecluster 的 Aurora 数据库集群。该注册表示应动态扩展数据库集群以具有 1-8 个 Aurora 副本。

对于 Linux、macOS 或 Unix：

```
aws application-autoscaling register-scalable-target \
    --service-namespace rds \
    --resource-id cluster:myscalablecluster \
    --scalable-dimension rds:cluster:ReadReplicaCount \
    --min-capacity 1 \
    --max-capacity 8 \
```

对于 Windows：

```
aws application-autoscaling register-scalable-target ^
    --service-namespace rds ^
    --resource-id cluster:myscalablecluster ^
    --scalable-dimension rds:cluster:ReadReplicaCount ^
    --min-capacity 1 ^
```

```
    --max-capacity 8 ^
```

Application Auto Scaling API

要在 Application Auto Scaling 中注册 Aurora 数据库集群，请使用具有以下参数的 RegisterScalableTarget Application Auto Scaling API 操作：

- `ServiceNamespace` – 将该值设置为 `rds`。
- `ResourceID` – Aurora 数据库集群的资源标识符。对于该参数，资源类型为 `cluster`，唯一标识符为 Aurora 数据库集群的名称，例如，`cluster:myscalablecluster`。
- `ScalableDimension` – 将该值设置为 `rds:cluster:ReadReplicaCount`。
- `MinCapacity` – 由 Application Auto Scaling 管理的最小读取器数据库实例数。有关集群中 `MinCapacity`、`MaxCapacity` 和数据库实例数之间关系的信息，请参阅 最小和最大容量。
- `MaxCapacity` – 由 Application Auto Scaling 管理的最大读取器数据库实例数。有关集群中 `MinCapacity`、`MaxCapacity` 和数据库实例数之间关系的信息，请参阅 最小和最大容量。

Example

在以下示例中，您使用 Application Auto Scaling API 注册一个名为 `myscalablecluster` 的 Aurora 数据库集群。该注册表示应动态扩展数据库集群以具有 1-8 个 Aurora 副本。

```
POST / HTTP/1.1
Host: autoscaling.us-east-2.amazonaws.com
Accept-Encoding: identity
Content-Length: 219
X-Amz-Target: AnyScaleFrontendService.RegisterScalableTarget
X-Amz-Date: 20160506T182145Z
User-Agent: aws-cli/1.10.23 Python/2.7.11 Darwin/15.4.0 botocore/1.4.8
Content-Type: application/x-amz-json-1.1
Authorization: AUTHPARAMS

{
    "ServiceNamespace": "rds",
    "ResourceId": "cluster:myscalablecluster",
    "ScalableDimension": "rds:cluster:ReadReplicaCount",
    "MinCapacity": 1,
    "MaxCapacity": 8
}
```

为 Aurora 数据库集群定义扩展策略

目标跟踪扩展策略配置是由 JSON 块表示的，其中定义了指标和目标值。您可以在文本文件中将扩展策略配置保存为 JSON 块。在调用 AWS CLI 或 Application Auto Scaling API 时，您可以使用该文本文件。有关策略配置语法的更多信息，请参阅 Application Auto Scaling API 参考 中的 `TargetTrackingScalingPolicyConfiguration`。

可以使用以下选项定义目标跟踪扩展策略配置。

主题

- 使用预定义的指标
- 使用自定义指标
- 使用冷却时间
- 禁用缩减活动

使用预定义的指标

通过使用预定义的指标，您可以快速为 Aurora 数据库集群定义目标跟踪扩展策略，它非常适合 Aurora Auto Scaling 中的目标跟踪和动态扩展。

目前，Aurora 在 Aurora Auto Scaling 中支持以下预定义指标：

- RDSReaderAverageCPUUtilization – CloudWatch 中的 `CPUUtilization` 指标在 Aurora 数据库集群的所有 Aurora 副本中的平均值。
- RDSReaderAverageDatabaseConnections – CloudWatch 中的 `DatabaseConnections` 指标在 Aurora 数据库集群的所有 Aurora 副本中的平均值。

有关 `CPUUtilization` 和 `DatabaseConnections` 指标的更多信息，请参阅Amazon Aurora 的 Amazon CloudWatch 指标。

要在扩展策略中使用预定义的指标，您需要为扩展策略创建一个目标跟踪配置。该配置必须包含 `PredefinedMetricSpecification` 以表示预定义的指标，并包含 `TargetValue` 以表示该指标的目标值。

Example

以下示例说明了 Aurora 数据库集群的典型目标跟踪扩展策略配置。在该配置中，`RDSReaderAverageCPUUtilization` 预定义指标用于根据所有 Aurora 副本中的平均 CPU 使用率 40% 调整 Aurora 数据库集群。

```
{
    "TargetValue": 40.0,
    "PredefinedMetricSpecification":
    {
        "PredefinedMetricType": "RDSReaderAverageCPUUtilization"
    }
}
```

使用自定义指标

通过使用自定义指标，您可以定义满足您的自定义要求的目标跟踪扩展策略。您可以根据随扩展按比例变化的任何 Aurora 指标定义一个自定义指标。

并非所有 Aurora 指标都适用于目标跟踪。指标必须是有效的使用率指标，它用于描述实例的繁忙程度。指标值必须随 Aurora 数据库集群中的 Aurora 副本数按比例增加或减少。要使用指标数据按比例扩展或缩减 Aurora 副本数，必须按比例进行这种增加或减少。

Example

以下示例说明了扩展策略的目标跟踪配置。在该配置中，一个自定义指标根据名为 `my-db-cluster` 的 Aurora 数据库集群的所有 Aurora 副本中的平均 CPU 使用率 50% 调整该 Aurora 数据库集群。

```
{
    "TargetValue": 50,
    "CustomizedMetricSpecification":
    {
        "MetricName": "CPUUtilization",
        "Namespace": "AWS/RDS",
        "Dimensions": [
            {"Name": "DBClusterIdentifier","Value": "my-db-cluster"},
            {"Name": "Role","Value": "READER"}
        ],
        "Statistic": "Average",
        "Unit": "Percent"
    }
}
```

使用冷却时间

您可以为 `ScaleOutCooldown` 指定一个值（秒）以添加扩展 Aurora 数据库集群的冷却时间。同样，您可以为 `ScaleInCooldown` 添加一个值（秒）以添加缩减 Aurora 数据库集群的冷却时间。有关

ScaleInCooldown 和 ScaleOutCooldown 的更多信息，请参阅 Application Auto Scaling API 参考 中的 TargetTrackingScalingPolicyConfiguration。

Example

以下示例说明了扩展策略的目标跟踪配置。在此配置中，RDSReaderAverageCPUUtilization 预定义指标用于根据 Aurora 数据库集群中所有 Aurora 副本上 40% 的平均 CPU 利用率来调整该 Aurora 数据库集群。该配置将缩减冷却时间指定为 10 分钟，并将扩展冷却时间指定为 5 分钟。

```
{
    "TargetValue": 40.0,
    "PredefinedMetricSpecification":
    {
        "PredefinedMetricType": "RDSReaderAverageCPUUtilization"
    },
    "ScaleInCooldown": 600,
    "ScaleOutCooldown": 300
}
```

禁用缩减活动

您可以禁用缩减活动以禁止目标跟踪扩展策略配置缩减 Aurora 数据库集群。禁用缩减活动将禁止扩展策略删除 Aurora 副本，同时仍允许扩展策略根据需要创建副本。

您可以为 DisableScaleIn 指定一个布尔值，以便为 Aurora 数据库集群启用或禁用缩减活动。有关 DisableScaleIn 的更多信息，请参阅 Application Auto Scaling API 参考 中的 TargetTrackingScalingPolicyConfiguration。

Example

以下示例说明了扩展策略的目标跟踪配置。在该配置中，RDSReaderAverageCPUUtilization 预定义指标根据一个 Aurora 数据库集群的所有 Aurora 副本中的平均 CPU 使用率 40% 调整该 Aurora 数据库集群。该配置禁用扩展策略的缩减活动。

```
{
    "TargetValue": 40.0,
    "PredefinedMetricSpecification":
    {
        "PredefinedMetricType": "RDSReaderAverageCPUUtilization"
    },
    "DisableScaleIn": true
```

```
}
```

为 Aurora 数据库集群应用扩展策略

在 Application Auto Scaling 中注册 Aurora 数据库集群并定义扩展策略后，您可以将扩展策略应用于注册的 Aurora 数据库集群。要将扩展策略应用于 Aurora 数据库集群，您可以使用 AWS CLI 或 Application Auto Scaling API。

AWS CLI

要将扩展策略应用于 Aurora 数据库集群，请使用具有以下参数的 put-scaling-policy AWS CLI 命令：

- `--policy-name` – 扩展策略的名称。
- `--policy-type` – 将该值设置为 `TargetTrackingScaling`。
- `--resource-id` – Aurora 数据库集群的资源标识符。对于该参数，资源类型为 `cluster`，唯一标识符为 Aurora 数据库集群的名称，例如，`cluster:myscalablecluster`。
- `--service-namespace` – 将该值设置为 `rds`。
- `--scalable-dimension` – 将该值设置为 `rds:cluster:ReadReplicaCount`。
- `--target-tracking-scaling-policy-configuration` – 用于 Aurora 数据库集群的目标跟踪扩展策略配置。

Example

在以下示例中，您使用 Application Auto Scaling 将名为 `myscalablepolicy` 的目标跟踪扩展策略应用于名为 `myscalablecluster` 的 Aurora 数据库集群。为此，请使用在名为 `config.json` 的文件中保存的策略配置。

对于 Linux、macOS 或 Unix：

```
aws application-autoscaling put-scaling-policy \
    --policy-name myscalablepolicy \
    --policy-type TargetTrackingScaling \
    --resource-id cluster:myscalablecluster \
    --service-namespace rds \
    --scalable-dimension rds:cluster:ReadReplicaCount \
    --target-tracking-scaling-policy-configuration file://config.json
```

对于 Windows：

```
aws application-autoscaling put-scaling-policy ^
    --policy-name myscalablepolicy ^
    --policy-type TargetTrackingScaling ^
    --resource-id cluster:myscalablecluster ^
    --service-namespace rds ^
    --scalable-dimension rds:cluster:ReadReplicaCount ^
    --target-tracking-scaling-policy-configuration file://config.json
```

Application Auto Scaling API

要使用 Application Auto Scaling API 将扩展策略应用于 Aurora 数据库集群，请使用具有以下参数的 PutScalingPolicy Application Auto Scaling API 操作：

- `PolicyName` – 扩展策略的名称。
- `ServiceNamespace` – 将该值设置为 `rds`。
- `ResourceID` – Aurora 数据库集群的资源标识符。对于该参数，资源类型为 `cluster`，唯一标识符为 Aurora 数据库集群的名称，例如，`cluster:myscalablecluster`。
- `ScalableDimension` – 将该值设置为 `rds:cluster:ReadReplicaCount`。
- `PolicyType` – 将该值设置为 `TargetTrackingScaling`。
- `TargetTrackingScalingPolicyConfiguration` – 用于 Aurora 数据库集群的目标跟踪扩展策略配置。

Example

在以下示例中，您使用 Application Auto Scaling 将名为 `myscalablepolicy` 的目标跟踪扩展策略应用于名为 `myscalablecluster` 的 Aurora 数据库集群。您使用的策略配置基于 `RDSReaderAverageCPUUtilization` 预定义指标。

```
POST / HTTP/1.1
Host: autoscaling.us-east-2.amazonaws.com
Accept-Encoding: identity
Content-Length: 219
X-Amz-Target: AnyScaleFrontendService.PutScalingPolicy
X-Amz-Date: 20160506T182145Z
```

```
User-Agent: aws-cli/1.10.23 Python/2.7.11 Darwin/15.4.0 botocore/1.4.8
Content-Type: application/x-amz-json-1.1
Authorization: AUTHPARAMS

{
    "PolicyName": "myscalablepolicy",
    "ServiceNamespace": "rds",
    "ResourceId": "cluster:myscalablecluster",
    "ScalableDimension": "rds:cluster:ReadReplicaCount",
    "PolicyType": "TargetTrackingScaling",
    "TargetTrackingScalingPolicyConfiguration": {
        "TargetValue": 40.0,
        "PredefinedMetricSpecification":
        {
            "PredefinedMetricType": "RDSReaderAverageCPUUtilization"
        }
    }
}
```

编辑 Amazon Aurora 数据库集群的自动扩缩策略

您可以使用 AWS Management Console、AWS CLI 或 Application Auto Scaling API 编辑扩缩策略。

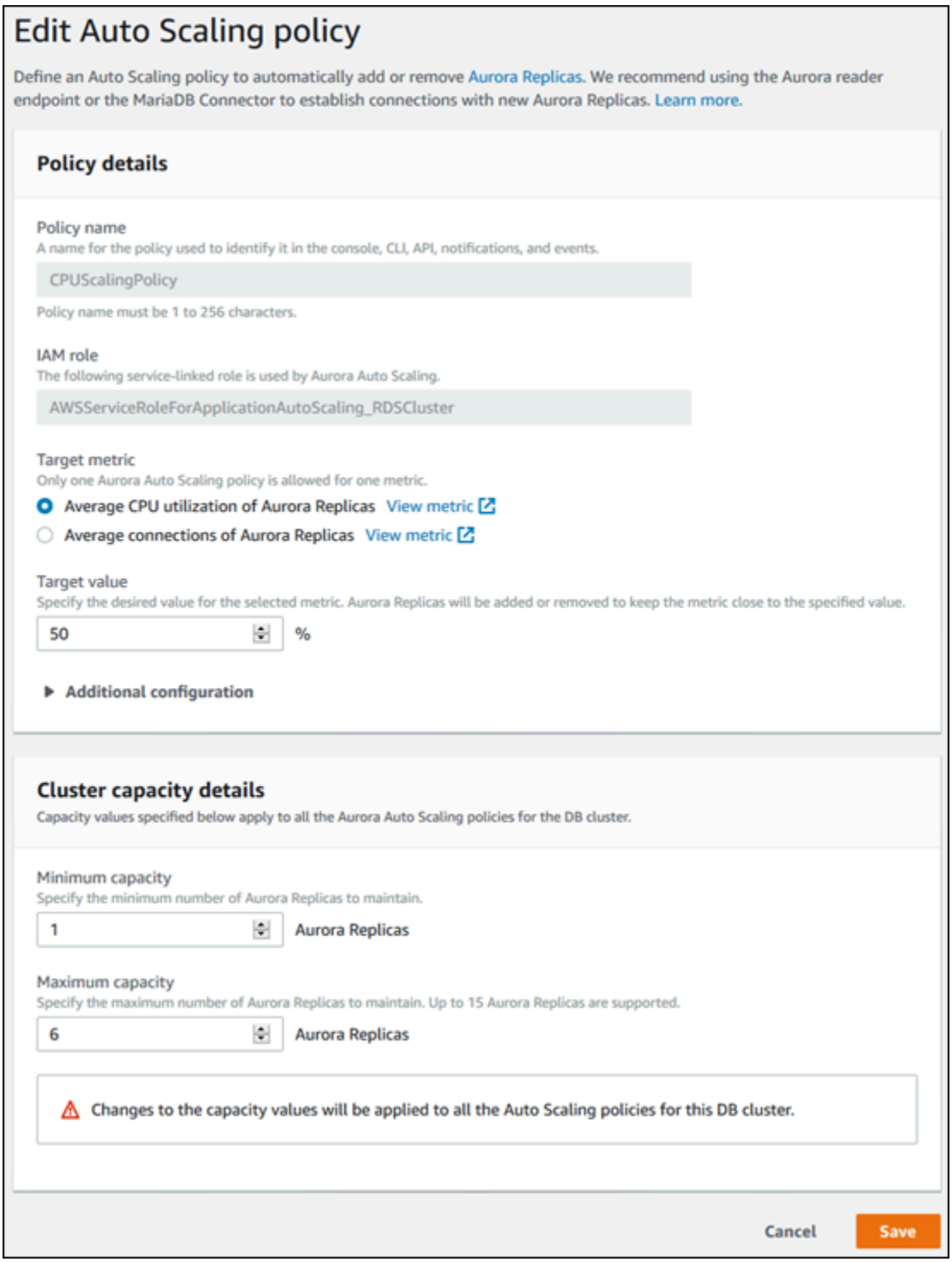
控制台

您可以使用 AWS Management Console编辑扩展策略。

编辑 Aurora 数据库集群的 Auto Scaling 策略

1. 登录AWS Management Console并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Databases (数据库)。
3. 选择要编辑 Auto Scaling 策略的 Aurora 数据库集群。
4. 选择 Logs & events (日志和事件) 选项卡。
5. 在 Auto Scaling 策略部分中，选择 Auto Scaling 策略，然后选择编辑。
6. 对该策略进行更改。
7. 选择 Save。

以下是一个示例编辑 Auto Scaling 策略对话框。



AWS CLI 或 Application Auto Scaling API

您可以使用 AWS CLI 或 Application Auto Scaling API 按照与应用扩缩策略相同的方式编辑扩缩策略：

- 在使用 AWS CLI 时，请在 `--policy-name` 参数中指定要编辑的策略名称。为要更改的参数指定新的值。

- 在使用 Application Auto Scaling API 时，请在 `PolicyName` 参数中指定要编辑的策略名称。为要更改的参数指定新的值。

有关更多信息，请参阅 为 Aurora 数据库集群应用扩展策略。

从 Amazon Aurora 数据库集群删除自动扩缩策略

您可以使用AWS Management Console、AWS CLI 或 Application Auto Scaling API 删除扩缩策略。

控制台

您可以使用 AWS Management Console删除扩展策略。

删除 Aurora 数据库集群的 Auto Scaling 策略

1. 登录AWS Management Console并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Databases (数据库)。
3. 选择要删除 Auto Scaling 策略的 Aurora 数据库集群。
4. 选择 Logs & events (日志和事件) 选项卡。
5. 在 Auto Scaling 策略部分中，选择 Auto Scaling 策略，然后选择删除。

AWS CLI

要从 Aurora 数据库集群中删除扩展策略，请使用具有以下参数的 `delete-scaling-policy` AWS CLI 命令：

- `--policy-name` – 扩展策略的名称。
- `--resource-id` – Aurora 数据库集群的资源标识符。对于该参数，资源类型为 `cluster`，唯一标识符为 Aurora 数据库集群的名称，例如，`cluster:myscalablecluster`。
- `--service-namespace` – 将该值设置为 `rds`。
- `--scalable-dimension` – 将该值设置为 `rds:cluster:ReadReplicaCount`。

Example

在以下示例中，您从名为 `myscalablepolicy` 的 Aurora 数据库集群中删除名为 `myscalablecluster` 的目标跟踪扩展策略。

对于 Linux、macOS 或 Unix：

```
aws application-autoscaling delete-scaling-policy \
    --policy-name myscalablepolicy \
    --resource-id cluster:myscalablecluster \
    --service-namespace rds \
    --scalable-dimension rds:cluster:ReadReplicaCount \
```

对于 Windows：

```
aws application-autoscaling delete-scaling-policy ^
    --policy-name myscalablepolicy ^
    --resource-id cluster:myscalablecluster ^
    --service-namespace rds ^
    --scalable-dimension rds:cluster:ReadReplicaCount ^
```

Application Auto Scaling API

要从 Aurora 数据库集群中删除扩展策略，请使用具有以下参数的 DeleteScalingPolicy Application Auto Scaling API 操作：

- `PolicyName` – 扩展策略的名称。
- `ServiceNamespace` – 将该值设置为 `rds`。
- `ResourceID` – Aurora 数据库集群的资源标识符。对于该参数，资源类型为 `cluster`，唯一标识符为 Aurora 数据库集群的名称，例如，`cluster:myscalablecluster`。
- `ScalableDimension` – 将该值设置为 `rds:cluster:ReadReplicaCount`。

Example

在以下示例中，您将从 Application Auto Scaling 使用 API 名为的 Aurora 数据库群集中删除 `myscalablepolicy` 名为 `myscalablecluster` 的目标跟踪扩展策略。

```
POST / HTTP/1.1
Host: autoscaling.us-east-2.amazonaws.com
Accept-Encoding: identity
```

```
Content-Length: 219
X-Amz-Target: AnyScaleFrontendService.DeleteScalingPolicy
X-Amz-Date: 20160506T182145Z
User-Agent: aws-cli/1.10.23 Python/2.7.11 Darwin/15.4.0 botocore/1.4.8
Content-Type: application/x-amz-json-1.1
Authorization: AUTHPARAMS

{
    "PolicyName": "myscalablepolicy",
    "ServiceNamespace": "rds",
    "ResourceId": "cluster:myscalablecluster",
    "ScalableDimension": "rds:cluster:ReadReplicaCount"
}
```

管理 Aurora 数据库集群的性能和扩展

您可以使用以下选项管理 Aurora 数据库集群和数据库实例的性能和扩展：

主题

- 存储扩展
- 实例扩展
- 读取扩展
- 管理连接
- 管理查询执行计划

存储扩展

Aurora 存储自动使用您的集群卷中的数据进行扩展。随着数据的增长，您的集群卷存储最大可扩展到 128 TiB 或 64 TiB。最大大小取决于数据库引擎版本。要了解集群卷中包含哪些类型的数据，请参阅Amazon Aurora 存储。有关特定版本最大大小的详细信息，请参阅 Amazon Aurora 大小限制。

每小时评估一次集群卷的大小以确定存储成本。有关定价的信息，请参阅 Aurora 定价页面。

尽管 Aurora 集群卷的大小可以扩展到许多 TB，但您只需为卷中使用的空间付费。确定计费存储空间的机制取决于 Aurora 集群的版本。

- 从集群卷中删除 Aurora 数据时，总计费空间会减少相当数量。当删除或重组基础表空间以占用更少的空间时，就会发生这种动态调整大小的行为。因此，您可以通过删除不再需要的表和数据库来减少存储费用。动态调整大小适用于某些 Aurora 版本。以下是在删除数据后集群卷会动态调整大小的 Aurora 版本：

数据库引擎	具有动态调整大小功能的版本
Aurora MySQL	• 版本 3（与 MySQL 8.0 兼容）：所有支持的版本 • 版本 2（与 MySQL 5.7 兼容）：2.11 及更高版本
Aurora PostgreSQL	支持的所有版本
Aurora Serverless v2	支持的所有版本

数据库引擎	具有动态调整大小功能的版本
Aurora Serverless v1	支持的所有版本

- 在低于前面所列版本的 Aurora 版本中，集群卷可以重用在删除数据时释放的空间，但卷本身的大小从不会减小。
- 此功能正在分阶段部署到支持 Aurora 的AWS区域。根据集群所在的区域，此功能可能尚不可用。

动态调整大小适用于以物理方式删除集群卷中的表空间或调整其大小的操作。因此，它适用于 `DROP TABLE`、`DROP DATABASE`、`TRUNCATE TABLE` 和 `ALTER TABLE ... DROP PARTITION` 之类的 SQL 语句。它不适用于使用 `DELETE` 语句删除行。如果从表中删除了大量的行，则可以在之后运行 Aurora MySQL `OPTIMIZE TABLE` 语句或使用 Aurora PostgreSQL `pg_repack` 扩展以重组表并动态调整集群卷的大小。

对于 Aurora MySQL，应注意以下事项：

- 将数据库集群升级到支持动态调整大小的数据库引擎版本，并在该特定 AWS 区域中启用此功能后，一些 SQL 语句（例如 `DROP TABLE`）在以后释放的任何空间均可回收。

  如果在特定 AWS 区域中明确禁用了此功能，则该空间可能只能重复使用但不可回收，即使在支持动态调整大小的版本上也是如此。

  此功能在 2020 年 11 月至 2022 年 3 月期间针对特定的数据库引擎版本（1.23.0-1.23.4、2.09.0-2.09.3 和 2.10.0）启用，并在任何后续版本中默认启用。
- 表在内部存储在一个或多个大小可变的连续片段中。运行 `TRUNCATE TABLE` 操作时，与第一个片段对应的空间可重复使用且不可回收。其他片段是可以回收的。在 `DROP TABLE` 操作期间，可以回收与整个表空间相对应的空间。
- `innodb_file_per_table` 参数影响表存储的组织方式。当表是系统表空间的一部分时，删除表不会减小系统表空间的大小。因此，请确保对于 Aurora MySQL 数据库集群将 `innodb_file_per_table` 设置为 1，以充分利用动态调整大小功能。
- 在版本 2.11 及更高版本中，InnoDB 临时表空间将被删除并在重启时重新创建。这会将临时表空间占用的空间释放给系统，然后调整集群卷的大小。为了充分利用动态调整大小功能，我们建议您将数据库集群升级到版本 2.11 或更高版本。

> **Note**
>
> 当删除表空间中的表时，动态调整大小功能不会立即回收空间，而是以每天大约 10TB 的速度逐渐回收空间。不会回收系统表空间中的空间，因为从不会删除系统表空间。当某项操作需要某个表空间中的空间时，该表空间中未回收的空闲空间将被重用。只有当集群处于可用状态时，动态调整大小功能才能回收存储空间。

您可以通过监控 CloudWatch 中的 `VolumeBytesUsed` 指标来检查集群使用的存储空间量。有关存储账单的更多信息，请参阅Aurora 数据存储的计费方式。

- 在 AWS Management Console 中，您可以通过查看集群详细信息页面上的 `Monitoring` 选项卡，在图表中查看此数字。
- 使用 AWS CLI，您可以运行类似于以下 Linux 示例的命令。用您自己的值替换开始和结束时间以及集群的名称。

```
aws cloudwatch get-metric-statistics --metric-name "VolumeBytesUsed" \
  --start-time "$(date -d '6 hours ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" \
  --statistics Average Maximum Minimum \
  --dimensions Name=DBClusterIdentifier,Value=my_cluster_identifier
```

该命令产生的输出类似于以下内容。

```
{
    "Label": "VolumeBytesUsed",
    "Datapoints": [
        {
            "Timestamp": "2020-08-04T21:25:00+00:00",
            "Average": 182871982080.0,
            "Minimum": 182871982080.0,
            "Maximum": 182871982080.0,
            "Unit": "Bytes"
        }
    ]
}
```

以下示例显示了如何在 Linux 系统上使用 AWS CLI 命令，跟踪一段时间内的 Aurora 集群存储使用量。`--start-time` 和 `--end-time` 参数将整个时间间隔定义为一天。`--period` 参数每隔一小时

请求一次测量。选择一个小的 --period 值没有意义，因为这些指标是按间隔收集的，而不是连续收集的。此外，在相关 SQL 语句完成后，Aurora 存储操作有时会在后台继续一段时间。

第一个示例以默认 JSON 格式返回输出。数据点以任意顺序返回，而不是按时间戳排序。您可以将此 JSON 数据导入到图表工具中以进行排序和可视化。

```
$ aws cloudwatch get-metric-statistics --metric-name "VolumeBytesUsed" \
  --start-time "$(date -d '1 day ago')" --end-time "$(date -d 'now')" --period 3600
  --namespace "AWS/RDS" --statistics Maximum --dimensions
 Name=DBClusterIdentifier,Value=my_cluster_id
{
    "Label": "VolumeBytesUsed",
    "Datapoints": [
        {
            "Timestamp": "2020-08-04T19:40:00+00:00",
            "Maximum": 182872522752.0,
            "Unit": "Bytes"
        },
        {
            "Timestamp": "2020-08-05T00:40:00+00:00",
            "Maximum": 198573719552.0,
            "Unit": "Bytes"
        },
        {
            "Timestamp": "2020-08-05T05:40:00+00:00",
            "Maximum": 206827454464.0,
            "Unit": "Bytes"
        },
        {
            "Timestamp": "2020-08-04T17:40:00+00:00",
            "Maximum": 182872522752.0,
            "Unit": "Bytes"
        },
... output omitted ...
```

此示例返回与前一个示例相同的数据。--output 参数以紧凑的纯文本格式表示数据。aws cloudwatch 命令通过管道将其输出传递给 sort 命令。-k 命令的 sort 参数按第三个字段（通用协调时间 (UTC) 格式的时间戳）对输出进行排序。

```
$ aws cloudwatch get-metric-statistics --metric-name "VolumeBytesUsed" \
  --start-time "$(date -d '1 day ago')" --end-time "$(date -d 'now')" --period 3600 \
```

```
  --namespace "AWS/RDS" --statistics Maximum --dimensions
 Name=DBClusterIdentifier,Value=my_cluster_id \
  --output text | sort -k 3
VolumeBytesUsed
DATAPOINTS  182872522752.0  2020-08-04T17:41:00+00:00 Bytes
DATAPOINTS  182872522752.0  2020-08-04T18:41:00+00:00 Bytes
DATAPOINTS  182872522752.0  2020-08-04T19:41:00+00:00 Bytes
DATAPOINTS  182872522752.0  2020-08-04T20:41:00+00:00 Bytes
DATAPOINTS  187667791872.0  2020-08-04T21:41:00+00:00 Bytes
DATAPOINTS  190981029888.0  2020-08-04T22:41:00+00:00 Bytes
DATAPOINTS  195587244032.0  2020-08-04T23:41:00+00:00 Bytes
DATAPOINTS  201048915968.0  2020-08-05T00:41:00+00:00 Bytes
DATAPOINTS  205368492032.0  2020-08-05T01:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T02:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T03:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T04:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T05:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T06:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T07:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T08:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T09:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T10:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T11:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T12:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T13:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T14:41:00+00:00 Bytes
DATAPOINTS  206833664000.0  2020-08-05T15:41:00+00:00 Bytes
DATAPOINTS  206833664000.0  2020-08-05T16:41:00+00:00 Bytes
```

排序后的输出显示在监控周期开始和结束时使用的存储空间量。您还可以找到在此期间 Aurora 为集群分配更多存储空间的时间点。以下示例使用 Linux 命令将 VolumeBytesUsed 开始值和结束值重新格式化为 GB 和 GiB。GB 表示以 10 的幂测量的单位，通常用于讨论旋转硬盘驱动器的存储。千兆字节表示以 2 的幂进行测量的单位。Aurora 存储空间的测量和限制通常以 2 的幂为单位表示，如 GiB 和 TiB。

```
$ GiB=$((1024*1024*1024))
$ GB=$((1000*1000*1000))
$ echo "Start: $((182872522752/$GiB)) GiB, End: $((206833664000/$GiB)) GiB"
Start: 170 GiB, End: 192 GiB
$ echo "Start: $((182872522752/$GB)) GB, End: $((206833664000/$GB)) GB"
Start: 182 GB, End: 206 GB
```

VolumeBytesUsed 指标告诉您集群中有多少存储空间产生费用。因此，最好在可行的情况下最小化此数字。但是，此指标不包括一些 Aurora 在集群内部使用且不收费的存储空间。如果您的集群接近存储限制并且可能会用完空间，则监控 AuroraVolumeBytesLeftTotal 指标并尝试最大化此数字会更有帮助。以下示例运行与前一个示例类似的计算，但针对 AuroraVolumeBytesLeftTotal 而不是针对 VolumeBytesUsed。

```
$ aws cloudwatch get-metric-statistics --metric-name "AuroraVolumeBytesLeftTotal" \
  --start-time "$(date -d '1 hour ago')" --end-time "$(date -d 'now')" --period 3600 \
  --namespace "AWS/RDS" --statistics Maximum --dimensions
 Name=DBClusterIdentifier,Value=my_old_cluster_id \
  --output text | sort -k 3
AuroraVolumeBytesLeftTotal
DATAPOINTS      140530528288768.0       2023-02-23T19:25:00+00:00       Count
$ TiB=$((1024*1024*1024*1024))
$ TB=$((1000*1000*1000*1000))
$ echo "$((69797067915264 / $TB)) TB remaining for this cluster"
69 TB remaining for this cluster
$ echo "$((69797067915264 / $TiB)) TiB remaining for this cluster"
63 TiB remaining for this cluster
```

对于运行 Aurora MySQL 版本 2.09 或更高版本或 Aurora PostgreSQL 的集群，添加数据时 VolumeBytesUsed 报告的可用大小会增大，删除数据时会减小。下面的示例演示如何操作。在创建和删除包含临时数据的表时，此报告每隔 15 分钟显示一次集群的最大和最小存储大小。此报告在最小值之前列出最大值。因此，要了解在 15 分钟间隔内存储使用量如何变化，请从右向左解释这些数字。

```
$ aws cloudwatch get-metric-statistics --metric-name "VolumeBytesUsed" \
  --start-time "$(date -d '4 hours ago')" --end-time "$(date -d 'now')" --period 1800 \
  --namespace "AWS/RDS" --statistics Maximum Minimum --dimensions
 Name=DBClusterIdentifier,Value=my_new_cluster_id
  --output text | sort -k 4
VolumeBytesUsed
DATAPOINTS 14545305600.0 14545305600.0 2020-08-05T20:49:00+00:00 Bytes
DATAPOINTS 14545305600.0 14545305600.0 2020-08-05T21:19:00+00:00 Bytes
DATAPOINTS 22022176768.0 14545305600.0 2020-08-05T21:49:00+00:00 Bytes
DATAPOINTS 22022176768.0 22022176768.0 2020-08-05T22:19:00+00:00 Bytes
DATAPOINTS 22022176768.0 22022176768.0 2020-08-05T22:49:00+00:00 Bytes
DATAPOINTS 22022176768.0 15614263296.0 2020-08-05T23:19:00+00:00 Bytes
DATAPOINTS 15614263296.0 15614263296.0 2020-08-05T23:49:00+00:00 Bytes
DATAPOINTS 15614263296.0 15614263296.0 2020-08-06T00:19:00+00:00 Bytes
```

以下示例显示了对于运行 Aurora MySQL 版本 2.09 或更高版本或 Aurora PostgreSQL 的集群，AuroraVolumeBytesLeftTotal 报告的可用大小如何反映较高的 128TiB 大小限制。

```
$ aws cloudwatch get-metric-statistics --region us-east-1 --metric-name
 "AuroraVolumeBytesLeftTotal" \
  --start-time "$(date -d '4 hours ago')" --end-time "$(date -d 'now')" --period 1800 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBClusterIdentifier,Value=pq-57 \
  --output text | sort -k 3
AuroraVolumeBytesLeftTotal
DATAPOINTS 140515818864640.0 2020-08-05T20:56:00+00:00 Count
DATAPOINTS 140515818864640.0 2020-08-05T21:26:00+00:00 Count
DATAPOINTS 140515818864640.0 2020-08-05T21:56:00+00:00 Count
DATAPOINTS 140514866757632.0 2020-08-05T22:26:00+00:00 Count
DATAPOINTS 140511020580864.0 2020-08-05T22:56:00+00:00 Count
DATAPOINTS 140503168843776.0 2020-08-05T23:26:00+00:00 Count
DATAPOINTS 140503168843776.0 2020-08-05T23:56:00+00:00 Count
DATAPOINTS 140515818864640.0 2020-08-06T00:26:00+00:00 Count
$ TiB=$((1024*1024*1024*1024))
$ TB=$((1000*1000*1000*1000))
$ echo "$((140515818864640 / $TB)) TB remaining for this cluster"
140 TB remaining for this cluster
$ echo "$((140515818864640 / $TiB)) TiB remaining for this cluster"
127 TiB remaining for this cluster
```

实例扩展

可通过修改数据库集群中每个数据库实例的数据库实例类来按需扩展 Aurora 数据库集群。Aurora 支持多个针对 Aurora 进行优化的数据库实例类，具体情况取决于数据库引擎的兼容性。

数据库引擎	实例扩展
Amazon Aurora MySQL	请参阅 扩展 Aurora MySQL 数据库实例
Amazon Aurora PostgreSQL	请参阅 扩展 Aurora PostgreSQL 数据库实例

读取扩展

可通过在 Aurora 数据库集群中创建最多 15 个 Aurora 副本，以实现针对该数据库集群的读取扩缩。每个 Aurora 副本均返回集群卷中的相同数据，且副本滞后时间最短 — 通常大大少于主实例写入更新后

的 100 毫秒。当读取流量增大时，可创建额外 Aurora 副本并直接连接到这些副本以为您的数据库集群分配读取负载。Aurora 副本不必具有与主实例相同的数据库实例类。

有关将 Aurora 副本添加到数据库集群的信息，请参阅将 Aurora 副本添加到数据库集群。

管理连接

允许连接到 Aurora 数据库实例的最大数量由数据库实例的实例级参数组中的 `max_connections` 参数确定。参数的默认值因用于数据库实例的数据库实例类和数据库引擎兼容性而异。

数据库引擎	max_connections 默认值
Amazon Aurora MySQL	请参阅 至 Aurora MySQL 数据库实例的最大连接数
Amazon Aurora PostgreSQL	请参阅 至 Aurora PostgreSQL 数据库实例的最大连接数

> **Tip**
>
> 如果您的应用程序经常打开和关闭连接，或使大量长期连接保持打开，我们建议您使用 Amazon RDS 代理。RDS 代理是一种完全托管的高可用性数据库代理，它使用连接池安全有效地共享数据库连接。要了解有关 RDS 代理的更多信息，请参阅适用于 Aurora 的 Amazon RDS 代理。

管理查询执行计划

如果使用 Aurora PostgreSQL 的查询计划管理，则可以控制优化程序运行的计划。有关更多信息，请参阅“管理 Aurora PostgreSQL 的查询执行计划”。

克隆 Amazon Aurora 数据库集群卷

使用 Aurora 克隆功能，您可以创建一个新集群，该集群最初与原始集群共享相同的数据页，但它是一个单独且独立的卷。该过程旨在快速且经济高效。我们将新集群及其关联的数据卷称为克隆。与使用其他技术（如还原快照）实际复制数据相比，创建克隆速度更快且空间利用效率更高。

主题

- Aurora 克隆概述
- Aurora 克隆的限制
- Aurora 克隆的工作原理
- 创建 Amazon Aurora 克隆
- 使用 Amazon Aurora 的跨 VPC 克隆
- 使用 AWS RAM 与 Amazon Aurora 进行跨账户克隆

Aurora 克隆概述

Aurora 使用写入时复制协议创建克隆。此机制占用最少的额外空间来创建初始克隆。首次创建克隆时，Aurora 会保留源 Aurora 数据库集群和新（克隆的）Aurora 数据库集群使用的数据的单个副本。只有当源 Aurora 数据库集群或 Aurora 数据库集群克隆对数据（在 Aurora 存储卷上）进行更改时，才会分配额外的存储空间。如需了解有关写入时复制协议的更多信息，请参阅 Aurora 克隆的工作原理。

Aurora 克隆非常适合使用您的生产数据快速设置测试环境，且不会有损坏数据的风险。您可以将克隆用于多种类型的应用程序，例如：

- 对潜在的变化（例如模式变化和参数组变化）进行试验，以评估所有影响。
- 执行工作负载密集型操作，例如导出数据或在克隆上运行分析查询。
- 为开发、测试或其他用途创建生产数据库集群的副本。

您可以从同一个 Aurora 数据库集群创建多个克隆。您还可以从另一个克隆创建多个克隆。

创建 Aurora 克隆后，您可以以不同于源 Aurora 数据库集群的方式配置 Aurora 数据库实例。例如，您可能不需要用于开发目的的克隆来满足与源生产 Aurora 数据库集群相同的高可用性要求。在这种情况下，您可以使用单个 Aurora 数据库实例来配置克隆，而不是使用 Aurora 数据库集群使用的多个数据库实例。

当您使用与源不同的部署配置创建克隆时，将使用源的 Aurora 数据库引擎的最新次要版本创建克隆。

当您从 Aurora 数据库集群创建克隆时，将在您的AWS账户中创建克隆，该账户即是拥有源 Aurora 数据库集群的账户。但是，您也可以与其它 AWS 账户共享 Aurora Serverless v2 和预调配 Aurora 数据库集群和克隆。有关更多信息，请参阅 使用 AWS RAM 与 Amazon Aurora 进行跨账户克隆。

当克隆完成测试、开发等使用目的时，您可以将其删除。

Aurora 克隆的限制

Aurora 克隆具有以下限制：

- 您可以根据需要创建任意数量的克隆，最多为 AWS 区域中允许的最大数据库集群数。
- 您可以使用写入时复制协议创建多达 15 个克隆。在具有 15 个克隆后，您创建的下一个克隆是完整副本。完全复制协议的作用类似于时间点恢复。
- 您不能在与源 Aurora 数据库集群不同的 AWS 区域中创建克隆。
- 不支持并行查询功能的 Aurora 数据库集群无法创建克隆到使用并行查询的集群。要将数据传输到使用并行查询的集群，请创建原始集群的快照，并将其还原到使用并行查询功能的集群。
- 您无法从没有数据库实例的 Aurora 数据库集群创建克隆。您只能克隆具有至少一个数据库实例的 Aurora 数据库集群。
- 您可以在与 Aurora 数据库集群不同的 Virtual Private Cloud (VPC) 中创建克隆。不过，这些 VPC 的子网必须映射到相同的可用区。
- 您可以从预置的 Aurora 数据库集群创建 Aurora 预置的克隆。
- 具有 Aurora Serverless v2 实例的集群遵循与预置集群相同的规则。
- 对于 Aurora Serverless v1：
  - 您可以从 Aurora Serverless v1 数据库集群创建预调配克隆。
  - 您可以从 Aurora Serverless v1 或预调配数据库集群创建 Aurora Serverless v1 克隆。
  - 您无法从未加密的预调配 Aurora 数据库集群创建 Aurora Serverless v1 克隆。
  - 跨账户克隆目前不支持克隆 Aurora Serverless v1 数据库集群。有关更多信息，请参阅 跨账户克隆的限制。
  - 克隆的Aurora Serverless v1 数据库集群与其他 Aurora Serverless v1 数据库集群具有相同的行为和限制。有关更多信息，请参阅 使用 Amazon Aurora Serverless v1。
  - Aurora Serverless v1 数据库集群始终加密。当您将 Aurora Serverless v1 数据库集群克隆到预置的 Aurora 数据库集群时，配置的 Aurora 数据库集群将被加密。您可以选择加密密钥，但不能禁用加密。要从预调配 Aurora 数据库集群克隆到 Aurora Serverless v1，必须以加密的预调配 Aurora 数据库集群开始。

Aurora 克隆的工作原理

Aurora 克隆运行于 Aurora 数据库集群的存储层。它使用写入时复制协议，该协议在支持 Aurora 存储卷的底层持久介质方面既快速又节省空间。有关 Aurora 集群卷的更多信息，请参阅 Amazon Aurora 存储概述。

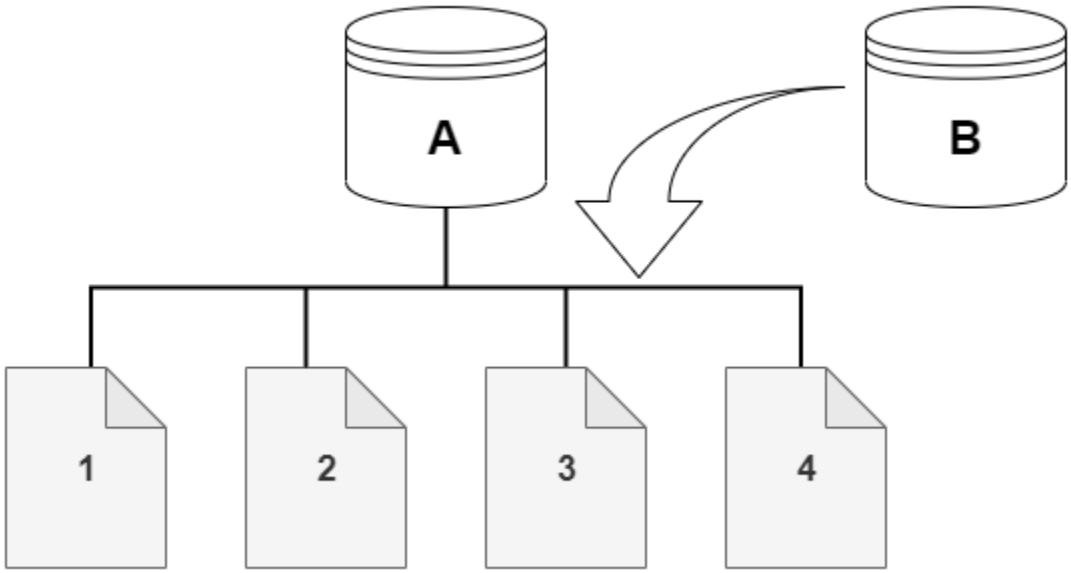
主题

- 了解写入时复制协议
- 删除源集群卷

了解写入时复制协议

Aurora 数据库集群将数据存储在底层 Aurora 存储卷中的页面中。

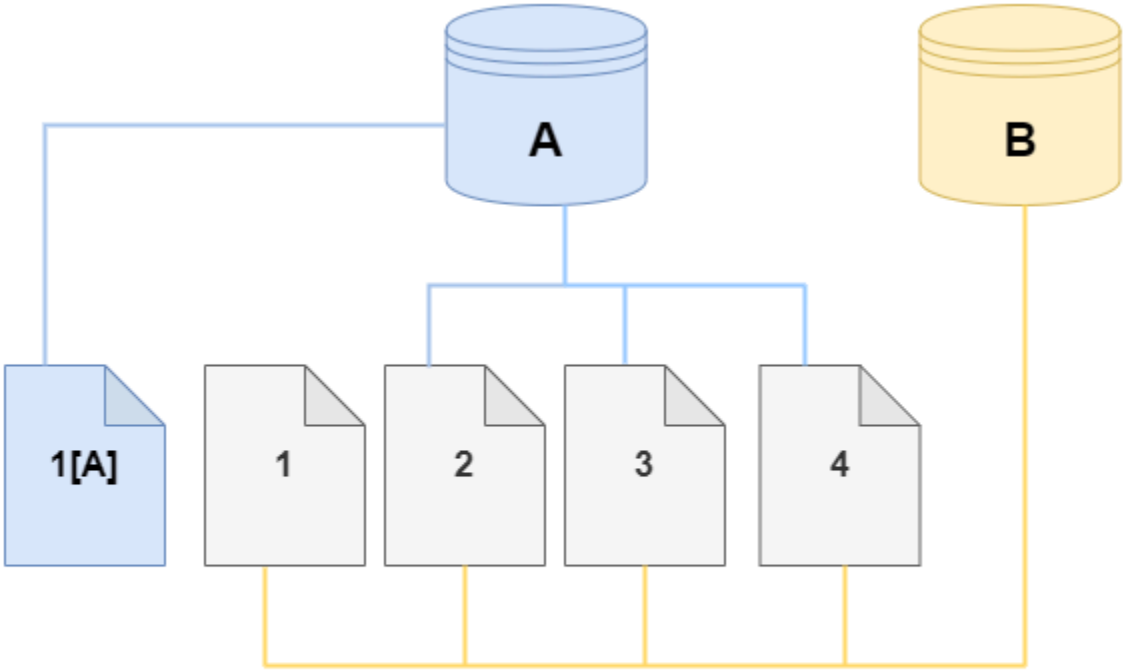
例如，在下图中，您可以找到拥有四个数据页（1、2、3 和 4）的 Aurora 数据库集群（A）。假设从 Aurora 数据库集群创建了一个克隆 B。创建克隆时，未复制任何数据。相反，克隆指向与源 Aurora 数据库集群相同的页面集。



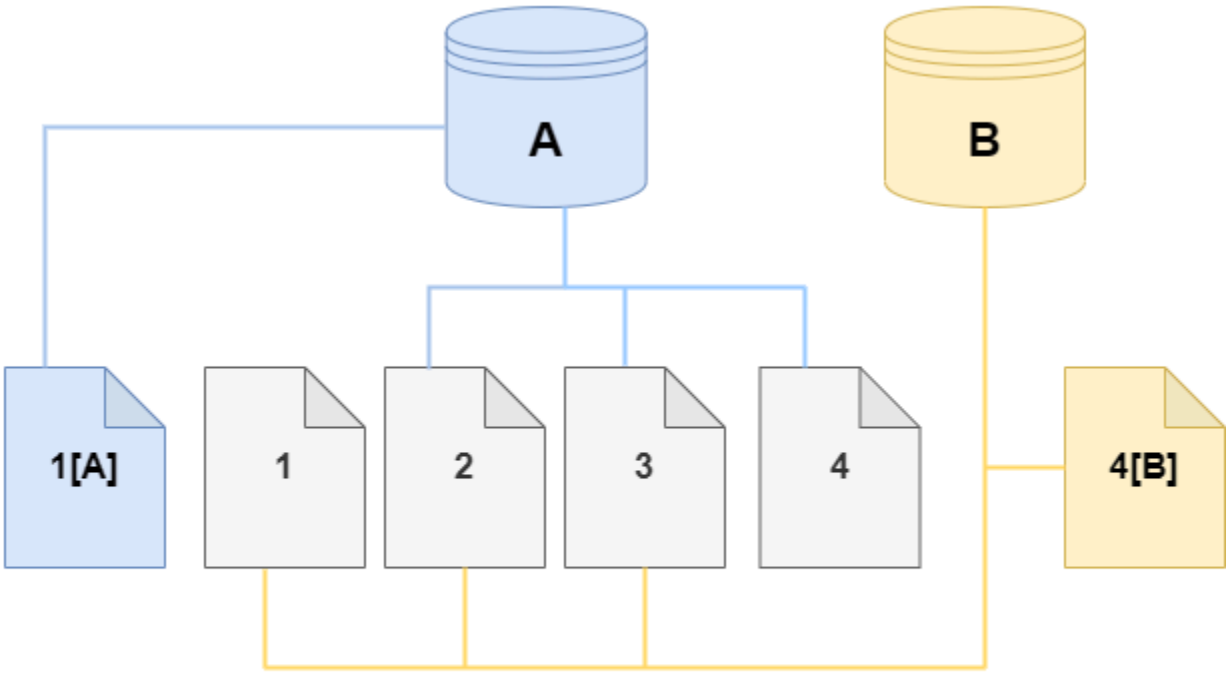
创建克隆时，通常不需要额外的存储空间。写入时复制协议在物理存储介质上使用与源段相同的数据段。只有当源段的容量不足以容纳整个克隆段时，才需要额外的存储空间。如果是这种情况，源段将被复制到另一个物理设备。

在下图中，您可以找到使用相同集群 A 及其克隆 B 的写入操作时复制协议的示例，如前所示。如果您对 Aurora 数据库集群（A）进行更改，那么第 1 页上保存的数据也将随之而发生改变。Aurora 没

有写入原始页面 1，而是创建了一个新页面 1[A]。集群（A）的 Aurora 数据库集群卷现在指向页面 1[A]、2、3 和 4，而克隆（B）仍引用原始页面。



在克隆上，对存储卷的第 4 页进行了更改。Aurora 没有写入原始页面 4，而是创建了一个新页面 4[B]。克隆现在指向页面 1、2、3 和页面 4[B]，而集群（A）继续指向 1[A]、2、3 和 4。



随着时间推移，当源 Aurora 集群卷和克隆上出现了更多更改时，因此需要更多存储空间来捕获和存储更改。

删除源集群卷

最初，克隆卷与从中创建克隆的原始卷共享相同的数据页面。只要原始卷存在，克隆卷就只能被视为克隆创建或修改的页面的拥有者。因此，克隆卷的 `VolumeBytesUsed` 指标起初很小，只会随着原始集群和克隆之间的数据差异而增长。对于源卷和克隆之间相同的页面，存储费用仅适用于原始集群。有关 `VolumeBytesUsed` 指标的更多信息，请参阅Amazon Aurora 的集群级指标。

删除与一个或多个克隆关联的源集群卷时，不会更改克隆的集群卷中的数据。Aurora 保留以前由源集群卷拥有的页面。Aurora 会重新分配已删除集群所拥有的页面的存储计费。例如，假设一个原始集群有两个克隆，然后删除了原始集群。现在，原始集群拥有的数据页面中有一半将归一个克隆所拥有。另一半页面将归另一个克隆所拥有。

如果您删除原始集群，则当您创建或删除更多克隆时，Aurora 将继续在共享相同页面的所有克隆间重新分配数据页面的拥有权。因此，您可能会发现，克隆的集群卷的 `VolumeBytesUsed` 指标值发生变化。随着创建的克隆越来越多以及页面拥有权分布在更多集群中，此指标的值可能会降低。随着删除克

隆以及页面拥有权分配给更少数量的集群，此指标的值也可能增加。有关写入操作如何影响克隆卷上的数据页面的信息，请参阅了解写入时复制协议。

当原始集群和克隆归同一个 AWS 账户拥有时，这些集群的所有存储费用都适用于该同一个 AWS 账户。如果某些集群是跨账户克隆，则删除原始集群可能会导致对拥有跨账户克隆的 AWS 账户收取额外的存储费用。

例如，假设一个集群卷在您创建任何克隆之前具有 1000 个已用数据页面。克隆该集群时，最初克隆卷的已用页面数为零。如果克隆对 100 个数据页面进行修改，则只有这 100 个页面会存储在克隆卷上并标记为已用。父卷中其它 900 个未更改的页面由这两个集群共享。在这种情况下，将对父集群收取 1000 个页面的存储费用，而对克隆卷收取 100 个页面的存储费用。

如果您删除源卷，则克隆的存储费用包括它更改的 100 个页面，再加上原始卷中的 900 个共享页面，总共 1000 个页面。

创建 Amazon Aurora 克隆

您可以在与源 Aurora 数据库集群相同的AWS账户中创建克隆。为此，您可以使用 AWS Management Console 或 AWS CLI 并按照以下步骤进行操作。

若要允许其他 AWS 账户创建克隆或与其他 AWS 账户共享克隆，请遵照 使用 AWS RAM 与 Amazon Aurora 进行跨账户克隆 中的步骤。

控制台

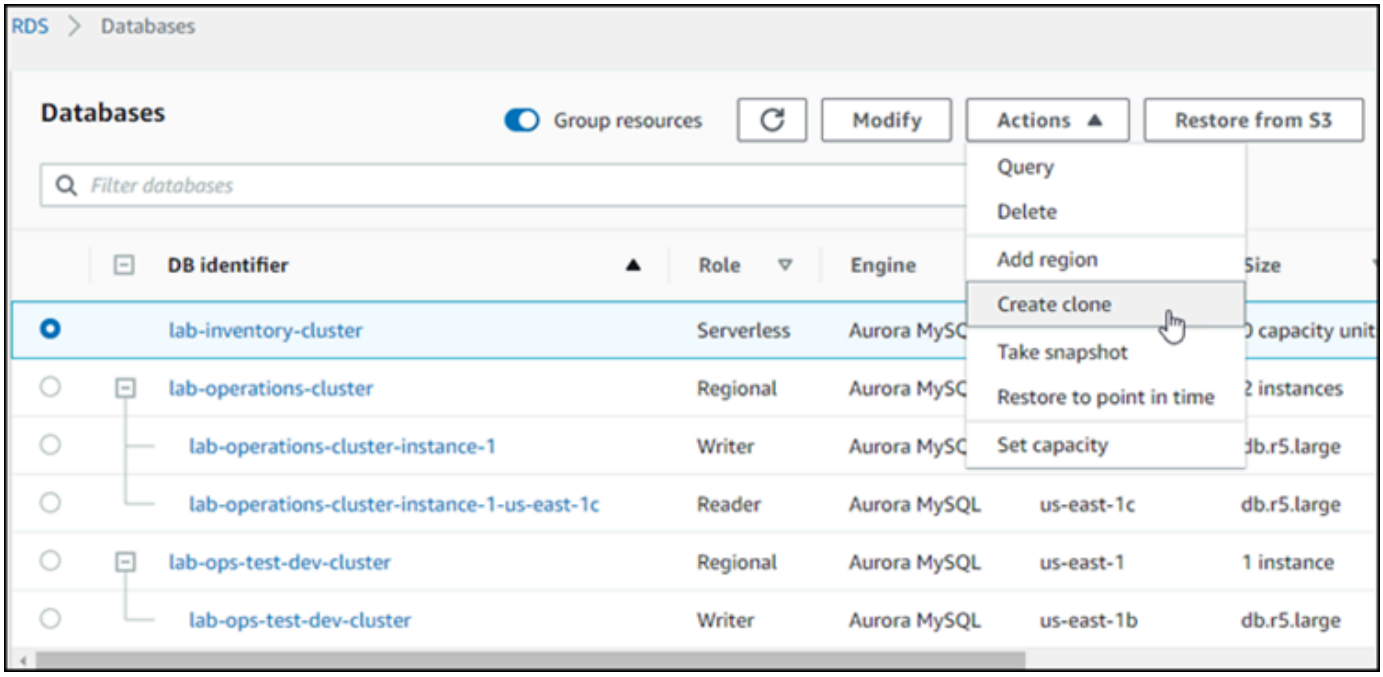
以下过程介绍了如何使用 AWS Management Console 克隆 Aurora 数据库集群。

在含有一个 Aurora 数据库实例的 Aurora 数据库集群中使用 AWS Management Console 结果创建克隆。

这些说明适用于创建克隆的同一 AWS 账户所拥有的数据库集群。如果数据库集群由其他 AWS 账户拥有，请改为参阅 使用 AWS RAM 与 Amazon Aurora 进行跨账户克隆。

使用 AWS 创建 AWS Management Console 账户拥有的数据库集群的克隆

1. 登录AWS Management Console并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Databases (数据库)。
3. 从列表中选择您的 Aurora 数据库集群，从 Actions (操作) 中选择 Create clone (创建克隆)。



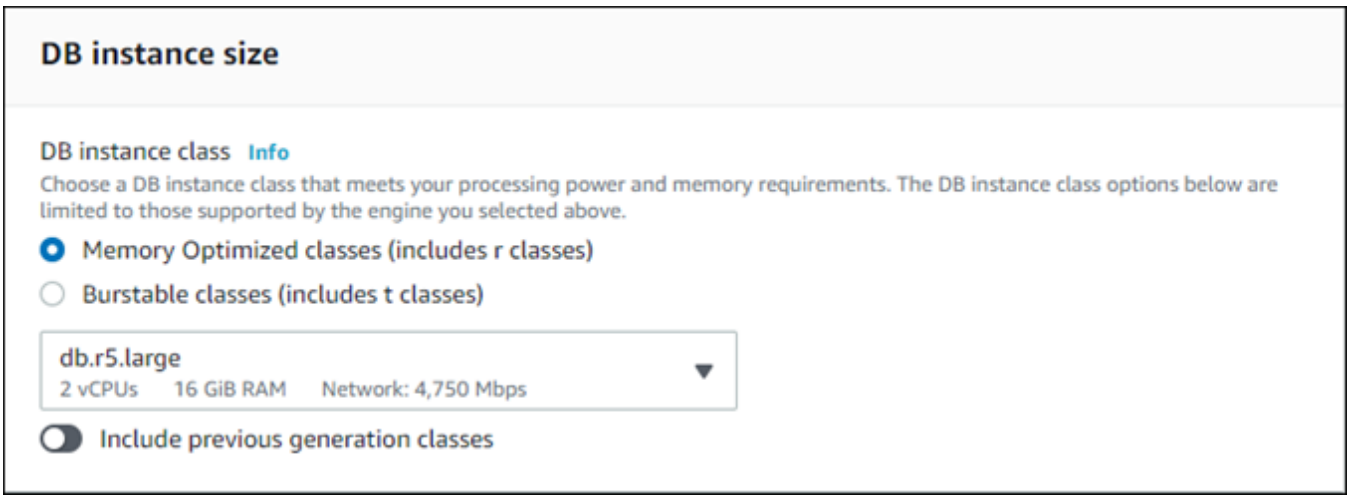
“创建克隆”页面打开后，您可以配置 Aurora 数据库集群克隆的设置、连接和其他选项。

4. 对于数据库实例标识符，请输入您要为克隆的 Aurora 数据库集群拟定的名称。
5. 对于 Aurora Serverless v1 数据库集群，请为容量类型选择预调配或无服务器。

   仅当源 Aurora 数据库集群是 Aurora Serverless v1 数据库集群或者是加密的预置 Aurora 数据库集群时，您才可以选择 Serverless (无服务器)。
6. 对于 Aurora Serverless v2 或预调配数据库集群，请为集群存储配置选择 Aurora I/O-Optimized 或 Aurora Standard。

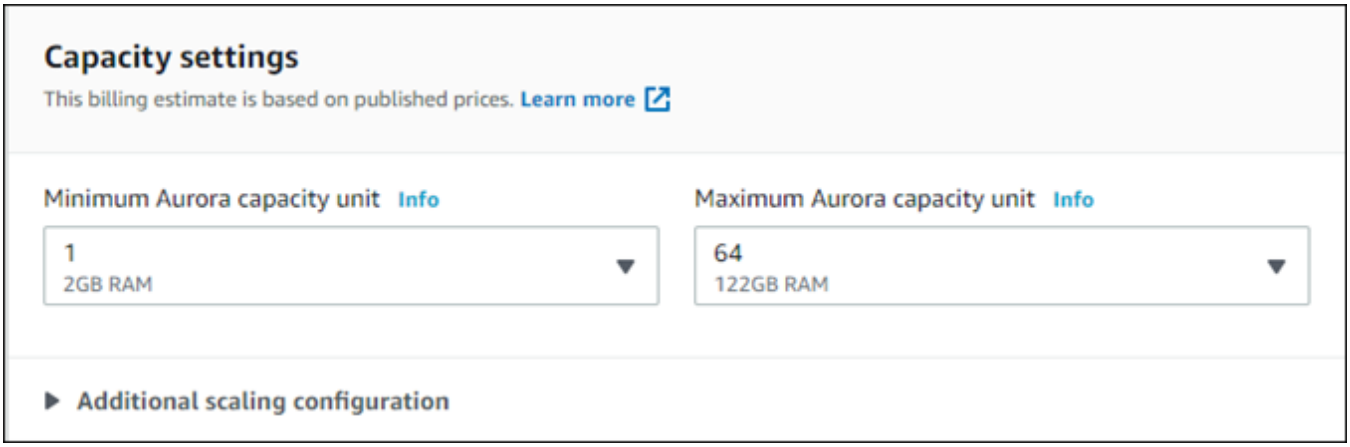
   有关更多信息，请参阅 Amazon Aurora 数据库集群的存储配置。
7. 选择数据库实例大小或数据库集群容量：

   - 对于预调配克隆，请选择数据库实例类。



您可以使用所提供的设置，也可以为克隆使用不同的数据库实例类。

- 对于 Aurora Serverless v1 或 Aurora Serverless v2 克隆，请选择容量设置。



您可以使用提供的设置，也可以根据您的克隆更改此类设置。

8. 根据需要为克隆选择其它设置。要了解有关 Aurora 数据库集群和实例设置的更多信息，请参阅 创建 Amazon Aurora 数据库集群。
9. 选择创建克隆。

克隆创建完成后，它将会与您的其他 Aurora 数据库集群一起列在控制台 Databases (数据库) 部分，而且其当前状态也会一起显示。当其状态为可用时，您的克隆即可以使用。

AWS CLI

使用 AWS CLI 克隆 Aurora 数据库集群涉及用于创建克隆集群和向集群添加一个或多个数据库实例的单独步骤。

您使用的 `restore-db-cluster-to-point-in-time` AWS CLI 命令将导致 Aurora 数据库集群具有与原始集群相同的存储数据，但没有 Aurora 数据库实例。在克隆可用之后，可以单独创建数据库实例。可以选择数据库实例的数量及其实例类别，来为克隆指定多于或者少于原始集群的计算容量。该过程的步骤如下所示：

1. 使用 restore-db-cluster-to-point-in-time CLI 命令创建克隆。
2. 使用 create-db-instance CLI 命令创建克隆的写入器数据库实例。
3. （可选）再运行一个 create-db-instance CLI 命令，用于将一个或多个读取器实例添加到克隆集群。使用读取器实例有助于提高克隆的高可用性和读取可扩展性。如果只要将克隆用于开发和测试，则可以跳过此步骤。

主题

- 创建克隆
- 检查状态并获取克隆的详细信息
- 为克隆创建 Aurora 数据库实例
- 用于克隆的参数

创建克隆

使用 `restore-db-cluster-to-point-in-time` CLI 命令创建初始克隆集群。

从源 Aurora 数据库集群创建克隆

- 使用 `restore-db-cluster-to-point-in-time` CLI 命令。为以下参数指定值。在这种典型情况下，克隆使用与原始集群相同的引擎模式：要么是预置，要么是 Aurora Serverless v1。
  - `--db-cluster-identifier` – 为克隆选择一个有意义的名称。使用 restore-db-cluster-to-point-in-time CLI 命令命名克隆。然后在 create-db-instance CLI 命令中传递克隆的名称。
  - `--restore-type` – 使用 `copy-on-write` 创建源数据库集群的克隆。如果没有此参数，`restore-db-cluster-to-point-in-time` 将还原 Aurora 数据库集群，而不会创建克隆。
  - `--source-db-cluster-identifier` – 使用要克隆的源 Aurora 数据库集群的名称。
  - `--use-latest-restorable-time` – 此值指向源数据库集群的最新可还原卷数据。使用它来创建克隆。

以下示例从名为 my-source-cluster 的集群创建一个名为 my-clone 的克隆。

对于 Linux、macOS 或 Unix：

```
aws rds restore-db-cluster-to-point-in-time \
    --source-db-cluster-identifier my-source-cluster \
    --db-cluster-identifier my-clone \
    --restore-type copy-on-write \
    --use-latest-restorable-time
```

对于 Windows：

```
aws rds restore-db-cluster-to-point-in-time ^
    --source-db-cluster-identifier my-source-cluster ^
    --db-cluster-identifier my-clone ^
    --restore-type copy-on-write ^
    --use-latest-restorable-time
```

该命令返回包含克隆详细信息的 JSON 对象。在尝试为您的克隆创建数据库实例之前，请检查以确保您的克隆数据库集群可用。有关更多信息，请参阅 检查状态并获取克隆的详细信息。

例如，假设您有一个名为 tpch100g 的集群需要克隆。下面的 Linux 示例创建了一个名为 tpch100g-clone 的克隆集群，并为该新集群创建了一个名为 tpch100g-clone-instance 的 Aurora Serverless v2 写入器实例和一个名为 tpch100g-clone-instance-2 的预置读取器实例。

有些参数并不需要提供，例如 --master-username 和 --master-user-password。Aurora 会自动从原始集群中确定那些参数。您需要指定的是要使用的数据库引擎。因此，该示例测试新集群以确定要用于 --engine 参数的正确值。

此示例还在创建克隆集群时包括 --serverless-v2-scaling-configuration 选项。这样，即使原始集群未使用 Aurora Serverless v2，您也可以向克隆中添加 Aurora Serverless v2 实例。

```
$ aws rds restore-db-cluster-to-point-in-time \
  --source-db-cluster-identifier tpch100g \
  --db-cluster-identifier tpch100g-clone \
  --serverless-v2-scaling-configuration MinCapacity=0.5,MaxCapacity=16 \
  --restore-type copy-on-write \
  --use-latest-restorable-time

$ aws rds describe-db-clusters \
  --db-cluster-identifier tpch100g-clone \
    --query '*[].[Engine]' \
```

```
    --output text
aurora-mysql

$ aws rds create-db-instance \
  --db-instance-identifier tpch100g-clone-instance \
  --db-cluster-identifier tpch100g-clone \
  --db-instance-class db.serverless \
  --engine aurora-mysql

$ aws rds create-db-instance \
  --db-instance-identifier tpch100g-clone-instance-2 \
  --db-cluster-identifier tpch100g-clone \
  --db-instance-class db.r6g.2xlarge \
  --engine aurora-mysql
```

使用与源 Aurora 数据库集群不同的引擎模式创建克隆

- 此过程仅适用于支持 Aurora Serverless v1 的较旧引擎版本。假设您有一个 Aurora Serverless v1 集群，并且想要创建一个作为预置集群的克隆。在这种情况下，请使用 `restore-db-cluster-to-point-in-time` CLI 命令，并指定与上一个示例中的参数值类似的值，另加以下附加参数：
  - `--engine-mode` – 此参数仅用于创建与源 Aurora 数据库集群属于不同引擎模式的克隆。此参数仅适用于支持 Aurora Serverless v1 的较旧引擎版本。选择与 `--engine-mode` 一起传递的值，如下所示：
    - 使用 `--engine-mode provisioned` 从 Aurora Serverless 数据库集群创建预置 Aurora 数据库集群克隆。

      > **Note**
      >
      > 如果您打算将 Aurora Serverless v2 与从 Aurora Serverless v1 中克隆的集群一起使用，则仍将克隆的引擎模式指定为 `provisioned`。接着，您可以随后执行其它升级和迁移步骤。

    - 使用 `--engine-mode serverless` 从预置 Aurora 数据库集群创建 Aurora Serverless v1 克隆。当您指定 `serverless` 引擎模式时，您也可以选择 `--scaling-configuration`。
  - `--scaling-configuration` – （可选）与 `--engine-mode serverless` 一起使用以配置 Aurora Serverless v1 克隆的最小和最大容量。如果不使用此参数，Aurora 将使用数据库引擎的默认 Aurora Serverless v1 容量值创建 Aurora Serverless v1 克隆。

以下示例从名为 `my-source-cluster` 的 Aurora Serverless v1 数据库集群创建名为 `my-clone` 的预置克隆。

对于 Linux、macOS 或 Unix：

```
aws rds restore-db-cluster-to-point-in-time \
    --source-db-cluster-identifier my-source-cluster \
    --db-cluster-identifier my-clone \
    --engine-mode provisioned \
    --restore-type copy-on-write \
    --use-latest-restorable-time
```

对于 Windows：

```
aws rds restore-db-cluster-to-point-in-time ^
    --source-db-cluster-identifier my-source-cluster ^
    --db-cluster-identifier my-clone ^
    --engine-mode provisioned ^
    --restore-type copy-on-write ^
    --use-latest-restorable-time
```

这些命令返回包含创建数据库实例所需的克隆的详细信息的 JSON 对象。在克隆（空的 Aurora 数据库集群）的状态变为可用之前，您不能这样做。

> **Note**
>
> restore-db-cluster-to-point-in-time AWS CLI 命令仅还原数据库集群，而不还原该数据库集群的数据库实例。要为还原的数据库集群创建数据库实例，请运行 create-db-instance 命令。使用该命令，可以将还原的数据库集群的标识符指定为 `--db-cluster-identifier` 参数。只有在完成 `restore-db-cluster-to-point-in-time` 命令并且数据库集群可用后，您才能创建数据库实例。
>
> 假设您从一个 Aurora Serverless v1 集群开始，并打算将其迁移到 Aurora Serverless v2 集群。作为迁移的初始步骤，可以创建 Aurora Serverless v1 集群的预置克隆。有关完整过程，包括任何必需的版本升级，请参阅从 Aurora Serverless v1 集群升级到 Aurora Serverless v2。

检查状态并获取克隆的详细信息

可以使用以下命令来检查新创建的克隆集群的状态。

```
$ aws rds describe-db-clusters --db-cluster-identifier my-clone --query '*[].[Status]'
 --output text
```

或者，您可以使用下面的 AWS CLI 查询获取为您的克隆创建数据库实例所需的状态和其他值。

对于 Linux、macOS 或 Unix：

```
aws rds describe-db-clusters --db-cluster-identifier my-clone \
  --query '*[].
{Status:Status,Engine:Engine,EngineVersion:EngineVersion,EngineMode:EngineMode}'
```

对于 Windows：

```
aws rds describe-db-clusters --db-cluster-identifier my-clone ^
  --query "*[].
{Status:Status,Engine:Engine,EngineVersion:EngineVersion,EngineMode:EngineMode}"
```

此查询返回类似于下述信息的输出：

```
[
  {
        "Status": "available",
        "Engine": "aurora-mysql",
        "EngineVersion": "8.0.mysql_aurora.3.04.1",
        "EngineMode": "provisioned"
    }
]
```

为克隆创建 Aurora 数据库实例

使用 create-db-instance CLI 命令为您的 Aurora Serverless v2 或预调配克隆创建数据库实例。您不会为 Aurora Serverless v1 克隆创建数据库实例。

数据库实例从源数据库集群继承 `--master-username` 和 `--master-user-password` 属性。

以下示例为预调配克隆创建数据库实例。

对于 Linux、macOS 或 Unix：

```
aws rds create-db-instance \
    --db-instance-identifier my-new-db \
    --db-cluster-identifier my-clone \
```

```
    --db-instance-class db.r6g.2xlarge \
    --engine aurora-mysql
```

对于 Windows：

```
aws rds create-db-instance ^
    --db-instance-identifier my-new-db ^
    --db-cluster-identifier my-clone ^
    --db-instance-class db.r6g.2xlarge ^
    --engine aurora-mysql
```

以下示例为克隆（此克隆使用支持 Aurora Serverless v2 的引擎版本）创建 Aurora Serverless v2 数据库实例。

对于 Linux、macOS 或 Unix：

```
aws rds create-db-instance \
    --db-instance-identifier my-new-db \
    --db-cluster-identifier my-clone \
    --db-instance-class db.serverless \
  --engine aurora-postgresql
```

对于 Windows：

```
aws rds create-db-instance ^
    --db-instance-identifier my-new-db ^
    --db-cluster-identifier my-clone ^
    --db-instance-class db.serverless ^
    --engine aurora-mysql
```

用于克隆的参数

下表总结了与 `restore-db-cluster-to-point-in-time` 一起用于克隆 Aurora 数据库集群的各种参数。

参数	描述
`--source-db-cluster-identifier`	使用要克隆的源 Aurora 数据库集群的名称。

参数	描述
`--db-cluster-identifier`	使用 `restore-db-cluster-to-point-in-time` 命令创建克隆时，为克隆选择一个有意义的名称。然后将此名称传递给 `create-db-instance` 命令。
`--restore-type`	将 `copy-on-write` 指定为 `--restore-type` 以创建源数据库集群的克隆，而不是还原源 Aurora 数据库集群。
`--use-latest-restorable-time`	此值指向源数据库集群的最新可还原卷数据。使用它来创建克隆。
`--serverless-v2-scaling-configuration`	（支持 Aurora Serverless v2 的较新版本）使用此参数可配置 Aurora Serverless v2 克隆的最小和最大容量。如果未指定此参数，则在修改集群来添加此属性之前，无法在克隆集群中创建任何 Aurora Serverless v2 实例。
`--engine-mode`	（仅限支持 Aurora Serverless v1 的较旧版本）使用此参数创建与源 Aurora 数据库集群类型不同的克隆，采用以下值之一： • 使用 `provisioned` 从 Aurora Serverless v1 数据库集群创建预置克隆。 • 使用 `serverless` 从预调配或 Aurora Serverless v2 数据库集群创建 Aurora Serverless v1 克隆。 当您指定 `serverless` 引擎模式时，您也可以选择 `--scaling-configuration` 。
`--scaling-configuration`	（仅限支持 Aurora Serverless v1 的较旧版本）使用此参数可配置 Aurora Serverless v1 克隆的最小和最大容量。如果不指定此参数，Aurora 将使用数据库引擎的默认容量值创建克隆。

有关跨 VPC 和跨账户克隆的信息，请参阅以下各节。

主题

- 使用 Amazon Aurora 的跨 VPC 克隆
- 使用 AWS RAM 与 Amazon Aurora 进行跨账户克隆

使用 Amazon Aurora 的跨 VPC 克隆

假设您要对原始集群和克隆群施加不同的网络访问控制。例如，使用克隆功能，您可以在不同 VPC 中创建生产 Aurora 集群的副本，以供开发和测试之用。或者，在从公有子网迁移到私有子网的过程中，您也可以创建克隆，用于增强数据库的安全性。

以下各节演示如何为克隆设置网络配置，使得原始集群和克隆集群都可以访问相同的 Aurora 存储节点，即使这些节点来自不同的子网或不同的 VPC，也可以访问。预先验证网络资源，可以避免在克隆过程中出现可能难以诊断的错误。

如果您不熟悉 Aurora 如何与 VPC、子网和数据库子网组交互，请先参阅 Amazon VPC 和 Amazon Aurora。您可以按照该部分中的教程，在 AWS 管理控制台中创建这些类型的资源，并了解这些资源如何组合在一起。

由于这些步骤涉及在 RDS 和 EC2 服务之间切换，因此示例使用 AWS 命令行界面命令，来帮助您了解如何自动执行此类操作并保存输出。

主题

- 开始前的准备工作
- 收集有关网络环境的信息
- 为克隆集群创建网络资源
- 使用新的网络设置创建 Aurora 克隆
- 将集群从公有子网移动到私有子网
- 创建跨 VPC 克隆的端到端示例

开始前的准备工作

在您开始设置跨 VPC 克隆之前，请确保具有以下资源：

- 用作克隆的源的 Aurora 数据库集群。如果这是您首次创建 Aurora 数据库集群，请参阅开始使用 Amazon Aurora 中的教程，使用 MySQL 或 PostgreSQL 数据库引擎设置集群。
- 第二个 VPC（如果您打算创建跨 VPC 克隆）。如果您没有可用于克隆的 VPC，请参阅教程：创建 VPC 以用于数据库集群（仅限 IPv4）或教程：创建 VPC 以用于数据库集群（双堆栈模式）。

收集有关网络环境的信息

使用跨 VPC 克隆时，原始集群与其克隆集群之间的网络环境可能会有很大不同。在创建克隆之前，请您收集并记录有关原始集群中使用的 VPC、子网、数据库子网组和可用区的信息。这样，您就可以尽可能减少出现问题的可能性。如果出现了网络问题，您无需中断任何故障排除活动即可搜索诊断信息。以下各个部分说明了收集这些类型信息的 CLI 示例。在创建克隆和进行任何故障排除时，您可以将详细信息保存为任意便于查阅的格式。

- 步骤 1：检查原始集群的可用区
- 步骤 2：检查原始集群的数据库子网组
- 步骤 3：检查原始集群的子网
- 步骤 4：检查原始集群中数据库实例的可用区
- 第 5 步：检查可用于克隆集群的 VPC

步骤 1：检查原始集群的可用区

在创建克隆之前，请核实原始集群将哪些可用区用于其存储。如 Amazon Aurora 存储中所述，每个 Aurora 集群的存储都与三个且仅与三个可用区相关联。由于 Amazon Aurora 数据库集群利用了计算和存储分离的优势，因此无论在集群中具有多少个实例，此规则都适用。

例如，运行类似下文的 CLI 命令，将 *my_cluster* 替换为您自己的集群名称。以下示例生成按可用区名称的字母顺序排序的列表。

```
aws rds describe-db-clusters \
  --db-cluster-identifier my_cluster \
  --query 'sort_by(*[].AvailabilityZones[].{Zone:@},&Zone)' \
  --output text
```

以下示例说明了前述 `describe-db-clusters` 命令的示例输出。它表明 Aurora 集群的存储始终使用三个可用区。

```
us-east-1c
us-east-1d
us-east-1e
```

要在没有足够资源连接到这些可用区的网络环境中创建克隆，您必须创建与其中至少两个可用区关联的子网，然后创建包含这两个或三个子网的数据库子网组。下面的示例演示了如何操作。

步骤 2：检查原始集群的数据库子网组

如果您要为克隆集群使用与原始集群相同数量的子网，则可以从原始集群的数据库子网组中获取子网的数量。Aurora 数据库子网组至少包含两个子网，每个子网都与不同的可用区关联。记下子网与哪些可用区关联。

以下示例演示如何查找原始集群的数据库子网组，然后逆向推导出对应的可用区。将第一个命令中的 *my_cluster* 替换为您的集群名称。将第二个命令中的 *my_subnet* 替换为数据库子网组的名称。

```
aws rds describe-db-clusters --db-cluster-identifier my_cluster \
  --query '*[].DBSubnetGroup' --output text

aws rds describe-db-subnet-groups --db-subnet-group-name my_subnet_group \
  --query '*[].Subnets[].[SubnetAvailabilityZone.Name]' --output text
```

对于其数据库子网组包含两个子网的集群，示例输出可能类似于以下内容。在本例中，two-subnets 是创建数据库子网组时指定的名称。

```
two-subnets

us-east-1d
us-east-1c
```

对于数据库子网组包含三个子网的集群，输出可能类似于下文。

```
three-subnets

us-east-1f
us-east-1d
us-east-1c
```

步骤 3：检查原始集群的子网

如果您需要有关原始集群中子网的更多详细信息，请运行类似于下文的 AWS 命令行界面命令。您可以检查子网属性，例如 IP 地址范围、所有者等。这样，您就可以确定是使用同一个 VPC 中的不同子网，还是在不同的 VPC 中创建具有相似特性的子网。

查找您的 VPC 中所有可用子网的子网 ID。

```
aws ec2 describe-subnets --filters Name=vpc-id,Values=my_vpc \
  --query '*[].[SubnetId]' --output text
```

查找在您的数据库子网组中所使用的那些子网。

```
aws rds describe-db-subnet-groups --db-subnet-group-name my_subnet_group \
  --query '*[].Subnets[].[SubnetIdentifier]' --output text
```

然后在列表中指定要调查的子网，如以下命令所示。用您的子网名称替换 *my_subnet_1*，依此类推。

```
aws ec2 describe-subnets \
  --subnet-ids '["my_subnet_1","my_subnet2","my_subnet3"]'
```

以下示例说明了此类 describe-subnets 命令的部分输出。输出显示了您可以查看的每个子网的一些重要属性，例如与其关联的可用区和所属的 VPC。

```
{
    'Subnets': [
        {
            'AvailabilityZone': 'us-east-1d',
            'AvailableIpAddressCount': 54,
            'CidrBlock': '10.0.0.64/26',
            'State': 'available',
            'SubnetId': 'subnet-000a0bca00e0b0000',
            'VpcId': 'vpc-3f3c3fc3333b3ffb3',
            ...
        },
        {
            'AvailabilityZone': 'us-east-1c',
            'AvailableIpAddressCount': 55,
            'CidrBlock': '10.0.0.0/26',
            'State': 'available',
            'SubnetId': 'subnet-4b4dbfe4d4a4fd4c4',
            'VpcId': 'vpc-3f3c3fc3333b3ffb3',
            ...
```

步骤 4：检查原始集群中数据库实例的可用区

您可以使用此过程来了解原始集群中用于数据库实例的可用区。这样，您就可以为克隆集群中的数据库实例设置完全相同的可用区。您还可以根据克隆集群用于生产环境还是开发和测试等，在克隆集群中增加或减少数据库实例的数量。

对于原始集群中的每个实例，运行类似于下文的命令。请确保该实例已完成创建，并且首先处于 Available 状态。将 *my_instance* 替换为实例标识符。

```
aws rds describe-db-instances --db-instance-identifier my_instance \
  --query '*[].AvailabilityZone' --output text
```

下面的示例显示了运行前述 describe-db-instances 命令的输出。Aurora 集群有四个数据库实例。因此，我们运行该命令四次，每次都替换为不同的数据库实例标识符。输出显示了这些数据库实例如何分布在最多三个可用区中。

```
us-east-1a
us-east-1c
us-east-1d
us-east-1a
```

创建克隆集群并向其添加数据库实例后，您可以在 create-db-instance 命令中指定这些相同的可用区名称。您可以采用这种做法来设置数据库实例，以便在新集群中配置与原始集群中完全相同的可用区。

第 5 步：检查可用于克隆集群的 VPC

如果您打算在与原始集群不同的 VPC 中创建克隆集群，则可以获取对您账户可用的 VPC ID 列表。如果您需要在与原始集群相同的 VPC 中创建任何额外的子网，也可以执行此步骤。运行命令创建子网时，您需要指定 VPC ID 作为参数。

要列出对您账户可用的所有 VPC，请运行以下 CLI 命令：

```
aws ec2 describe-vpcs --query '*[].[VpcId]' --output text
```

以下示例说明了前述 describe-vpcs 命令的示例输出。输出显示当前 AWS 账户中有四个 VPC，可以用作跨 VPC 克隆的源或目标。

```
vpc-fd111111
vpc-2222e2cd2a222f22e
vpc-33333333a33333d33
vpc-4ae4d4de4a4444dad
```

您可以将同一个 VPC 作为克隆集群的目标，也可以使用不同的 VPC。如果原始集群和克隆集群位于同一个 VPC 中，则可以为克隆集群重复使用相同的数据库子网组。您也可以创建不同的数据库子网组。例如，新的数据库子网组可能会使用私有子网，而原始集群的数据库子网组可能使用的是公有子网。如果您在不同的 VPC 中创建克隆集群，请确保新 VPC 中有足够的子网，并且这些子网与原始集群中的正确可用区相关联。

为克隆集群创建网络资源

如果在收集网络信息时，您发现克隆集群需要额外的网络资源，则可以先创建这些资源，然后再尝试设置克隆集群。例如，您可能需要创建更多子网、将子网与特定可用区关联或创建新数据库子网组。

- 步骤 1：为克隆集群创建子网
- 步骤 2：为克隆集群创建数据库子网组

步骤 1：为克隆集群创建子网

如果您需要为克隆集群创建新的子网，请运行类似下文的命令。在不同的 VPC 中创建克隆集群或者进行了其他网络更改（例如使用私有子网而非公有子网）时，您可能需要执行此操作。

AWS 会自动生成子网的 ID。将 *my_vpc* 替换为克隆集群的 VPC 的名称。为 `--cidr-block` 选项选择地址范围，允许范围内至少有 16 个 IP 地址。您可以包括要指定的任何其他属性。运行命令 `aws ec2 create-subnet help` 以查看所有选项。

```
aws ec2 create-subnet --vpc-id my_vpc \
  --availability-zone AZ_name --cidr-block IP_range
```

以下示例显示了新创建子网的一些重要属性。

```
{
    'Subnet': {
        'AvailabilityZone': 'us-east-1b',
        'AvailableIpAddressCount': 59,
        'CidrBlock': '10.0.0.64/26',
        'State': 'available',
        'SubnetId': 'subnet-44b4a44f4e44db444',
        'VpcId': 'vpc-555fc5df555e555dc',
        ...
        }
}
```

步骤 2：为克隆集群创建数据库子网组

如果您要在不同的 VPC 中创建克隆集群，或者使用同一 VPC 内的一组不同的子网，则您可以创建一个新的数据库子网组，然后在创建克隆集群时指定该子网组。

请确保您了解以下所有详细信息。您可从前面示例的输出中找到所有这些信息。

1. 原始集群的 VPC。有关说明，请参阅 步骤 3：检查原始集群的子网。
2. 克隆集群的 VPC（当您在不同的 VPC 中创建克隆集群时）。有关说明，请参阅 第 5 步：检查可用于克隆集群的 VPC。
3. 与原始集群的 Aurora 存储相关联的三个可用区。有关说明，请参阅 步骤 1：检查原始集群的可用区。
4. 与原始集群的数据库子网组关联的两个或三个可用区。有关说明，请参阅 步骤 2：检查原始集群的数据库子网组。
5. 您打算用于克隆集群的 VPC 中所有子网的子网 ID 和关联可用区。使用与步骤 3：检查原始集群的子网中相同的 `describe-subnets` 命令，替换目标 VPC 的 VPC ID。

检查有多少个可用区既与原始集群的存储相关联，又与目标 VPC 中的子网相关联。要成功创建克隆集群，必须有两个或三个共同的可用区。如果共同可用区少于两个，请返回到步骤 1：为克隆集群创建子网。创建一个、两个或三个新的子网，这些子网关联到与原始集群存储相关联的可用区。

在目标 VPC 中选择子网，这些子网应关联到与原始集群中的 Aurora 存储关联的相同可用区。理想情况下，选择三个可用区。采用这种做法，克隆集群的数据库实例可以尽可能灵活地分布到多个可用区，从而实现计算资源的高可用性。

运行类似下文的命令来创建新的数据库子网组。替换为您在列表中的子网 ID。如果您使用环境变量指定子网 ID，请注意在括起 `--subnet-ids` 参数列表时，要保留 ID 两边的双引号。

```
aws rds create-db-subnet-group --db-subnet-group-name my_subnet_group \
  --subnet-ids '["my_subnet_1","my_subnet_2","my_subnet3"]' \
  --db-subnet-group-description 'DB subnet group with 3 subnets for clone'
```

下面的示例显示了 `create-db-subnet-group` 命令的部分输出。

```
{
    'DBSubnetGroup': {
        'DBSubnetGroupName': 'my_subnet_group',
        'DBSubnetGroupDescription': 'DB subnet group with 3 subnets for clone',
        'VpcId': 'vpc-555fc5df555e555dc',
        'SubnetGroupStatus': 'Complete',
        'Subnets': [
            {
                'SubnetIdentifier': 'my_subnet_1',
                'SubnetAvailabilityZone': {
                    'Name': 'us-east-1c'
                },
```

```
                'SubnetStatus': 'Active'
            },
            {
                'SubnetIdentifier': 'my_subnet_2',
                'SubnetAvailabilityZone': {
                    'Name': 'us-east-1d'
                },
                'SubnetStatus': 'Active'
            }
            ...
        ],
        'SupportedNetworkTypes': [
            'IPV4'
        ]
    }
}
```

此时，您还没有实际创建克隆。您已创建所有相关的 VPC 和子网资源，这样在创建克隆集群时，您可以为 `restore-db-cluster-to-point-in-time` 和 `create-db-instance` 命令指定相应的参数。

使用新的网络设置创建 Aurora 克隆

在您确保 VPC、子网、可用区和子网组的配置正确且可供新集群使用时，就可以执行实际的克隆操作了。以下 CLI 示例重点介绍在设置克隆及其数据库实例时，您在命令中指定的选项，例如 `--db-subnet-group-name`、`--availability-zone` 和 `--vpc-security-group-ids`。

- 步骤 1：为克隆集群指定数据库子网组
- 步骤 2：为克隆集群中的实例指定网络设置
- 步骤 3：建立从客户端系统到克隆集群的连接

步骤 1：为克隆集群指定数据库子网组

创建克隆时，您可以通过指定数据库子网组来正确配置所有 VPC、子网和可用区设置。使用前面示例中的命令来验证进入数据库子网组的所有关系和映射。

例如，以下命令用于将原始集群克隆到克隆集群。在第一个示例中，源集群与两个子网关联，克隆集群与三个子网关联。第二个示例显示了相反的情况，即从具有三个子网的集群克隆到具有两个子网的集群。

```
aws rds restore-db-cluster-to-point-in-time \
```

```
  --source-db-cluster-identifier cluster-with-3-subnets \
  --db-cluster-identifier cluster-cloned-to-2-subnets \
  --restore-type copy-on-write --use-latest-restorable-time \
  --db-subnet-group-name two-subnets
```

如果您打算在克隆集群中使用 Aurora Serverless v2 实例，请在创建克隆时提供 `--serverless-v2-scaling-configuration` 选项，如下所示。这样做可以让您在克隆集群中创建数据库实例时使用 `db.serverless` 类。您也可以在以后修改克隆集群来添加此扩展配置属性。此示例中的容量数字允许集群中的每个 Serverless v2 实例在 2 到 32 个 Aurora 容量单位（ACU，Aurora Capacity Unit）之间扩展。有关 Aurora Serverless v2 功能以及如何选择容量范围的信息，请参阅使用 Aurora Serverless v2。

```
aws rds restore-db-cluster-to-point-in-time \
  --source-db-cluster-identifier cluster-with-2-subnets \
  --db-cluster-identifier cluster-cloned-to-3-subnets \
  --restore-type copy-on-write --use-latest-restorable-time \
  --db-subnet-group-name three-subnets \
  --serverless-v2-scaling-configuration 'MinCapacity=2,MaxCapacity=32'
```

无论数据库实例使用多少个子网，源集群和克隆集群的 Aurora 存储都与三个可用区相关联。以下示例针对前面示例中的两个 `restore-db-cluster-to-point-in-time` 命令，列出了与源集群和克隆集群关联的可用区。

```
aws rds describe-db-clusters --db-cluster-identifier cluster-with-3-subnets \
  --query 'sort_by(*[].AvailabilityZones[].{Zone:@},&Zone)' --output text

us-east-1c
us-east-1d
us-east-1f

aws rds describe-db-clusters --db-cluster-identifier cluster-cloned-to-2-subnets \
  --query 'sort_by(*[].AvailabilityZones[].{Zone:@},&Zone)' --output text

us-east-1c
us-east-1d
us-east-1f

aws rds describe-db-clusters --db-cluster-identifier cluster-with-2-subnets \
  --query 'sort_by(*[].AvailabilityZones[].{Zone:@},&Zone)' --output text

us-east-1a
```

```
us-east-1c
us-east-1d

aws rds describe-db-clusters --db-cluster-identifier cluster-cloned-to-3-subnets \
  --query 'sort_by(*[].AvailabilityZones[].{Zone:@},&Zone)' --output text

us-east-1a
us-east-1c
us-east-1d
```

由于每一对原始集群和克隆集群之间至少有两个可用区重叠，因此两个集群都可以访问相同的底层 Aurora 存储。

步骤 2：为克隆集群中的实例指定网络设置

在克隆集群中创建数据库实例时，默认情况下，它们会继承集群本身的数据库子网组。这样，Aurora 就会自动将每个实例分配到一个特定的子网，并在与该子网关联的可用区中创建实例。这种选择非常方便，尤其是对于开发和测试系统，因为您在向克隆中添加新实例时，不必跟踪子网 ID 或可用区。

或者，您可以在为克隆集群创建 Aurora 数据库实例时指定可用区。您指定的可用区必须来自与克隆集群关联的一组可用区。如果您用于克隆集群的数据库子网组仅包含两个子网，则您只能从与这两个子网关联的可用区中进行选择。这种选择为高可用性系统提供了灵活性和韧性，因为您可以确保写入器实例和备用读取器实例位于不同的可用区中。或者，如果您向集群添加其他读取器，则可以确保它们分布在三个可用区中。这样，即使在极少的情况下可用区出现了故障，您在另外两个可用区中仍然有一个写入器实例和另一个读取器实例。

以下示例将预置的数据库实例添加到使用自定义数据库子网组的克隆 Aurora PostgreSQL 集群。

```
aws rds create-db-instance --db-cluster-identifier my_aurora_postgresql_clone \
  --db-instance-identifier my_postgres_instance \
  --db-subnet-group-name my_new_subnet \
  --engine aurora-postgresql \
  --db-instance-class db.t4g.medium
```

以下示例说明了此命令的部分输出。

```
{
  'DBInstanceIdentifier': 'my_postgres_instance',
  'DBClusterIdentifier': 'my_aurora_postgresql_clone',
  'DBInstanceClass': 'db.t4g.medium',
  'DBInstanceStatus': 'creating'
```

```
  ...
}
```

以下示例将 Aurora Serverless v2 数据库实例添加到使用自定义数据库子网组的 Aurora MySQL 克隆集群中。为了能够使用 Serverless v2 实例，请确保为 `restore-db-cluster-to-point-in-time` 命令指定 `--serverless-v2-scaling-configuration` 选项，如前面的示例所示。

```
aws rds create-db-instance --db-cluster-identifier my_aurora_mysql_clone \
  --db-instance-identifier my_mysql_instance \
  --db-subnet-group-name my_other_new_subnet \
  --engine aurora-mysql \
  --db-instance-class db.serverless
```

以下示例说明了此命令的部分输出。

```
{
  'DBInstanceIdentifier': 'my_mysql_instance',
  'DBClusterIdentifier': 'my_aurora_mysql_clone',
  'DBInstanceClass': 'db.serverless',
  'DBInstanceStatus': 'creating'
  ...
}
```

步骤 3：建立从客户端系统到克隆集群的连接

如果您已经从客户端系统连接到 Aurora 集群，则可能需要允许对新克隆集群使用相同类型的连接。例如，您可能从 Amazon Cloud9 实例或 EC2 实例连接到原始集群。要允许来自相同客户端系统或您在目标 VPC 中创建的新客户端系统的连接，请设置与 VPC 中等同的数据库子网组和 VPC 安全组。然后，在创建克隆集群时指定子网组和安全组。

以下示例设置 Aurora Serverless v2 克隆集群。该配置基于创建数据库集群时 `--engine-mode provisioned` 和 `--serverless-v2-scaling-configuration` 的组合，然后在集群中创建每个数据库实例使用 `--db-instance-class db.serverless`。`provisioned` 引擎模式是默认模式，因此如果您首选该模式，可以省略该选项。

```
aws rds restore-db-cluster-to-point-in-time \
  --source-db-cluster-identifier serverless-sql-postgres\
  --db-cluster-identifier serverless-sql-postgres-clone \
  --db-subnet-group-name 'default-vpc-1234' \
  --vpc-security-group-ids 'sg-4567' \
```

```
  --serverless-v2-scaling-configuration 'MinCapacity=0.5,MaxCapacity=16' \
  --restore-type copy-on-write \
  --use-latest-restorable-time
```

然后，在克隆集群中创建数据库实例时，请指定相同的 `--db-subnet-group-name` 选项。或者，您可以提供 `--availability-zone` 选项并指定与该子网组中子网关联的可用区之一。此可用区还必须是与原始集群关联的可用区之一。

```
aws rds create-db-instance \
  --db-cluster-identifier serverless-sql-postgres-clone \
  --db-instance-identifier serverless-sql-postgres-clone-instance \
  --db-instance-class db.serverless \
  --db-subnet-group-name 'default-vpc-987zyx654' \
  --availability-zone 'us-east-1c' \
  --engine aurora-postgresql
```

将集群从公有子网移动到私有子网

您可以使用克隆在公有子网和私有子网之间移动集群。在将应用程序部署到生产环境之前，要为其添加额外的安全层时，您可以执行此操作。在本示例中，在开始 Aurora 的克隆过程之前，您应该已经设置好私有子网和 NAT 网关。

对于涉及 Aurora 的步骤，您可以按照前面示例中的相同常规步骤操作，以便收集有关网络环境的信息和使用新的网络设置创建 Aurora 克隆。主要区别在于，即使您有映射到原始集群中所有可用区的公有子网，现在您也必须验证 Aurora 集群是否有足够的私有子网，以及这些子网是否与原始集群中用于 Aurora 存储的相同的所有可用区关联。与其他克隆应用场景类似，您可以为克隆生成数据库子网组，在子网组中包括与所需可用区关联的两个或三个私有子网。但是，如果您在数据库子网组中使用两个私有子网，则必须有第三个私有子网，该子网与原始集群中用于 Aurora 存储的第三个可用区相关联。

您可以查阅此清单，验证是否满足执行此类克隆操作的所有要求。

- 记录与原始集群关联的三个可用区。有关说明，请参阅 步骤 1：检查原始集群的可用区。
- 记录与原始集群的数据库子网组中公有子网关联的两个或三个可用区。有关说明，请参阅 步骤 3：检查原始集群的子网。
- 创建映射到与原始集群关联的全部三个可用区的私有子网。此外还需要完成与私有子网通信所需的任何其他联网设置，例如创建 NAT 网关。有关说明，请参阅《Amazon Virtual Private Cloud 用户指南》中的创建子网。
- 创建一个新数据库子网组，其中包含三个或两个私有子网，这些子网一开始就与可用区关联。有关说明，请参阅 步骤 2：为克隆集群创建数据库子网组。

满足所有先决条件后，您可以在创建克隆集群时暂停原始集群上的数据库活动，然后切换应用程序来使用克隆集群。在创建了克隆集群并确认可以连接到克隆集群、运行应用程序代码等之后，您可以停止使用原始集群。

创建跨 VPC 克隆的端到端示例

在与原始集群不同的 VPC 中创建克隆集群时，所使用的常规步骤与前面的示例相同。因为 VPC ID 是子网的一个属性，所以在运行任何 RDS CLI 命令时，您无需指定 VPC ID 作为参数。主要区别在于，您更可能需要创建新的子网，将新子网映射到特定可用区，创建 VPC 安全组和新的数据库子网组。如果这是您在该 VPC 中创建的第一个 Aurora 集群，则尤其如此。

您可以查阅此清单，验证是否满足执行此类克隆操作的所有要求。

- 记录与原始集群关联的三个可用区。有关说明，请参阅 步骤 1：检查原始集群的可用区。
- 记录与原始集群的数据库子网组中子网关联的两个或三个可用区。有关说明，请参阅 步骤 2：检查原始集群的数据库子网组。
- 创建映射到与原始集群关联的全部三个可用区的子网。有关说明，请参阅 步骤 1：为克隆集群创建子网。
- 针对客户端系统、应用程序服务器等，执行与克隆集群中的数据库实例进行通信所需的任何其他联网设置，例如设置 VPC 安全组。有关说明，请参阅 使用安全组控制访问权限。
- 创建一个新数据库子网组，其中包含三个或两个子网，这些子网一开始就与可用区关联。有关说明，请参阅 步骤 2：为克隆集群创建数据库子网组。

满足所有先决条件后，您可以在创建克隆集群时暂停原始集群上的数据库活动，然后切换应用程序来使用克隆集群。在创建了克隆集群并确认可以连接到克隆集群、运行应用程序代码等之后，您可以考虑是保持原始集群和克隆集群继续运行，还是停止使用原始集群。

以下 Linux 示例说明了将 Aurora 数据库集群从一个 VPC 克隆到另一个 VPC 的 AWS 命令行界面操作顺序。命令输出中没有显示与示例无关的一些字段。

首先，我们检查源和目标 VPC 的 ID。在 VPC 元数据中，您在创建 VPC 时为其分配的描述性名称表示为标签。

```
$ aws ec2 describe-vpcs --query '*[].[VpcId,Tags]'
[
    [
        'vpc-0f0c0fc0000b0ffb0',
        [
```

```
            {
                'Key': 'Name',
                'Value': 'clone-vpc-source'
            }
        ]
    ],
    [
        'vpc-9e99d9f99a999bd99',
        [
            {
                'Key': 'Name',
                'Value': 'clone-vpc-dest'
            }
        ]
    ]
]
```

源 VPC 中已存在原始集群。要使用 Aurora 存储的同一组可用区来设置克隆集群，我们需要检查原始集群使用的可用区。

```
$ aws rds describe-db-clusters --db-cluster-identifier original-cluster \
  --query 'sort_by(*[].AvailabilityZones[].{Zone:@},&Zone)' --output text

us-east-1c
us-east-1d
us-east-1f
```

我们需要确保存在与原始集群使用的可用区相对应的子网：us-east-1c、us-east-1d 和 us-east-1f。

```
$ aws ec2 create-subnet --vpc-id vpc-9e99d9f99a999bd99 \
  --availability-zone us-east-1c --cidr-block 10.0.0.128/28
{
    'Subnet': {
        'AvailabilityZone': 'us-east-1c',
        'SubnetId': 'subnet-3333a33be3ef3e333',
        'VpcId': 'vpc-9e99d9f99a999bd99',
    }
}

$ aws ec2 create-subnet --vpc-id vpc-9e99d9f99a999bd99 \
--availability-zone us-east-1d --cidr-block 10.0.0.160/28
```

```
{
    'Subnet': {
        'AvailabilityZone': 'us-east-1d',
        'SubnetId': 'subnet-4eeb444cd44b4d444',
        'VpcId': 'vpc-9e99d9f99a999bd99',
    }
}

$ aws ec2 create-subnet --vpc-id vpc-9e99d9f99a999bd99 \
--availability-zone us-east-1f --cidr-block 10.0.0.224/28
{
    'Subnet': {
        'AvailabilityZone': 'us-east-1f',
        'SubnetId': 'subnet-66eea6666fb66d66c',
        'VpcId': 'vpc-9e99d9f99a999bd99',
    }
}
```

此示例确认了目标 VPC 中存在映射到所需可用区的子网。

```
aws ec2 describe-subnets --query 'sort_by(*[] | [?VpcId == `vpc-9e99d9f99a999bd99`] |
[].{SubnetId:SubnetId,VpcId:VpcId,AvailabilityZone:AvailabilityZone},
 &AvailabilityZone)' --output table

-------------------------------------------------------------------------
|                            DescribeSubnets                            |
+------------------+----------------------------+-------------------------+
| AvailabilityZone |         SubnetId           |          VpcId          |
+------------------+----------------------------+-------------------------+
|  us-east-1a      |  subnet-000ff0e00000c0aea  |  vpc-9e99d9f99a999bd99  |
|  us-east-1b      |  subnet-1111d111111ca11b1  |  vpc-9e99d9f99a999bd99  |
|  us-east-1c      | subnet-3333a33be3ef3e333   |  vpc-9e99d9f99a999bd99  |
|  us-east-1d      | subnet-4eeb444cd44b4d444   |  vpc-9e99d9f99a999bd99  |
|  us-east-1f      | subnet-66eea6666fb66d66c   |  vpc-9e99d9f99a999bd99  |
+------------------+----------------------------+-------------------------+
```

在 VPC 中创建 Aurora 数据库集群之前，您必须有一个数据库子网组，其子网映射到用于 Aurora 存储的可用区。创建常规集群时，您可以使用任意三个可用区的组合。克隆现有集群时，子网组必须与用于 Aurora 存储的三个可用区中的至少两个相匹配。

```
$ aws rds create-db-subnet-group \
  --db-subnet-group-name subnet-group-in-other-vpc \
```

```
  --subnet-ids
 '["subnet-3333a33be3ef3e333","subnet-4eeb444cd44b4d444","subnet-66eea6666fb66d66c"]' \
  --db-subnet-group-description 'DB subnet group with 3 subnets:
 subnet-3333a33be3ef3e333,subnet-4eeb444cd44b4d444,subnet-66eea6666fb66d66c'

{
    'DBSubnetGroup': {
        'DBSubnetGroupName': 'subnet-group-in-other-vpc',
        'DBSubnetGroupDescription': 'DB subnet group with 3 subnets:
 subnet-3333a33be3ef3e333,subnet-4eeb444cd44b4d444,subnet-66eea6666fb66d66c',
        'VpcId': 'vpc-9e99d9f99a999bd99',
        'SubnetGroupStatus': 'Complete',
        'Subnets': [
            {
                'SubnetIdentifier': 'subnet-4eeb444cd44b4d444',
                'SubnetAvailabilityZone': { 'Name': 'us-east-1d' }
            },
            {
                'SubnetIdentifier': 'subnet-3333a33be3ef3e333',
                'SubnetAvailabilityZone': { 'Name': 'us-east-1c' }
            },
            {
                'SubnetIdentifier': 'subnet-66eea6666fb66d66c',
                'SubnetAvailabilityZone': { 'Name': 'us-east-1f' }
            }
        ]
    }
}
```

现在，子网和数据库子网组已准备就绪。以下示例说明了用于克隆集群的 restore-db-cluster-to-point-in-time。--db-subnet-group-name 选项将克隆集群与正确的子网组相关联，这些子网映射到原始集群中正确的一组可用区。

```
$ aws rds restore-db-cluster-to-point-in-time \
  --source-db-cluster-identifier original-cluster \
  --db-cluster-identifier clone-in-other-vpc \
  --restore-type copy-on-write --use-latest-restorable-time \
  --db-subnet-group-name subnet-group-in-other-vpc

{
  'DBClusterIdentifier': 'clone-in-other-vpc',
  'DBSubnetGroup': 'subnet-group-in-other-vpc',
  'Engine': 'aurora-postgresql',
```

```
  'EngineVersion': '15.4',
  'Status': 'creating',
  'Endpoint': 'clone-in-other-vpc.cluster-c0abcdef.us-east-1.rds.amazonaws.com'
}
```

以下示例确认克隆集群中的 Aurora 存储使用了与原始集群相同的一组可用区。

```
$ aws rds describe-db-clusters --db-cluster-identifier clone-in-other-vpc \
  --query 'sort_by(*[].AvailabilityZones[].{Zone:@},&Zone)' --output text

us-east-1c
us-east-1d
us-east-1f
```

此时，您可以为克隆集群创建数据库实例。在与每个实例关联的 VPC 安全组中，确保允许来自目标 VPC 中的 EC2 实例、应用程序服务器等所用的 IP 地址范围的连接。

使用 AWS RAM 与 Amazon Aurora 进行跨账户克隆

通过将 AWS Resource Access Manager（AWS RAM）与 Amazon Aurora 结合使用，您可以与另一个AWS账户或企业共享属于您的AWS账户的 Aurora 数据库集群和克隆。这种跨账户克隆比创建和恢复数据库快照快很多。您可以为您的 Aurora 数据库集群创建一个克隆并共享该克隆。或者，您可以与另一个AWS账户共享您的 Aurora 数据库集群，并让账户持有人创建克隆。方式的选择因使用案例而定。

如，您可能需要定期与组织的内部审计团队共享财务数据库的克隆。在这种情况下，您的审计团队对其使用的应用程序拥有自己的 AWS 账户。您可以授予审计团队的AWS账户访问 Aurora 数据库集群，并根据需要进行克隆的权限。

另一方面，如果外部供应商审核您的财务数据，您可能更愿意自己创建克隆。然后，您只允许外部供应商访问该克隆。

您还可以使用跨账户克隆来支持在同一AWS账户内进行克隆的许多相同用例，例如开发和测试。例如，您的企业可能会使用不同的 AWS 账户进行生产、开发、测试等。有关更多信息，请参阅“Aurora 克隆概述”。

因此，您可能希望与另一个AWS账户共享一个克隆或允许其他AWS账户创建您的 Aurora 数据库集群的克隆。无论哪种情况，请先使用 AWS RAM 来创建共享对象。有关在 AWS 账户之间共享 AWS 资源的完整信息，请参阅 AWS RAM 用户指南。

创建跨账户克隆需要来自拥有原始集群的 AWS 账户和创建克隆的 AWS 账户的操作。首先，原始集群拥有者修改集群以允许一个或多个其他账户克隆它。对于在不同的AWS组织中的任何账户，AWS会生成共享邀请。另一个账户必须接受邀请，然后才能继续。之后，每个授权账户便能克隆集群。在整个过程中，集群由其唯一的 Amazon Resource Name (ARN) 标识。

与在同一 AWS 账户内进行克隆一样，只有在源或克隆对数据进行更改时才使用额外的存储空间。届时将会收取存储费用。如果删除源集群，则会在剩余的克隆集群中平均分配存储成本。

主题

- 跨账户克隆的限制
- 允许其他 AWS 账户克隆您的集群
- 克隆另一个AWS账户拥有的集群

跨账户克隆的限制

Aurora 跨账户克隆具有以下限制：

- 您无法跨 Aurora Serverless v1 账户克隆 AWS 集群。
- 您无法使用 AWS Management Console 来查看或接受共享资源的邀请。使用 AWS CLI、Amazon RDS API 或 AWS RAM 控制台查看和接受共享资源的邀请。
- 您仅可以从与您的 AWS 账户共享的克隆创建一个新克隆。
- 您无法共享已与您的AWS账户共享的资源（克隆或 Aurora 数据库集群）。
- 您最多可从任何单个 Aurora 数据库集群创建 15 个跨账户克隆。
- 这 15 个跨账户克隆中的每一个都必须由不同的 AWS 账户拥有。也就是说，任何AWS账户都只能创建一个集群的跨账户克隆。
- 克隆集群之后，为了对跨账户克隆实施限制，原始集群及其克隆应视为相同。不能在同一 AWS 账户内同时创建原始集群和克隆集群的跨账户克隆。原始集群及其任何克隆的跨账户克隆总数不能超过 15 个。
- 除非集群处于 `ACTIVE` 状态，否则您无法与其他AWS账户共享 Aurora 数据库集群。
- 您无法重命名与其他AWS账户共享的 Aurora 数据库集群。
- 您无法创建使用默认 RDS 密钥加密的集群的跨账户克隆。
- 对于从另一个 AWS 账户共享的加密 Aurora 数据库集群，您无法在 AWS 账户中创建非加密克隆。集群拥有者必须授予访问源集群的 AWS KMS key 的权限。但是，您可以在创建克隆时使用不同的密钥。

允许其他 AWS 账户克隆您的集群

要允许其他 AWS 账户克隆您拥有的集群，请使用 AWS RAM 设置共享权限。这样做还会向位于其他 AWS 组织中的所有其他账户发送邀请。

有关在 AWS RAM 控制台中共享您拥有的资源的过程，请参阅 AWS RAM 用户指南中的共享您拥有的资源。

主题

- 向其他 AWS 账户授予克隆集群的权限
- 检查是否与其他AWS账户共享您拥有的集群

向其他 AWS 账户授予克隆集群的权限

如果您共享的集群已加密，则您还可以共享集群的 AWS KMS key。您可以允许一个 AWS 账户中的 AWS Identity and Access Management (IAM) 用户或角色使用其他账户中的 KMS 密钥。

为此，您首先通过 AWS KMS 将外部账户（根用户）添加到 KMS 密钥的密钥策略中。您无需将单个用户或角色添加到密钥策略，只需添加拥有这些用户或角色的外部账户。您只能共享您创建的 KMS 密钥，而不能共享默认 RDS 服务密钥。有关 KMS 密钥的访问控制的信息，请参阅 AWS KMS 的身份验证和访问控制。

控制台

授予克隆您的集群的权限

1. 登录AWS Management Console并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Databases (数据库)。
3. 选择要共享的数据库集群以查看其 Details (详细信息) 页面，并选择 Connectivity & security (连接和安全性) 选项卡。
4. 在 与其他 AWS 账户共享数据库集群 部分中，输入要允许克隆此集群的 AWS 账户的数字账户 ID。对于同一组织中的账户 ID，您可以开始在框中键入，然后从菜单中选择。

> **Important**
>
> 在某些情况下，您可能需要使用一个不在您的账户所在的 AWS 组织中的账户来克隆集群。在这些情况下，出于安全原因，控制台不会报告谁拥有账户 ID 或该账户是否存在。

请小心输入不在您的 AWS 账户所在的 AWS 组织中的账号。立即验证您是否与目标账户共享。

5. 在确认页面上，验证您指定的账户 ID 是否正确。在确认框中输入 `share` 以进行确认。

   在 Details (详细信息) 页面上，Accounts that this DB cluster is shared with (此数据库集群与之共享的账户) 下显示一个条目，该条目显示指定的AWS账户 ID。Status (状态) 列最初显示 Pending (待处理) 状态。

6. 联系另一个 AWS 账户的拥有者，或者如果您同时拥有这两个账户，则登录到该账户。指示其他账户的拥有者接受共享邀请并克隆数据库集群，如下所述。

AWS CLI

授予克隆您的集群的权限

1. 收集所需参数的信息。您需要集群的 ARN 和其他 AWS 账户的数字 ID。
2. 运行 AWS RAM CLI 命令 `create-resource-share`。

   对于 Linux、macOS 或 Unix：

```
aws ram create-resource-share --name descriptive_name \
  --region region \
  --resource-arns cluster_arn \
  --principals other_account_ids
```

   对于 Windows：

```
aws ram create-resource-share --name descriptive_name ^
  --region region ^
  --resource-arns cluster_arn ^
  --principals other_account_ids
```

   要包含 `--principals` 参数的多个账户 ID，请用空格将各个 ID 分隔开。要指定允许的账户 ID 是否能位于 AWS 组织外部，请包含 `--allow-external-principals` 的 `--no-allow-external-principals` 或 `create-resource-share` 参数。

AWS RAM API

授予克隆您的集群的权限

1. 收集所需参数的信息。您需要集群的 ARN 和其他 AWS 账户的数字 ID。
2. 调用 AWS RAM API 操作 CreateResourceShare，并指定以下值：
   - 指定一个或多个AWS账户的账户 ID 作为 `principals` 参数。
   - 指定一个或多个 Aurora 数据库集群的 ARN 作为 `resourceArns` 参数。
   - 通过包含 `allowExternalPrincipals` 参数的布尔值来指定允许的账户 ID 是否能位于您的 AWS 企业外部。

重新创建使用默认 RDS 密钥的集群

如果您计划共享的加密集群使用默认 RDS 密钥，请确保重新创建集群。为此，请为数据库集群创建手动快照，使用 AWS KMS key，然后将集群还原到新集群。然后共享新集群。要执行此过程，请按以下步骤进行操作。

重新创建使用默认 RDS 密钥的加密集群

1. 登录AWS Management Console并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Snapshots (快照)。
3. 选择您的快照。
4. 对于 Actions (操作)，选择 Copy Snapshot (复制快照)，然后选择 Enable encryption (启用加密)。
5. 对于 AWS KMS key，选择要使用的新加密密钥。
6. 还原复制的快照。为此，请按照 中的过程操作从数据库集群快照还原 新数据库实例使用新的加密密钥。
7. （可选）如果您不再需要旧的数据库集群，请将其删除。为此，请按照 中的过程操作删除数据库集群快照 在这样做之前，请确认您的新集群拥有所有必要的数据，并且您的应用程序能够成功地访问该数据。

检查是否与其他AWS账户共享您拥有的集群

您可以检查其他用户是否有权共享集群。这样做可以帮助您了解集群是否正在接近跨账户克隆的最大数量限制。

有关使用 AWS RAM 控制台共享资源的过程，请参阅 AWS RAM 用户指南中的共享您拥有的资源。

AWS CLI

要查明您拥有的集群是否与其他AWS账户共享

- 调用 AWS RAM CLI 命令 `list-principals`，将账户 ID 用作资源拥有者，并将集群 ARN 用作资源 ARN。可使用以下命令来查看所有共享。结果指示允许哪些 AWS 账户克隆集群。

```
aws ram list-principals \
    --resource-arns your_cluster_arn \
    --principals your_aws_id
```

AWS RAM API

要查明您拥有的集群是否与其他AWS账户共享

- 调用 AWS RAM API 操作 ListPrincipals。将账户 ID 用作资源拥有者，并将集群 ARN 用作资源 ARN。

克隆另一个AWS账户拥有的集群

要克隆另一个 AWS 账户拥有的集群，请使用 AWS RAM 获取创建克隆的权限。在获得必需权限后，您可以使用克隆 Aurora 集群的标准过程。

您还可以检查您拥有的集群是否为其他 AWS 账户拥有的集群的克隆。

有关在 AWS RAM 控制台中使用其他账户拥有的资源的过程，请参阅 AWS RAM 用户指南中的访问与您共享的资源。

主题

- 查看克隆其他AWS账户拥有的集群的邀请
- 接受共享其他 AWS 账户拥有的集群的邀请
- 克隆另一个AWS账户拥有的 Aurora 集群
- 检查数据库集群是否为跨账户克隆

查看克隆其他AWS账户拥有的集群的邀请

要处理克隆其他 AWS 组织中的 AWS 账户拥有的集群的邀请，请使用 AWS CLI、AWS RAM 控制台或 AWS RAM API。目前，您无法使用 Amazon RDS 控制台执行此过程。

有关在 AWS RAM 控制台中处理邀请的过程，请参阅 AWS RAM 用户指南中的访问与您共享的资源。

AWS CLI

查看克隆其他AWS账户拥有的集群的邀请

1. 运行 AWS RAM CLI 命令 get-resource-share-invitations。

   ```
   aws ram get-resource-share-invitations --region region_name
   ```

   上述命令的结果显示所有克隆集群的邀请，包括您已接受或拒绝的任何邀请。
2. （可选）筛选列表，以便仅查看需要您执行操作的邀请。为此，请添加参数 `--query 'resourceShareInvitations[?status==`PENDING`]'`。

AWS RAM API

查看克隆其他AWS账户拥有的集群的邀请

1. 调用 AWS RAM API 操作 GetResourceShareInvitations。此操作返回所有此类邀请，包括您已接受或拒绝的任何邀请。
2. （可选）通过在 `resourceShareAssociations` 返回字段中检查 `status` 值 `PENDING`，仅查找需要您执行操作的邀请。

接受共享其他 AWS 账户拥有的集群的邀请

您可以接受共享其他 AWS 组织中的其他 AWS 账户拥有的集群的邀请。要处理这些邀请，请使用 AWS CLI、AWS RAM 和 RDS API 或 AWS RAM 控制台。目前，您无法使用 RDS 控制台执行此过程。

有关在 AWS RAM 控制台中处理邀请的过程，请参阅 AWS RAM 用户指南中的访问与您共享的资源。

AWS CLI

接受共享其他AWS账户的集群的邀请

1. 通过运行 AWS RAM CLI 命令 get-resource-share-invitations 查找邀请 ARN，如上所示。
2. 通过调用 AWS RAM CLI 命令 accept-resource-share-invitation 接受邀请，如下所示。

   对于 Linux、macOS 或 Unix：

   ```
   aws ram accept-resource-share-invitation \
     --resource-share-invitation-arn invitation_arn \
     --region region
   ```

   对于 Windows：

   ```
   aws ram accept-resource-share-invitation ^
     --resource-share-invitation-arn invitation_arn ^
     --region region
   ```

AWS RAM 和 RDS API

接受共享他人集群的邀请

1. 通过调用 AWS RAM API 操作 GetResourceShareInvitations 查找邀请 ARN，如上所示。
2. 将 ARN 作为 resourceShareInvitationArn 参数传递到 RDS API 操作 AcceptResourceShareInvitation。

克隆另一个AWS账户拥有的 Aurora 集群

接受来自拥有数据库集群的 AWS 账户的邀请后（如上所示），您可以克隆集群。

控制台

克隆另一个AWS账户拥有的 Aurora 集群

1. 登录AWS Management Console并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Databases (数据库)。

在数据库列表的顶部，您应看到一个或多个 Role (角色) 值为 `Shared from account #account_id` 的项。出于安全原因，您只能看到有关原始集群的有限信息。您可以看到的属性是数据库引擎和版本等属性，这些属性在克隆的集群中必须相同。

3. 选择您打算克隆的集群。
4. 对于 Actions (操作)，选择 Create clone (创建克隆)。
5. 按照控制台中的过程进行操作以完成对克隆集群的设置。
6. 根据需要，为克隆的集群启用加密。如果要克隆的集群已加密，则必须为克隆的集群启用加密。与您共享集群的 AWS 账户还必须共享用于加密集群的 KMS 密钥。您可以使用相同的 KMS 密钥来加密克隆，也可以使用您自己的 KMS 密钥。无法为使用原定设置 KMS 密钥加密的集群创建跨账户克隆。

   拥有加密密钥的账户必须通过使用密钥策略来向目标账户授予密钥的使用权限。此过程类似于共享加密快照的方式，使用密钥策略来向目标账户授予密钥的使用权限。

AWS CLI

克隆另一个AWS账户拥有的 Aurora 集群

1. 接受来自拥有数据库集群的 AWS 账户的邀请，如上所示。
2. 通过在 RDS CLI 命令 `source-db-cluster-identifier` 的 `restore-db-cluster-to-point-in-time` 参数中指定源集群的完整 ARN 来克隆集群，如下所示。

   如果未共享作为 `source-db-cluster-identifier` 传递的 ARN，则返回相同的错误，就像指定的集群不存在一样。

   对于 Linux、macOS 或 Unix：

```
aws rds restore-db-cluster-to-point-in-time \
  --source-db-cluster-identifier=arn:aws:rds:arn_details \
  --db-cluster-identifier=new_cluster_id \
  --restore-type=copy-on-write \
  --use-latest-restorable-time
```

   对于 Windows：

```
aws rds restore-db-cluster-to-point-in-time ^
  --source-db-cluster-identifier=arn:aws:rds:arn_details ^
```

```
  --db-cluster-identifier=new_cluster_id ^
  --restore-type=copy-on-write ^
  --use-latest-restorable-time
```

3. 如果要克隆的集群已加密，请通过包含 kms-key-id 参数来加密克隆的集群。此 kms-key-id 值可以是用于加密原始数据库集群的值，也可以是您自己的 KMS 密钥。您的账户必须具有使用该加密密钥的权限。

   对于 Linux、macOS 或 Unix：

```
aws rds restore-db-cluster-to-point-in-time \
  --source-db-cluster-identifier=arn:aws:rds:arn_details \
  --db-cluster-identifier=new_cluster_id \
  --restore-type=copy-on-write \
  --use-latest-restorable-time \
  --kms-key-id=arn:aws:kms:arn_details
```

   对于 Windows：

```
aws rds restore-db-cluster-to-point-in-time ^
  --source-db-cluster-identifier=arn:aws:rds:arn_details ^
  --db-cluster-identifier=new_cluster_id ^
  --restore-type=copy-on-write ^
  --use-latest-restorable-time ^
  --kms-key-id=arn:aws:kms:arn_details
```

   拥有加密密钥的账户必须通过使用密钥策略来向目标账户授予密钥的使用权限。此过程类似于共享加密快照的方式，使用密钥策略来向目标账户授予密钥的使用权限。下面是密钥策略的示例。

```
{
  "Id": "key-policy-1",
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Allow use of the key",
      "Effect": "Allow",
      "Principal": {"AWS": [
        "arn:aws:iam::account_id:user/KeyUser",
        "arn:aws:iam::account_id:root"
      ]},
      "Action": [
        "kms:CreateGrant",
```

```
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:DescribeKey"
      ],
      "Resource": "*"
    },
    {
      "Sid": "Allow attachment of persistent resources",
      "Effect": "Allow",
      "Principal": {"AWS": [
        "arn:aws:iam::account_id:user/KeyUser",
        "arn:aws:iam::account_id:root"
      ]},
      "Action": [
        "kms:CreateGrant",
        "kms:ListGrants",
        "kms:RevokeGrant"
      ],
      "Resource": "*",
      "Condition": {"Bool": {"kms:GrantIsForAWSResource": true}}
    }
  ]
}
```

> **Note**
>
> restore-db-cluster-to-point-in-time AWS CLI 命令仅还原数据库集群，而不还原该数据库集群的数据库实例。要为恢复的数据库集群创建数据库实例，请调用 create-db-instance 命令。在 `--db-cluster-identifier` 中指定恢复的数据库集群的标识符。
> 只有在完成 `restore-db-cluster-to-point-in-time` 命令并且数据库集群可用后，您才能创建数据库实例。

RDS API

克隆另一个AWS账户拥有的 Aurora 集群

1. 接受来自拥有数据库集群的 AWS 账户的邀请，如上所示。

2. 通过在 RDS API 操作 SourceDBClusterIdentifier 的 RestoreDBClusterToPointInTime 参数中指定源集群的完整 ARN 来克隆集群。

   如果未共享作为 SourceDBClusterIdentifier 传递的 ARN，则返回相同的错误，就像指定的集群不存在一样。

3. 如果要克隆的集群已加密，请包含 KmsKeyId 参数以加密克隆的集群。此 kms-key-id 值可以是用于加密原始数据库集群的值，也可以是您自己的 KMS 密钥。您的账户必须具有使用该加密密钥的权限。

   在克隆卷时，目标账户必须具有使用用于加密源集群的加密密钥的权限。Aurora 使用 KmsKeyId 中指定的加密密钥对新的克隆集群进行加密。

   拥有加密密钥的账户必须通过使用密钥策略来向目标账户授予密钥的使用权限。此过程类似于共享加密快照的方式，使用密钥策略来向目标账户授予密钥的使用权限。下面是密钥策略的示例。

```
{
  "Id": "key-policy-1",
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Allow use of the key",
      "Effect": "Allow",
      "Principal": {"AWS": [
        "arn:aws:iam::account_id:user/KeyUser",
        "arn:aws:iam::account_id:root"
      ]},
      "Action": [
        "kms:CreateGrant",
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:DescribeKey"
      ],
      "Resource": "*"
    },
    {
      "Sid": "Allow attachment of persistent resources",
      "Effect": "Allow",
      "Principal": {"AWS": [
        "arn:aws:iam::account_id:user/KeyUser",
        "arn:aws:iam::account_id:root"
```

```
      ]},
      "Action": [
        "kms:CreateGrant",
        "kms:ListGrants",
        "kms:RevokeGrant"
      ],
      "Resource": "*",
      "Condition": {"Bool": {"kms:GrantIsForAWSResource": true}}
    }
  ]
}
```

> **Note**
>
> RestoreDBClusterToPointInTime RDS API 操作仅还原数据库集群，而不还原该数据库集群的数据库实例。要为恢复的数据库集群创建数据库实例，请调用 CreateDBInstance RDS API 操作。在 DBClusterIdentifier 中指定恢复的数据库集群的标识符。只有在完成 RestoreDBClusterToPointInTime 操作并且数据库集群可用后，您才能创建数据库实例。

检查数据库集群是否为跨账户克隆

DBClusters 对象标识每个集群是否是跨账户克隆。在运行 RDS CLI 命令 include-shared 时，可使用 describe-db-clusters 选项查看您有权克隆的集群。但是，您看不到此类集群的大部分配置详细信息。

AWS CLI

检查数据库集群是否为跨账户克隆

- 调用 RDS CLI 命令 describe-db-clusters。

  以下示例说明实际或潜在的跨账户克隆数据库集群在 describe-db-clusters 输出中的显示方式。对于您的 AWS 账户现在拥有的集群，CrossAccountClone 字段指示集群是否为另一个 AWS 账户拥有的数据库集群的克隆。

  在某些情况下，条目具有的 AWS 账号可能与您在 AWS 字段中的 DBClusterArn 账号不同。在此情况下，该条目表示由其他 AWS 账户拥有且可克隆的集群。此类条目仅具有 DBClusterArn

字段。在创建克隆的集群时，指定与原始集群中相同的 StorageEncrypted、Engine 和 EngineVersion 值。

```
$aws rds describe-db-clusters --include-shared --region us-east-1
{
  "DBClusters": [
      {
          "EarliestRestorableTime": "2023-02-01T21:17:54.106Z",
          "Engine": "aurora-mysql",
          "EngineVersion": "8.0.mysql_aurora.3.02.0",
          "CrossAccountClone": false,
...
      },
      {
          "EarliestRestorableTime": "2023-02-09T16:01:07.398Z",
          "Engine": "aurora-mysql",
          "EngineVersion": "8.0.mysql_aurora.3.02.0",
          "CrossAccountClone": true,
...
      },
      {
          "StorageEncrypted": false,
          "DBClusterArn": "arn:aws:rds:us-east-1:12345678:cluster:cluster-
abcdefgh",
          "Engine": "aurora-mysql",
          "EngineVersion": "8.0.mysql_aurora.3.02.0
  ]
}
```

RDS API

检查数据库集群是否为跨账户克隆

- 调用 RDS API 操作 DescribeDBClusters。

  对于您的 AWS 账户现在拥有的集群，CrossAccountClone 字段指示集群是否为另一个 AWS 账户拥有的数据库集群的克隆。在 AWS 字段中具有其他 DBClusterArn 账号的条目表示其他 AWS 账户拥有且可克隆的集群。这些条目仅具有 DBClusterArn 字段。在创建克隆的集群时，指定与原始集群中相同的 StorageEncrypted、Engine 和 EngineVersion 值。

  以下示例显示一个返回值，该值说明了实际和潜在的克隆的集群。

```
{
  "DBClusters": [
      {
          "EarliestRestorableTime": "2023-02-01T21:17:54.106Z",
          "Engine": "aurora-mysql",
          "EngineVersion": "8.0.mysql_aurora.3.02.0",
          "CrossAccountClone": false,
...
      },
      {
          "EarliestRestorableTime": "2023-02-09T16:01:07.398Z",
          "Engine": "aurora-mysql",
          "EngineVersion": "8.0.mysql_aurora.3.02.0",
          "CrossAccountClone": true,
...
      },
      {
          "StorageEncrypted": false,
          "DBClusterArn": "arn:aws:rds:us-east-1:12345678:cluster:cluster-
abcdefgh",
          "Engine": "aurora-mysql",
          "EngineVersion": "8.0.mysql_aurora.3.02.0"
      }
  ]
}
```

将 Aurora 与其他AWS服务集成

将 Amazon Aurora 与其他AWS服务集成，这样便可扩展 Aurora 数据库集群以在AWS云中使用其他功能。

主题

- 将AWS服务与 Amazon Aurora MySQL 集成
- 将AWS服务与 Amazon Aurora PostgreSQL 集成

将AWS服务与 Amazon Aurora MySQL 集成

Amazon Aurora MySQL 与其他AWS服务集成在一起，因此，您可以扩展 Aurora MySQL 数据库集群以在AWS云中使用其他功能。您的 Aurora MySQL 数据库集群可以使用AWS服务来执行以下操作：

- 使用 AWS Lambda 或 `lambda_sync` 本机函数同步或异步地调用 `lambda_async` 函数。或者，使用 AWS Lambda 过程异步地调用 `mysql.lambda_async` 函数。
- 通过使用 `LOAD DATA FROM S3` 或 `LOAD XML FROM S3` 命令，将数据从 Amazon S3 存储桶中存储的文本或 XML 文件加载到您的数据库集群中。
- 通过使用 `SELECT INTO OUTFILE S3` 命令，将数据从您的数据库集群保存到 Amazon S3 存储桶中存储的文本文件。
- 使用 Application Auto Scaling 自动添加或删除 Aurora 副本。有关更多信息，请参阅“Amazon Aurora Auto Scaling 与 Aurora 副本结合使用”。

有关将 Aurora MySQL 与其他AWS服务集成的更多信息，请参阅 将 Amazon Aurora MySQL 与其他AWS服务集成。

将AWS服务与 Amazon Aurora PostgreSQL 集成

Amazon Aurora PostgreSQL 与其他AWS服务集成在一起，因此，您可以扩展 Aurora PostgreSQL 数据库集群以在AWS云中使用附加功能。您的 Aurora PostgreSQL 数据库集群可以使用AWS服务来执行以下操作：

- 使用 Performance Insights 快速收集、查看和评估关系数据库工作负载的性能。
- 使用 Aurora Auto Scaling 自动添加或删除 Aurora 副本。有关更多信息，请参阅“Amazon Aurora Auto Scaling 与 Aurora 副本结合使用”。

有关将 Aurora PostgreSQL 与其他AWS服务集成的更多信息，请参阅 将 Amazon Aurora PostgreSQL 与其他AWS服务集成。

维护 Amazon Aurora 数据库集群

Amazon RDS 会定期对 Amazon RDS 资源执行维护。

主题

- 数据库集群维护更新概述
- 查看待处理维护更新
- 对数据库集群应用更新
- Amazon RDS 维护时段
- 调整首选数据库集群维护时段
- Aurora 数据库集群的自动次要版本升级
- 选择 Aurora MySQL 维护更新的频率

数据库集群维护更新概述

维护通常涉及对数据库集群中以下资源的更新：

- 底层硬件
- 底层操作系统（OS）
- 数据库引擎版本

针对操作系统的更新最常见的原因是安全问题。我们建议您尽快进行更新。有关操作系统更新的更多信息，请参阅对数据库集群应用更新。

主题

- 维护更新期间的离线资源
- 推迟的数据库实例和数据库集群修改
- DescribePendingMaintenanceActions API 的最终一致性

维护更新期间的离线资源

一些维护项目要求 Amazon RDS 使您的数据库集群脱机一小段时间。要求资源脱机的维护项目包括必需的操作系统或数据库修补。仅对与安全性和实例可靠性相关的修补程序自动安排必需的修补。此类补丁很少发生，通常每隔几个月发生一次。它所需要的维护时间很少超过维护窗口的一小部分。

推迟的数据库实例和数据库集群修改

您已选择不立即应用的推迟的数据库集群和实例修改会在维护时段内应用。例如，您可以选择在维护时段内更改数据库实例类或集群或数据库参数组。您使用等待重启设置指定的此类修改不会显示在等待维护列表中。有关修改数据库集群的信息，请参阅修改 Amazon Aurora 数据库集群。

要查看下一个维护时段待处理的修改，请使用 describe-db-clusters AWS CLI 命令并选中 `PendingModifiedValues` 字段。

DescribePendingMaintenanceActions API 的最终一致性

Amazon RDS `DescribePendingMaintenanceActions` API 采用最终一致性模型。这意味着，`DescribePendingMaintenanceActions` 命令的结果可能不会立即对所有后续 RDS 命令可见。当您使用之前的 API 命令后立即使用 `DescribePendingMaintenanceActions` 时，请记住这一点。

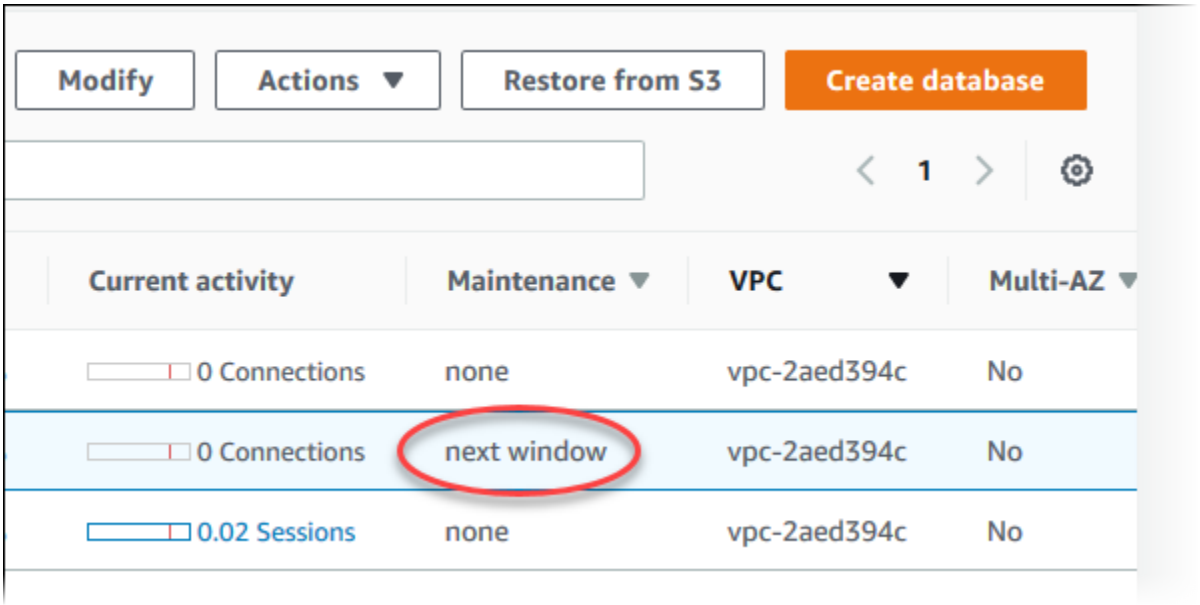
最终一致性可能会影响您管理维护更新的方式。例如，如果您运行 `ApplyPendingMaintenanceActions` 命令来更新数据库集群的数据库引擎版本，则该版本最终将对 `DescribePendingMaintenanceActions` 可见。在这种情况下，`DescribePendingMaintenanceActions` 可能表明维护操作未被应用，即使已应用也是如此。

要管理最终一致性，您可以执行以下操作：

- 请先确认数据库集群的状态，然后运行命令来对其进行修改。使用指数回退算法运行相应的 `DescribePendingMaintenanceActions` 命令，来确保有足够的时间让前一个命令传播遍整个系统。为此，请重复运行 `DescribePendingMaintenanceActions` 命令，以几秒钟的等待时间开始，然后逐渐增加达到五分钟的等待时间。
- 增加后续命令之间的等待时间，即使 `DescribePendingMaintenanceActions` 命令返回准确的响应，也是如此。应用指数回退算法，以几秒钟的等待时间开始，然后逐渐增加达到大约五分钟的等待时间。

查看待处理维护更新

通过使用 RDS 控制台、AWS CLI 或 RDS API 来查看维护更新是否可用于数据库集群。如果某个更新可用，则将在 Amazon RDS 控制台上的数据库集群的维护列中指示它，如下所示。



如果没有维护更新可用于数据库集群，则它的列值为无。

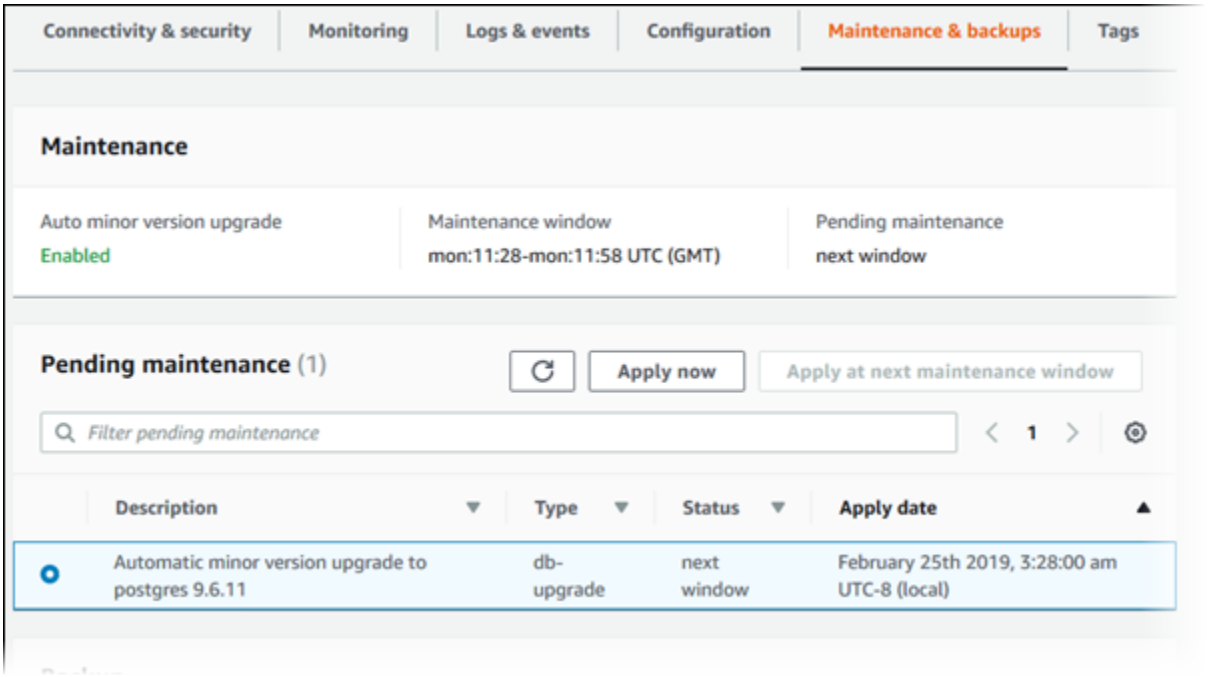
如果有维护更新可用于数据库集群，则可能为以下列值：

- 必需 – 维护操作将应用于资源且不能无限期推迟。
- available (可用) – 维护操作可用，但不会自动应用于资源。您可以手动应用它。
- 下一个窗口 – 维护操作将在下一个维护窗口期间应用于资源。
- In progress (正在进行) – 维护操作正在应用于资源。

如果更新可用，则可执行这些操作之一：

- 如果维护值为 next window (下一时段)，请通过从 Actions (操作) 中选择 Defer upgrade (推迟升级) 来推迟维护项目。如果维护操作已经启动，则无法推迟该操作。
- 立即应用维护项目。
- 计划下一个维护时段内要开始的维护项目。
- 不执行任何操作。

要采取操作，请选择数据库集群以显示其详细信息，然后选择 Maintenance & backups (维护和备份)。将显示待处理维护项目。



维护时段确定待处理的操作何时开始，但不限制这些操作的总运行时间。维护操作不保证在维护时段结束前完成，可以在超出指定的结束时间后继续。有关更多信息，请参阅“Amazon RDS 维护时段”。

有关 Amazon Aurora 引擎更新的信息以及升级和修补这些引擎的说明，请参阅Amazon Aurora MySQL 的数据库引擎更新和Amazon Aurora PostgreSQL 的数据库引擎更新。

您可以通过运行 `describe-pending-maintenance-actions` AWS CLI 命令来查看维护更新是否可用于数据库集群。

有关应用维护更新的信息，请参阅对数据库集群应用更新。

对数据库集群应用更新

通过 Amazon RDS，您可以选择何时应用维护操作。您可通过使用 AWS Management Console、AWS CLI 或 RDS API 来决定 Amazon RDS 何时应用更新。

控制台

管理数据库集群的更新

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择数据库。
3. 选择具有必需更新的数据库集群。
4. 对于操作，请选择下列选项之一：

   - 立即修补
   - 在下一个时段修补

   > **Note**
   >
   > 如果您选择在下一个时段修补，并且以后希望延迟更新，可以选择推迟升级。如果维护操作已经启动，则无法推迟该操作。
   > 要取消维护操作，请修改数据库实例并禁用 Auto minor version upgrade (自动次要版本升级)。

AWS CLI

要将待处理的更新应用于数据库集群，请使用 apply-pending-maintenance-action AWS CLI 命令。

Example

对于 Linux、macOS 或 Unix：

```
aws rds apply-pending-maintenance-action \
    --resource-identifier arn:aws:rds:us-west-2:001234567890:db:mysql-db \
    --apply-action system-update \
    --opt-in-type immediate
```

对于 Windows：

```
aws rds apply-pending-maintenance-action ^
    --resource-identifier arn:aws:rds:us-west-2:001234567890:db:mysql-db ^
    --apply-action system-update ^
    --opt-in-type immediate
```

> **Note**
>
> 要推迟维护操作，请为 undo-opt-in 指定 --opt-in-type。如果维护操作已启动，则无法为 undo-opt-in 指定 --opt-in-type。
> 要取消维护操作，请运行 modify-db-instance AWS CLI 命令并指定 --no-auto-minor-version-upgrade。

要返回具有至少一个待处理更新的资源的列表，请使用 describe-pending-maintenance-actions AWS CLI 命令。

Example

对于 Linux、macOS 或 Unix：

```
aws rds describe-pending-maintenance-actions \
    --resource-identifier arn:aws:rds:us-west-2:001234567890:db:mysql-db
```

对于 Windows：

```
aws rds describe-pending-maintenance-actions ^
    --resource-identifier arn:aws:rds:us-west-2:001234567890:db:mysql-db
```

您还可以通过指定 describe-pending-maintenance-actions AWS CLI 命令的 --filters 参数返回数据库集群的资源列表。--filters 命令的格式是 Name=*filter-name*,Value=*resource-id*,...。

下面是筛选条件的 Name 参数中接受的值：

- db-instance-id – 接受数据库实例标识符或 Amazon Resource Names (ARN) 的列表。返回的列表只包括这些标识符或 ARN 标识的数据库实例的待处理维护操作。
- db-cluster-id – 接受 Amazon Aurora 的数据库集群标识符或 ARN 的列表。返回的列表只包括这些标识符或 ARN 标识的数据库集群的待处理维护操作。

例如，以下示例返回 sample-cluster1 和 sample-cluster2 数据库集群的待处理维护操作。

Example

对于 Linux、macOS 或 Unix：

```
aws rds describe-pending-maintenance-actions \
 --filters Name=db-cluster-id,Values=sample-cluster1,sample-cluster2
```

对于 Windows：

```
aws rds describe-pending-maintenance-actions ^
 --filters Name=db-cluster-id,Values=sample-cluster1,sample-cluster2
```

RDS API

要将更新应用于数据库集群，请调用 Amazon RDS API ApplyPendingMaintenanceAction 操作。

要返回具有至少一个待处理更新的资源的列表，请调用 Amazon RDS API DescribePendingMaintenanceActions 操作。

Amazon RDS 维护时段

维护时段是每周时间间隔，在此期间会应用任何系统更改。每个数据库集群都具有每周维护时段。可以利用维护时段来控制何时进行修改和软件修补。有关调整维护时段的更多信息，请参阅调整首选数据库集群维护时段。

在应用维护时，RDS 会使用您的数据库集群上的一些资源。您可观察到对性能的影响甚微。对于数据库实例来说，在极少数情况下，可能需要多可用区故障转移才能完成维护更新。

如果在给定的周内安排了维护事件，则将在您确定的 30 分钟维护时段内启动维护。大部分维护事件也将在 30 分钟的维护时段内完成，但较大的维护事件可能需要 30 分钟以上的时间才能完成。数据库集群停止后，维护时段将暂停。

这个 30 分钟维护时段是随机从每个地区的 8 小时时间段中选择出来的。如果在创建数据库集群时未指定维护时段，则 RDS 在该星期内随机选择的某一天中分配 30 分钟的维护时段。

在下面可以找到为每个区域分配默认维护时段的时间段。

区域名称	区域	时间段
美国东部（俄亥俄州）	us-east-2	03:00–11:00 UTC
美国东部（弗吉尼亚北部）	us-east-1	03:00–11:00 UTC
美国西部（加利福尼亚北部）	us-west-1	06:00–14:00 UTC
美国西部（俄勒冈州）	us-west-2	06:00–14:00 UTC
非洲（开普敦）	af-south-1	03:00–11:00 UTC
亚太地区（香港）	ap-east-1	06:00–14:00 UTC
亚太地区（海得拉巴）	ap-south-2	06:30–14:30 UTC
亚太地区（雅加达）	ap-southeast-3	08:00–16:00 UTC
亚太地区（墨尔本）	ap-southeast-4	11:00–19:00 UTC
亚太地区（孟买）	ap-south-1	06:00–14:00 UTC
Asia Pacific (Osaka)	ap-northeast-3	22:00–23:59 UTC
Asia Pacific (Seoul)	ap-northeast-2	13:00–21:00 UTC
亚太地区（新加坡）	ap-southeast-1	14:00–22:00 UTC
亚太地区（悉尼）	ap-southeast-2	12:00–20:00 UTC
亚太地区（东京)	ap-northeast-1	13:00–21:00 UTC
加拿大（中部）	ca-central-1	03:00–11:00 UTC
加拿大西部（卡尔加里）	ca-west-1	18:00–02:00 UTC

区域名称	区域	时间段
中国（北京）	cn-north-1	06:00–14:00 UTC
中国（宁夏）	cn-northwest-1	06:00–14:00 UTC
Europe (Frankfurt)	eu-central-1	21:00–05:00 UTC
欧洲地区（爱尔兰）	eu-west-1	22:00–06:00 UTC
欧洲地区（伦敦）	eu-west-2	22:00–06:00 UTC
欧洲（米兰）	eu-south-1	02:00–10:00 UTC
欧洲（巴黎）	eu-west-3	23:59–07:29 UTC
欧洲（西班牙）	eu-south-2	02:00–10:00 UTC
Europe (Stockholm)	eu-north-1	23:00–07:00 UTC
欧洲（苏黎世）	eu-central-2	02:00–10:00 UTC
以色列（特拉维夫）	il-central-1	03:00–11:00 UTC
中东（巴林）	me-south-1	06:00–14:00 UTC
中东（阿联酋）	me-central-1	05:00–13:00 UTC
南美洲（圣保罗）	sa-east-1	00:00–08:00 UTC
AWS GovCloud（美国东部）	us-gov-east-1	17:00–01:00 UTC
AWS GovCloud（美国西部）	us-gov-west-1	06:00–14:00 UTC

调整首选数据库集群维护时段

Aurora 数据库集群维护时段应当选在使用量最小的时段上，因而可能必须不时予以修改。仅当正在应用的更新需要中断时，您的数据库集群才会在这段时间内不可用。执行必要更新所需的中断持续时间会非常短。

控制台

调整首选数据库集群维护时段

1. 登录AWS Management Console并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Databases (数据库)。
3. 选择要更改维护时段的数据库集群。
4. 选择修改。
5. 在维护部分中，更新维护时段。
6. 选择 Continue (继续)。

   在确认页面上，检查您的更改。
7. 要立即将更改应用于维护时段，请在修改计划部分中选择立即。
8. 选择修改集群以保存更改。

   或者，选择 Back 编辑您的更改，或者选择 Cancel 取消更改。

AWS CLI

要调整首选数据库集群维护时段，请使用具有以下参数的 AWS CLI `modify-db-cluster` 命令：

- `--db-cluster-identifier`
- `--preferred-maintenance-window`

Example

以下代码示例将维护时段设置为周二的凌晨 4:00–4:30 (UTC)。

对于 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster \
```

```
--db-cluster-identifier my-cluster \
--preferred-maintenance-window Tue:04:00-Tue:04:30
```

对于 Windows：

```
aws rds modify-db-cluster ^
--db-cluster-identifier my-cluster ^
--preferred-maintenance-window Tue:04:00-Tue:04:30
```

RDS API

要调整首选数据库集群维护时段，请使用带以下参数的 Amazon RDS `ModifyDBCluster` API 操作：

- `DBClusterIdentifier`
- `PreferredMaintenanceWindow`

Aurora 数据库集群的自动次要版本升级

自动次要版本升级设置指定 Aurora 是否自动将升级应用于您的数据库集群。这些升级包括新的次要版本，而次要版本包括其他功能以及补丁（其中包含错误修复）。

原定设置情况下，此设置处于启用状态。对于每个新的数据库集群，为此设置选择适当的值。此值基于其重要性、预期生命周期以及每次升级后执行的验证测试量。

有关打开或关闭自动次要版本升级设置的说明，请参阅以下内容：

- 对 Aurora 数据库集群启用自动次要版本升级
- 为 Aurora 数据库集群中的单个数据库实例启用自动次要版本升级

> **Important**
>
> 我们强烈建议，对于新的和现有的数据库集群，将此设置应用于数据库集群，而不是单独应用于集群中的数据库实例。如果对集群中的任何数据库实例关闭了此设置，则不会自动升级数据库集群。

下表显示了在集群和实例级别应用自动次要版本升级设置时的工作原理。

操作	集群设置	实例设置	是否自动升级集群？
您在数据库集群上将其设置为 True。	True	对于所有新实例和现有实例，均为 True	是
您在数据库集群上将其设置为 False。	False	对于所有新实例和现有实例，均为 False	否
之前在数据库集群上将其设置为 True。 您至少在一个数据库实例上将其设置为 False。	更改为 False	对于一个或多个实例为 False	否
之前在数据库集群上将其设置为 False。 您至少对于一个数据库实例（但并非所有实例）将其设置为 True。	False	对于一个或多个实例（但并非所有实例）为 True	否
之前在数据库集群上将其设置为 False。 您在所有数据库实例上将其设置为 True。	更改为 True	对于所有实例为 True	是

事先通过 Amazon RDS 数据库集群事件（类别为 `maintenance`，ID 为 `RDS-EVENT-0156`）与自动次要版本升级进行通信。有关更多信息，请参阅 Aurora 的 Amazon RDS 事件类别和事件消息。

自动升级在维护时段发生。如果数据库集群中的各个数据库实例的维护时段与集群维护时段不同，则集群维护时段优先。

有关 Aurora PostgreSQL 引擎更新的更多信息，请参阅Amazon Aurora PostgreSQL 的数据库引擎更新。

有关 Aurora MySQL 的自动次要版本升级设置的更多信息，请参阅 启用 Aurora MySQL 次要版本之间的自动升级。有关 Aurora MySQL 的引擎更新的一般信息，请参阅 Amazon Aurora MySQL 的数据库引擎更新。

对 Aurora 数据库集群启用自动次要版本升级

按照使用控制台、CLI 和 API 修改数据库集群中的常规程序进行操作。

控制台

在修改数据库集群页面的维护部分中，选中允许自动次要版本升级复选框。

AWS CLI

调用 modify-db-cluster AWS CLI 命令。为 `--db-cluster-identifier` 选项指定数据库集群的名称，并为 `--auto-minor-version-upgrade` 选项指定 `true`。（可选）指定 `--apply-immediately` 选项，立即为数据库集群启用此设置。

RDS API

调用 ModifyDBCluster API 操作，并为 `DBClusterIdentifier` 参数指定数据库集群的名称，为 `AutoMinorVersionUpgrade` 参数指定 `true`。（可选）将 `ApplyImmediately` 参数设置为 `true`，立即为数据库集群启用此设置。

为 Aurora 数据库集群中的单个数据库实例启用自动次要版本升级

按照修改数据库集群中的数据库实例中的常规程序进行操作。

控制台

在修改数据库实例页面的维护部分中，选中允许自动次要版本升级复选框。

AWS CLI

调用 modify-db-instance AWS CLI 命令。为 `--db-instance-identifier` 选项指定数据库实例的名称，为 `true` 选项指定 `--auto-minor-version-upgrade`。（可选）指定 `--apply-immediately` 选项，立即为数据库实例启用此设置。为集群中的每个数据库实例运行单独的 `modify-db-instance` 命令。

RDS API

调用 ModifyDBInstance API 操作，并为 `DBInstanceIdentifier` 参数指定数据库集群的名称，为 `AutoMinorVersionUpgrade` 参数指定 `true`。（可选）将 `ApplyImmediately`

参数设置为 `true`，立即为数据库实例启用此设置。为集群中的每个数据库实例调用单独的 `ModifyDBInstance` 操作。

您可以使用如下 CLI 命令来检查 Aurora MySQL 集群中所有数据库实例的 `AutoMinorVersionUpgrade` 设置的状态。

```
aws rds describe-db-instances \
  --query '*[].
{DBClusterIdentifier:DBClusterIdentifier,DBInstanceIdentifier:DBInstanceIdentifier,AutoMinorVer
```

该命令产生的输出类似于以下内容：

```
[
  {
      "DBInstanceIdentifier": "db-writer-instance",
      "DBClusterIdentifier": "my-db-cluster-57",
      "AutoMinorVersionUpgrade": true
  },
  {
      "DBInstanceIdentifier": "db-reader-instance1",
      "DBClusterIdentifier": "my-db-cluster-57",
      "AutoMinorVersionUpgrade": false
  },
  {
      "DBInstanceIdentifier": "db-writer-instance2",
      "DBClusterIdentifier": "my-db-cluster-80",
      "AutoMinorVersionUpgrade": true
  },
... output omitted ...
```

在此示例中，数据库集群 `my-db-cluster-57` 的允许自动次要版本升级处于关闭状态，因为对于集群中的其中一个数据库实例关闭了此功能。

选择 Aurora MySQL 维护更新的频率

您可以控制每个数据库集群是经常还是很少进行 Aurora MySQL 升级。最佳选择取决于 Aurora MySQL 使用情况以及在 Aurora 上运行的应用程序的优先级。有关不太需要频繁升级的 Aurora MySQL 长期稳定性 (LTS) 版本的信息，请参阅 Aurora MySQL 长期支持 (LTS) 版本。

如果符合以下部分或全部条件，您可能会选择很少升级 Aurora MySQL 集群：

- 对于 Aurora MySQL 数据库引擎的每次更新，应用程序的测试周期需要很长的时间。
- 很多数据库集群或很多应用程序运行相同的 Aurora MySQL 版本。您希望同时升级所有数据库集群和关联的应用程序。
- 您同时使用 Aurora MySQL 和 RDS for MySQL。您希望将 Aurora MySQL 集群和 RDS for MySQL 数据库实例与同一级别的 MySQL 保持兼容。
- Aurora MySQL 应用程序位于生产环境中或在其他方面对业务至关重要。除了在极少数情况下应用关键补丁以外，您无法承受升级停机。
- Aurora MySQL 应用程序不受在后续 Aurora MySQL 版本中解决的性能问题或功能差异的限制。

如果上述因素适用于您的情况，您可以限制 Aurora MySQL 数据库集群的强制升级次数。为此，您可以在创建或升级该数据库集群时选择称为“长期支持”(LTS) 版本的特定 Aurora MySQL 版本。这样做可以最大限度减少该数据库集群的升级周期数、测试周期数以及与升级相关的中断次数。

如果符合以下部分或全部条件，您可能会选择经常升级 Aurora MySQL 集群：

- 应用程序的测试周期简单明了。
- 应用程序仍处于开发阶段。
- 数据库环境使用各种不同的 Aurora MySQL 版本或 Aurora MySQL 和 RDS for MySQL 版本。每个 Aurora MySQL 集群具有自己的升级周期。
- 在增加 Aurora MySQL 使用量之前，您正在等待改进特定的性能或功能。

如果上述因素适用于您的情况，您可以使 Aurora 更频繁地应用重要升级。为此，将 Aurora MySQL 数据库集群升级到比 LTS 版本更高的 Aurora MySQL 版本。这样做可以使您更快地获得最新的性能增强、错误修复和功能。

重启 Amazon Aurora 数据库集群或 Amazon Aurora 数据库实例

通常出于维护原因，您可能需要重启数据库集群或该集群内的某些实例。例如，假设您修改了参数组中的参数或将不同参数组与集群关联起来。在这些情况下，您必须重启集群才能使更改生效。同样，您可以重启集群中的一个或多个读取器数据库实例。您可以为单个实例安排重启操作，以最大限度地减少整个集群的停机时间。

重启集群中每个数据库实例所需的时间取决于重启时的数据库活动，还取决于特定数据库引擎的恢复过程。如果可行，请在开始重启过程前减少该特定实例上的数据库活动。这样做可以减少重启数据库所需的时间。

您只能在集群中的每个数据库实例处于可用状态时重新启动该实例。由于多种原因，数据库实例可能无法使用。其中包括正处于停止状态的集群、正在对实例应用的修改以及维护窗口操作（如版本升级）。

重启数据库实例会重新启动数据库引擎流程。重启数据库实例将导致短暂中断，在此期间，数据库实例状态将设置为正在重启。

> **Note**
>
> 如果数据库实例未使用对其关联的数据库参数组所做的最新更改，则 AWS Management Console 将显示状态为 pending-reboot 的数据库参数组。pending-reboot 参数组状态不会在下一个维护时段期间导致自动重启。要向该数据库实例应用最新的参数更改，请手动重启数据库实例。有关参数组的更多信息，请参阅 Amazon Aurora 的参数组。

主题

- 重启 Aurora 集群内的数据库实例
- 在具有读取可用性的情况下重启 Aurora 集群
- 在没有读取可用性功能的情况下重启 Aurora 集群
- 检查 Aurora 集群和实例的正常运行时间
- Aurora 重启操作示例

重启 Aurora 集群内的数据库实例

此过程是您在利用 Aurora 执行重启时采取的最重要的操作。许多维护过程都涉及按特定顺序重启一个或多个 Aurora 数据库实例。

控制台

重启数据库实例

1. 登录AWS Management Console并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择数据库，然后选择要重启的数据库实例。
3. 对于操作，选择重启。

   将显示重启数据库实例页。
4. 选择重新引导以重新引导您的数据库实例。

   或者选择 Cancel（取消）。

AWS CLI

要使用 AWS CLI 重新引导数据库实例，请调用 `reboot-db-instance` 命令。

Example

对于 Linux、macOS 或 Unix：

```
aws rds reboot-db-instance \
    --db-instance-identifier mydbinstance
```

对于 Windows：

```
aws rds reboot-db-instance ^
    --db-instance-identifier mydbinstance
```

RDS API

要使用 Amazon RDS API 重启数据库实例，请调用 `RebootDBInstance` 操作。

在具有读取可用性的情况下重启 Aurora 集群

在使用读取可用性特征的情况下，您可以重启 Aurora 集群的写入器实例，而无需重启主要或辅助数据库集群中的读取器实例。这样做有助于在重启写入器实例时维持集群的高可用性，以进行读取操作。

您可以稍后按照方便的时间安排重启读取器实例。例如，在生产集群中，您可以一次重启一个读取器实例，但只有在主实例完成重启后才能开始。对于您重启的每个数据库实例，请按照 重启 Aurora 集群内的数据库实例 中的步骤操作。

主数据库集群的读取可用性特征在 Aurora MySQL 版本 2.10 及更高版本中可用。辅助数据库集群的读取可用性在 Aurora MySQL 版本 3.06 及更高版本中可用。

对于 Aurora PostgreSQL，原定设置情况下，此功能在以下版本中可用：

- 15.2 及更高的 15 版本
- 14.7 及更高的 14 版本
- 13.10 及更高的 13 版本
- 12.14 及更高的 12 版本

有关 Aurora PostgreSQL 中读取可用性功能的更多信息，请参阅提高 Aurora 副本的读取可用性。

在推出此特征之前，重启主实例会同时导致每个读取器实例重启。如果您的 Aurora 集群运行的是旧版本，请改用在没有读取可用性功能的情况下重启 Aurora 集群中的重启过程。

> **Note**
>
> 对于 Aurora MySQL 版本低于 3.06 的 Aurora 全局数据库，在具有读取可用性的 Aurora 数据库集群中对重启行为的更改有所不同。如果重启 Aurora 全局数据库中主集群的写入器实例，则主集群中的读取器实例仍可用。但是，所有辅助集群中的数据库实例都会同时重启。Aurora PostgreSQL 12.16、13.12、14.9、15.4 及更高版本的 Aurora 全局数据库支持改进的读取可用性特征的有限版本。

在对集群参数组进行更改后，您需要经常重启集群。您可以按照 Amazon Aurora 的参数组 中的过程进行参数更改。假设您重启 Aurora 集群中的写入器数据库实例，以便对集群参数应用更改。部分或全部读取器数据库实例可能会继续使用旧的参数设置。但是，不同的参数设置不会影响集群的数据完整性。任何影响数据文件条理性的集群参数只能由写入器数据库实例使用。

例如，在 Aurora MySQL 集群中，您可以先更新写入器实例上的集群参数（如 `binlog_format` 和 `innodb_purge_threads`），然后再对读取器实例进行相同的操作。只有写入器实例需要编写二进制日志并清除撤消记录。对于更改查询解释 SQL 语句或查询输出方式的参数，您可能需要注意立即重启读取器实例。这样做是为了避免查询期间出现意外的应用程序行为。例如，假设您更改了

lower_case_table_names 参数并重启了写入器实例。在这种情况下，只有全部重启后，读取器实例才可能可以访问新建的表。

有关所有 Aurora MySQL 集群参数的列表，请参阅 集群级别的参数。

有关所有 Aurora PostgreSQL 集群参数的列表，请参阅Aurora PostgreSQL 集群级参数。

> **Tip**
>
> 如果您的集群正在处理高吞吐量的工作负载，Aurora MySQL 可能仍会重启部分读取器实例以及写入器实例。
> 在故障转移操作过程中，重新启动次数减少也会如此。Aurora MySQL 在故障转移期间仅重启写入器数据库实例和故障转移目标。集群中的其他读取器数据库实例仍然可以通过与读取器终端节点的连接来继续处理查询。因此，您可以通过在集群中拥有多个读取器数据库实例来提高故障转移期间的可用性。

在没有读取可用性功能的情况下重启 Aurora 集群

如果没有读取可用性功能，则是通过重启整个 Aurora 数据库集群的写入器数据库实例来重启该集群。为此，请按照重启 Aurora 集群内的数据库实例中的过程操作

重启写入器数据库实例还会为集群中的每个读取器数据库实例启动重启操作。这样，任何集群范围内的参数更改都将同时应用于所有数据库实例。但是，重启所有数据库实例会导致集群短暂中断。在写入器数据库实例完成重启并变为可用状态之前，读取器数据库实例一直不可用。

这种重启行为适用于在 Aurora MySQL 版本 2.09 及更低版本中创建的所有数据库集群。

对于 Aurora PostgreSQL，此行为适用于以下版本：

- 14.6 及更低的 14 版本
- 13.9 及更低的 13 版本
- 12.13 及更低的 12 版本
- 所有 PostgreSQL 11 版本

在 RDS 控制台中，写入器数据库实例在数据库页面的角色列下的值为 Writer。在RDS CLI 中，describe-db-clusters 命令的输出包括部分 DBClusterMembers。表示写入器数据库实例的 DBClusterMembers 元素在 true 字段的值为 IsClusterWriter。

> **Important**
>
> 在具有读取可用性功能的情况下，Aurora MySQL 和 Aurora PostgreSQL 中的重启行为有所不同：读取器数据库实例通常在您重启写入器实例时仍然可用。然后，您可以在方便的时候重启读取器实例。如果希望某些读取器实例始终可用，您可以按错开的时间安排重启读取器实例。有关更多信息，请参阅 在具有读取可用性的情况下重启 Aurora 集群。

检查 Aurora 集群和实例的正常运行时间

您可以检查和监控 Aurora 集群中每个数据库实例自上次重启以来的时间长度。Amazon CloudWatch 指标 EngineUptime 会报告自上次启动数据库实例以来的秒数。您可以在某个时间点检查此指标，以了解数据库实例的正常运行时间。您还可以随着时间的推移监控此指标，以检测实例的重启时间。

您还可以在集群级别检查 EngineUptime 指标。Minimum 和 Maximum 会维度报告集群中所有数据库实例的最小和最大的正常运行时间值。要检查集群中的读取器实例最近的重启时间或出于其他原因重启的时间，请使用 Minimum 维度监控集群级指标。要在不重启的情况下检查集群中运行时间最长的实例，请使用 Maximum 维度监控集群级指标。例如，您可能需要确认在进行配置更改后，集群中的所有数据库实例都已重启。

> **Tip**
>
> 对于长期监控，我们建议监控单个实例的 EngineUptime 指标，而不是在集群级进行监控。新数据库实例添加到集群后，集群级 EngineUptime 指标将设置为零。此类集群更改可以作为维护和扩缩操作（例如由 Auto Scaling 执行的操作）的一部分。

以下 CLI 示例显示了如何检查集群中写入器和读取器实例的 EngineUptime 指标。这些示例使用名为 tpch100g 的集群。此集群有一个写入器数据库实例 instance-1234。它还有两个读取器数据库实例（instance-7448 和 instance-6305）。

首先，reboot-db-instance 命令会重启其中一个读取器实例。wait 命令会等到实例完成重启。

```
$ aws rds reboot-db-instance --db-instance-identifier instance-6305
{
    "DBInstance": {
        "DBInstanceIdentifier": "instance-6305",
        "DBInstanceStatus": "rebooting",
```

```
...
$ aws rds wait db-instance-available --db-instance-id instance-6305
```

CloudWatch get-metric-statistics 命令每隔一分钟便会检查过去五分钟内的 EngineUptime 指标。instance-6305 实例的正常运行时间将重置为零并再次开始向上计数。以下适用于 Linux 的 AWS CLI 示例使用 $( ) 变量替换在 CLI 命令中插入适当的时间戳。它还使用 Linux sort 命令按照指标收集时间对输出进行排序。该时间戳值是每行输出中的第三个字段。

```
$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" \
  --period 60 --namespace "AWS/RDS" --statistics Maximum \
  --dimensions Name=DBInstanceIdentifier,Value=instance-6305 --output text \
  | sort -k 3
EngineUptime
DATAPOINTS 231.0 2021-03-16T18:19:00+00:00 Seconds
DATAPOINTS 291.0 2021-03-16T18:20:00+00:00 Seconds
DATAPOINTS 351.0 2021-03-16T18:21:00+00:00 Seconds
DATAPOINTS 411.0 2021-03-16T18:22:00+00:00 Seconds
DATAPOINTS 471.0 2021-03-16T18:23:00+00:00 Seconds
```

由于集群中的一个实例已重启，因此集群的最短正常运行时间将重置为零。由于集群中至少有一个数据库实例仍然可用，因此集群的最长正常运行时间不会重置。

```
$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" \
  --period 60 --namespace "AWS/RDS" --statistics Minimum \
  --dimensions Name=DBClusterIdentifier,Value=tpch100g --output text \
  | sort -k 3
EngineUptime
DATAPOINTS 63099.0 2021-03-16T18:12:00+00:00 Seconds
DATAPOINTS 63159.0 2021-03-16T18:13:00+00:00 Seconds
DATAPOINTS 63219.0 2021-03-16T18:14:00+00:00 Seconds
DATAPOINTS 63279.0 2021-03-16T18:15:00+00:00 Seconds
DATAPOINTS 51.0 2021-03-16T18:16:00+00:00 Seconds

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" \
  --period 60 --namespace "AWS/RDS" --statistics Maximum \
  --dimensions Name=DBClusterIdentifier,Value=tpch100g --output text \
  | sort -k 3
EngineUptime
DATAPOINTS 63389.0 2021-03-16T18:16:00+00:00 Seconds
```

```
DATAPOINTS 63449.0 2021-03-16T18:17:00+00:00 Seconds
DATAPOINTS 63509.0 2021-03-16T18:18:00+00:00 Seconds
DATAPOINTS 63569.0 2021-03-16T18:19:00+00:00 Seconds
DATAPOINTS 63629.0 2021-03-16T18:20:00+00:00 Seconds
```

之后另一个 reboot-db-instance 命令将重启集群的写入器实例。另一个 wait 命令会暂停，直至写入器实例完成重启。

```
$ aws rds reboot-db-instance --db-instance-identifier instance-1234
{
  "DBInstanceIdentifier": "instance-1234",
  "DBInstanceStatus": "rebooting",
...
$ aws rds wait db-instance-available --db-instance-id instance-1234
```

现在，写入器实例的 EngineUptime 指标显示实例 instance-1234 最近已重启。读取器实例 instance-6305 也会与写入器实例一起自动重启。此集群运行 Aurora MySQL 2.09，因此在写入器实例重启时读取器实例不会保持运行状态。

```
$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" \
  --period 60 --namespace "AWS/RDS" --statistics Maximum \
  --dimensions Name=DBInstanceIdentifier,Value=instance-1234 --output text \
  | sort -k 3
EngineUptime
DATAPOINTS 63749.0 2021-03-16T18:22:00+00:00 Seconds
DATAPOINTS 63809.0 2021-03-16T18:23:00+00:00 Seconds
DATAPOINTS 63869.0 2021-03-16T18:24:00+00:00 Seconds
DATAPOINTS 41.0 2021-03-16T18:25:00+00:00 Seconds
DATAPOINTS 101.0 2021-03-16T18:26:00+00:00 Seconds

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" \
  --period 60 --namespace "AWS/RDS" --statistics Maximum \
  --dimensions Name=DBInstanceIdentifier,Value=instance-6305 --output text \
  | sort -k 3
EngineUptime
DATAPOINTS 411.0 2021-03-16T18:22:00+00:00 Seconds
DATAPOINTS 471.0 2021-03-16T18:23:00+00:00 Seconds
DATAPOINTS 531.0 2021-03-16T18:24:00+00:00 Seconds
DATAPOINTS 49.0 2021-03-16T18:26:00+00:00 Seconds
```

Aurora 重启操作示例

以下 Aurora MySQL 示例显示了 Aurora 数据库集群中读取器和写入器数据库实例的多种重启操作组合。每次重启后，SQL 查询会显示集群中各实例的正常运行时间。

主题

- 查找 Aurora 集群的写入器实例和读取器实例
- 重启单个读取器实例
- 重启写入器实例
- 独立重启写入器和读取器
- 将集群参数更改应用于 Aurora MySQL 2.10 版集群

查找 Aurora 集群的写入器实例和读取器实例

在具有多个数据库实例的 Aurora MySQL 集群中，重要的是要知道哪个是写入器实例，哪些是读取器实例。在发生故障转移操作时，写入器实例和读取器实例还可以切换角色。因此，最好在执行任何需要写入器或读取器实例的操作之前执行如下检查。在这种情况下，`False` 的 `IsClusterWriter` 值识别读取器实例 `instance-6305` 和 `instance-7448`。`True` 值识别写入器实例 `instance-1234`。

```
$ aws rds describe-db-clusters --db-cluster-id tpch100g \
  --query "*[].['Cluster:',DBClusterIdentifier,DBClusterMembers[*].
['Instance:',DBInstanceIdentifier,IsClusterWriter]]" \
  --output text
Cluster:     tpch100g
Instance:    instance-6305    False
Instance:    instance-7448    False
Instance:    instance-1234    True
```

在我们开始重启示例之前，写入器实例的正常运行时间约为一周。此示例中的 SQL 查询显示了一种特定于 MySQL 的正常运行时间检查方法。您可以在数据库应用程序中使用此技术。有关使用 AWS CLI 并适用于两个 Aurora 引擎的另一种技术，请参阅检查 Aurora 集群和实例的正常运行时间。

```
$ mysql -h instance-7448.a12345.us-east-1.rds.amazonaws.com -P 3306 -u my-user -p
...
mysql> select date_sub(now(), interval variable_value second) "Last Startup",
    -> time_format(sec_to_time(variable_value),'%Hh %im') as "Uptime"
    -> from performance_schema.global_status
    -> where variable_name='Uptime';
+----------------------------+---------+
```

```
| Last Startup               | Uptime  |
+----------------------------+---------+
| 2021-03-08 17:49:06.000000 | 174h 42m|
+----------------------------+---------+
```

重启单个读取器实例

此示例重启其中一个读取器数据库实例。也许是此实例被一个巨大查询或许多并发连接过载了。或者它可能由于网络问题落后于写入器实例。开始重启操作后，示例使用 wait 命令暂停，直至实例变为可用。届时，该实例的正常运行时间为几分钟。

```
$ aws rds reboot-db-instance --db-instance-identifier instance-6305
{
    "DBInstance": {
        "DBInstanceIdentifier": "instance-6305",
        "DBInstanceStatus": "rebooting",
...
    }
}
$ aws rds wait db-instance-available --db-instance-id instance-6305
$ mysql -h instance-6305.a12345.us-east-1.rds.amazonaws.com -P 3306 -u my-user -p
...
mysql> select date_sub(now(), interval variable_value second) "Last Startup",
    -> time_format(sec_to_time(variable_value),'%Hh %im') as "Uptime"
    -> from performance_schema.global_status
    -> where variable_name='Uptime';
+----------------------------+---------+
| Last Startup               | Uptime  |
+----------------------------+---------+
| 2021-03-16 00:35:02.000000 | 00h 03m |
+----------------------------+---------+
```

重启读取器实例不会影响写入器实例的正常运行时间。它的正常运行时间仍为大约一周。

```
$ mysql -h instance-7448.a12345.us-east-1.rds.amazonaws.com -P 3306 -u my-user -p
...
mysql> select date_sub(now(), interval variable_value second) "Last Startup",
    -> time_format(sec_to_time(variable_value),'%Hh %im') as "Uptime"
    -> from performance_schema.global_status where variable_name='Uptime';
+----------------------------+----------+
| Last Startup               | Uptime   |
+----------------------------+----------+
```

```
| 2021-03-08 17:49:06.000000 | 174h 49m |
+----------------------------+----------+
```

重启写入器实例

此示例将重启写入器实例。此集群正在运行 Aurora MySQL 2.09 版。由于 Aurora MySQL 版本低于 2.10，因此重启写入器实例也会重启集群中的所有读取器实例。

`wait` 命令会暂停，直至重启完成。现在，该实例的正常运行时间重置为零。写入器数据库实例和读取器数据库实例的重启操作所需的时间可能大不相同。写入器数据库实例和读取器数据库实例根据其角色执行不同类型的清理操作。

```
$ aws rds reboot-db-instance --db-instance-identifier instance-1234
{
    "DBInstance": {
        "DBInstanceIdentifier": "instance-1234",
        "DBInstanceStatus": "rebooting",
...
    }
}
$ aws rds wait db-instance-available --db-instance-id instance-1234
$ mysql -h instance-1234.a12345.us-east-1.rds.amazonaws.com -P 3306 -u my-user -p
...
mysql> select date_sub(now(), interval variable_value second) "Last Startup",
    -> time_format(sec_to_time(variable_value),'%Hh %im') as "Uptime"
    -> from performance_schema.global_status where variable_name='Uptime';
+----------------------------+---------+
| Last Startup               | Uptime  |
+----------------------------+---------+
| 2021-03-16 00:40:27.000000 | 00h 00m |
+----------------------------+---------+
```

在写入器数据库实例重启后，两个读取器数据库实例也会重置其正常运行时间。重启写入器实例也会导致读取器实例重启。此行为适用于 Aurora PostgreSQL 集群和低于 2.10 版的 Aurora MySQL 集群。

```
$ mysql -h instance-7448.a12345.us-east-1.rds.amazonaws.com -P 3306 -u my-user -p
...
mysql> select date_sub(now(), interval variable_value second) "Last Startup",
    -> time_format(sec_to_time(variable_value),'%Hh %im') as "Uptime"
    -> from performance_schema.global_status where variable_name='Uptime';
+----------------------------+---------+
| Last Startup               | Uptime  |
```

```
+----------------------------+---------+
| 2021-03-16 00:40:35.000000 | 00h 00m |
+----------------------------+---------+

$ mysql -h instance-6305.a12345.us-east-1.rds.amazonaws.com -P 3306 -u my-user -p
...
mysql> select date_sub(now(), interval variable_value second) "Last Startup",
    -> time_format(sec_to_time(variable_value),'%Hh %im') as "Uptime"
    -> from performance_schema.global_status where variable_name='Uptime';
+----------------------------+---------+
| Last Startup               | Uptime  |
+----------------------------+---------+
| 2021-03-16 00:40:33.000000 | 00h 01m |
+----------------------------+---------+
```

独立重启写入器和读取器

接下来的几个示例显示了运行 Aurora MySQL 2.10 版的集群。在此 Aurora MySQL 版本及更高版本中，您可以重启写入器实例，而不会导致所有读取器实例重启。这样一来，当您重启写入器实例时，查询密集型应用程序不会出现任何中断。您可以稍后重启读取器实例。可以在查询流量偏低时重启这些实例。也可以一次重启一个读取器实例。这样一来，至少有一个读取器实例始终可供您的应用程序的查询流量使用。

以下示例使用名为 `cluster-2393` 且运行 Aurora MySQL `5.7.mysql_aurora.2.10.0` 版的集群。此集群有一个写入器实例 (`instance-9404`) 和三个读取器实例（`instance-6772`、`instance-2470` 和 `instance-5138`）。

```
$ aws rds describe-db-clusters --db-cluster-id cluster-2393 \
  --query "*[].['Cluster:',DBClusterIdentifier,DBClusterMembers[*].
['Instance:',DBInstanceIdentifier,IsClusterWriter]]" \
  --output text
Cluster:        cluster-2393
Instance:       instance-5138        False
Instance:       instance-2470        False
Instance:       instance-6772        False
Instance:       instance-9404        True
```

通过 `uptime` 命令检查每个数据库实例的 `mysql` 值表明，每个数据库实例的正常运行时间大致相同。例如，这是 `instance-5138` 的正常运行时间。

```
mysql> SHOW GLOBAL STATUS LIKE 'uptime';
+---------------+-------+
```

```
| Variable_name | Value |
+---------------+-------+
| Uptime        | 3866  |
+---------------+-------+
```

通过使用 CloudWatch，我们可以获得相应的正常运行时间信息，而无需实际登录实例。这样，管理员可以监控数据库，但无法查看或更改任何表数据。在这种情况下，我们指定一个五分钟时间段，然后每分钟检查一次正常运行时间值。不断增加的正常运行时间值表明，这些实例在此期间没有重启。

```
$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBInstanceIdentifier,Value=instance-9404 \
  --output text | sort -k 3
EngineUptime
DATAPOINTS 4648.0 2021-03-17T23:42:00+00:00 Seconds
DATAPOINTS 4708.0 2021-03-17T23:43:00+00:00 Seconds
DATAPOINTS 4768.0 2021-03-17T23:44:00+00:00 Seconds
DATAPOINTS 4828.0 2021-03-17T23:45:00+00:00 Seconds
DATAPOINTS 4888.0 2021-03-17T23:46:00+00:00 Seconds

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBInstanceIdentifier,Value=instance-6772 \
  --output text | sort -k 3
EngineUptime
DATAPOINTS 4315.0 2021-03-17T23:42:00+00:00 Seconds
DATAPOINTS 4375.0 2021-03-17T23:43:00+00:00 Seconds
DATAPOINTS 4435.0 2021-03-17T23:44:00+00:00 Seconds
DATAPOINTS 4495.0 2021-03-17T23:45:00+00:00 Seconds
DATAPOINTS 4555.0 2021-03-17T23:46:00+00:00 Seconds
```

现在重启其中一个读取器实例 `instance-5138`。等待实例在重启后再次可用。现在监控五分钟内的正常运行时间，可以看到正常运行时间在此期间已重置为零。最近的正常运行时间值是在重启完成后 5 秒钟测得的。

```
$ aws rds reboot-db-instance --db-instance-identifier instance-5138
{
  "DBInstanceIdentifier": "instance-5138",
  "DBInstanceStatus": "rebooting"
}
```

```
$ aws rds wait db-instance-available --db-instance-id instance-5138

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBInstanceIdentifier,Value=instance-5138 \
  --output text | sort -k 3
EngineUptime
DATAPOINTS 4500.0 2021-03-17T23:46:00+00:00 Seconds
DATAPOINTS 4560.0 2021-03-17T23:47:00+00:00 Seconds
DATAPOINTS 4620.0 2021-03-17T23:48:00+00:00 Seconds
DATAPOINTS 4680.0 2021-03-17T23:49:00+00:00 Seconds
DATAPOINTS  5.0 2021-03-17T23:50:00+00:00 Seconds
```

接下来，对写入器实例 instance-9404 执行重启。比较写入器实例和其中一个读取器实例的正常运行时间值。借此，我们可以看到重启写入器并没有导致读取器重启。在 Aurora MySQL 2.10 之前的版本中，所有读取器的正常运行时间值将与写入器同时重置。

```
$ aws rds reboot-db-instance --db-instance-identifier instance-9404
{
  "DBInstanceIdentifier": "instance-9404",
  "DBInstanceStatus": "rebooting"
}
$ aws rds wait db-instance-available --db-instance-id instance-9404

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBInstanceIdentifier,Value=instance-9404 \
  --output text | sort -k 3
EngineUptime
DATAPOINTS 371.0 2021-03-17T23:57:00+00:00 Seconds
DATAPOINTS 431.0 2021-03-17T23:58:00+00:00 Seconds
DATAPOINTS 491.0 2021-03-17T23:59:00+00:00 Seconds
DATAPOINTS 551.0 2021-03-18T00:00:00+00:00 Seconds
DATAPOINTS  37.0  2021-03-18T00:01:00+00:00 Seconds

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBInstanceIdentifier,Value=instance-6772 \
  --output text | sort -k 3
EngineUptime
```

```
DATAPOINTS 5215.0 2021-03-17T23:57:00+00:00 Seconds
DATAPOINTS 5275.0 2021-03-17T23:58:00+00:00 Seconds
DATAPOINTS 5335.0 2021-03-17T23:59:00+00:00 Seconds
DATAPOINTS 5395.0 2021-03-18T00:00:00+00:00 Seconds
DATAPOINTS 5455.0 2021-03-18T00:01:00+00:00 Seconds
```

为确保所有读取器实例对配置参数的更改与写入器实例所做的更改相同，请在重启写入器实例之后重启所有读取器实例。此示例将重启所有读取器，然后等到所有读取器都可用后再继续操作。

```
$ aws rds reboot-db-instance --db-instance-identifier instance-6772
{
  "DBInstanceIdentifier": "instance-6772",
  "DBInstanceStatus": "rebooting"
}

$ aws rds reboot-db-instance --db-instance-identifier instance-2470
{
  "DBInstanceIdentifier": "instance-2470",
  "DBInstanceStatus": "rebooting"
}

$ aws rds reboot-db-instance --db-instance-identifier instance-5138
{
  "DBInstanceIdentifier": "instance-5138",
  "DBInstanceStatus": "rebooting"
}

$ aws rds wait db-instance-available --db-instance-id instance-6772
$ aws rds wait db-instance-available --db-instance-id instance-2470
$ aws rds wait db-instance-available --db-instance-id instance-5138
```

现在可以看到，写入器数据库实例的正常运行时间最长。该实例的正常运行时间值在整个监控期内稳步增加。读取器数据库实例都在读取器之后重启。我们可以看到监控期内每个读取器重启且其正常运行时间重置为零的时间点。

```
$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBInstanceIdentifier,Value=instance-9404 \
  --output text | sort -k 3
EngineUptime
DATAPOINTS 457.0 2021-03-18T00:08:00+00:00 Seconds
```

```
DATAPOINTS 517.0 2021-03-18T00:09:00+00:00 Seconds
DATAPOINTS 577.0 2021-03-18T00:10:00+00:00 Seconds
DATAPOINTS 637.0 2021-03-18T00:11:00+00:00 Seconds
DATAPOINTS  697.0 2021-03-18T00:12:00+00:00 Seconds

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBInstanceIdentifier,Value=instance-2470 \
  --output text | sort -k 3
EngineUptime
DATAPOINTS 5819.0 2021-03-18T00:08:00+00:00 Seconds
DATAPOINTS  35.0  2021-03-18T00:09:00+00:00 Seconds
DATAPOINTS 95.0 2021-03-18T00:10:00+00:00 Seconds
DATAPOINTS 155.0 2021-03-18T00:11:00+00:00 Seconds
DATAPOINTS 215.0 2021-03-18T00:12:00+00:00 Seconds

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBInstanceIdentifier,Value=instance-5138 \
  --output text | sort -k 3
EngineUptime
DATAPOINTS 1085.0 2021-03-18T00:08:00+00:00 Seconds
DATAPOINTS 1145.0 2021-03-18T00:09:00+00:00 Seconds
DATAPOINTS 1205.0 2021-03-18T00:10:00+00:00 Seconds
DATAPOINTS  49.0  2021-03-18T00:11:00+00:00 Seconds
DATAPOINTS 109.0 2021-03-18T00:12:00+00:00 Seconds
```

将集群参数更改应用于 Aurora MySQL 2.10 版集群

以下示例演示了如何将参数更改应用于 Aurora MySQL 2.10 集群中的所有数据库实例。使用此 Aurora MySQL 版本，您可以独立重启写入器实例和所有读取器实例。

该示例使用 MySQL 配置参数 `lower_case_table_names` 进行说明。当写入器数据库实例和读取器数据库实例的此参数设置不同时，查询可能无法访问使用大写字母或混合大小写名称声明的表。或者，如果两个表名仅在字母大小写方面有所不同，则查询可能会访问错误的表。

此示例显示如何通过检查每个实例的 `IsClusterWriter` 属性来确定集群中的写入器实例和读取器实例。该集群名为 `cluster-2393`。集群有一个名为 `instance-9404` 的写入器实例。集群中的读取器实例名为 `instance-5138` 和 `instance-2470`。

```
$ aws rds describe-db-clusters --db-cluster-id cluster-2393 \
```

```
  --query '*[].[DBClusterIdentifier,DBClusterMembers[*].
[DBInstanceIdentifier,IsClusterWriter]]' \
  --output text
cluster-2393
instance-5138        False
instance-2470        False
instance-9404        True
```

为了展示更改 lower_case_table_names 参数的影响，我们设置了两个数据库集群参数组。lower-case-table-names-0 参数组将此参数设置为 0。lower-case-table-names-1 参数组将此参数组设置为 1。

```
$ aws rds create-db-cluster-parameter-group --description 'lower-case-table-names-0' \
  --db-parameter-group-family aurora-mysql5.7 \
  --db-cluster-parameter-group-name lower-case-table-names-0
{
    "DBClusterParameterGroup": {
        "DBClusterParameterGroupName": "lower-case-table-names-0",
        "DBParameterGroupFamily": "aurora-mysql5.7",
        "Description": "lower-case-table-names-0"
    }
}

$ aws rds create-db-cluster-parameter-group --description 'lower-case-table-names-1' \
  --db-parameter-group-family aurora-mysql5.7 \
  --db-cluster-parameter-group-name lower-case-table-names-1
{
    "DBClusterParameterGroup": {
        "DBClusterParameterGroupName": "lower-case-table-names-1",
        "DBParameterGroupFamily": "aurora-mysql5.7",
        "Description": "lower-case-table-names-1"
    }
}

$ aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name lower-case-table-names-0 \
  --parameters
 ParameterName=lower_case_table_names,ParameterValue=0,ApplyMethod=pending-reboot
{
    "DBClusterParameterGroupName": "lower-case-table-names-0"
}

$ aws rds modify-db-cluster-parameter-group \
```

```
  --db-cluster-parameter-group-name lower-case-table-names-1 \
    --parameters
 ParameterName=lower_case_table_names,ParameterValue=1,ApplyMethod=pending-reboot
{
    "DBClusterParameterGroupName": "lower-case-table-names-1"
}
```

lower_case_table_names 的默认值为 0。使用此参数设置时，表 foo 与表 FOO 不相同。此示例验证参数是否仍使用默认设置。之后，该示例创建了三个表，区别仅在于其名称中字母的大小写。

```
mysql> create database lctn;
Query OK, 1 row affected (0.07 sec)

mysql> use lctn;
Database changed
mysql> select @@lower_case_table_names;
+--------------------------+
| @@lower_case_table_names |
+--------------------------+
|                        0 |
+--------------------------+

mysql> create table foo (s varchar(128));
mysql> insert into foo values ('Lowercase table name foo');

mysql> create table Foo (s varchar(128));
mysql> insert into Foo values ('Mixed-case table name Foo');

mysql> create table FOO (s varchar(128));
mysql> insert into FOO values ('Uppercase table name FOO');

mysql> select * from foo;
+--------------------------+
| s                        |
+--------------------------+
| Lowercase table name foo |
+--------------------------+

mysql> select * from Foo;
+--------------------------+
| s                        |
+--------------------------+
| Mixed-case table name Foo |
```

```
+--------------------------+

mysql> select * from FOO;
+--------------------------+
| s                        |
+--------------------------+
| Uppercase table name FOO |
+--------------------------+
```

接下来，将数据库参数组与集群关联以将 lower_case_table_names 参数设置为 1。此更改仅在每个数据库实例重启后生效。

```
$ aws rds modify-db-cluster --db-cluster-identifier cluster-2393 \
  --db-cluster-parameter-group-name lower-case-table-names-1
{
  "DBClusterIdentifier": "cluster-2393",
  "DBClusterParameterGroup": "lower-case-table-names-1",
  "Engine": "aurora-mysql",
  "EngineVersion": "5.7.mysql_aurora.2.10.0"
}
```

首先重启写入器数据库实例。然后等待该实例再次可用。届时，连接到写入器终端节点并验证写入器实例的参数值是否已更改。SHOW TABLES 命令确认数据库包含三个不同的表。但是，引用名为 foo、Foo 或 FOO 的表的查询都会访问名称全小写的表。foo

```
# Rebooting the writer instance
$ aws rds reboot-db-instance --db-instance-identifier instance-9404
$ aws rds wait db-instance-available --db-instance-id instance-9404
```

现在，使用集群终端节点的查询显示了参数更改的影响。无论查询中的表名是大写、小写还是混合大小写，SQL 语句都会访问名称全小写的表。

```
mysql> select @@lower_case_table_names;
+--------------------------+
| @@lower_case_table_names |
+--------------------------+
|                        1 |
+--------------------------+

mysql> use lctn;
mysql> show tables;
+----------------+
```

```
| Tables_in_lctn |
+----------------+
| FOO            |
| Foo            |
| foo            |
+----------------+

mysql> select * from foo;
+--------------------------+
| s                        |
+--------------------------+
| Lowercase table name foo |
+--------------------------+

mysql> select * from Foo;
+--------------------------+
| s                        |
+--------------------------+
| Lowercase table name foo |
+--------------------------+

mysql> select * from FOO;
+--------------------------+
| s                        |
+--------------------------+
| Lowercase table name foo |
+--------------------------+
```

下一个示例显示的查询与前一个示例相同。在这种情况下，查询使用读取器终端节点并在其中一个读取器数据库实例上运行。这些实例尚未重启。因此，他们仍然使用 lower_case_table_names 参数的原始设置。这意味着查询可以访问 foo、Foo 和 FOO 表中的每一个。

```
mysql> select @@lower_case_table_names;
+--------------------------+
| @@lower_case_table_names |
+--------------------------+
|                        0 |
+--------------------------+

mysql> use lctn;

mysql> select * from foo;
+--------------------------+
```

```
| s                         |
+---------------------------+
| Lowercase table name foo |
+---------------------------+

mysql> select * from Foo;
+---------------------------+
| s                          |
+---------------------------+
| Mixed-case table name Foo |
+---------------------------+

mysql> select * from FOO;
+---------------------------+
| s                         |
+---------------------------+
| Uppercase table name FOO |
+---------------------------+
```

接下来，重启其中一个读取器实例，然后等待它再次可用。

```
$ aws rds reboot-db-instance --db-instance-identifier instance-2470
{
  "DBInstanceIdentifier": "instance-2470",
  "DBInstanceStatus": "rebooting"
}
$ aws rds wait db-instance-available --db-instance-id instance-2470
```

在连接到 instance-2470 的实例终端节点时，查询显示新参数已生效。

```
mysql> select @@lower_case_table_names;
+---------------------------+
| @@lower_case_table_names |
+---------------------------+
|                         1 |
+---------------------------+
```

此时，集群中的两个读取器实例以不同的 lower_case_table_names 设置运行。因此，与集群读取器终端节点的任何连接都会对此设置使用一个不可预测的值。务必立即重启另一个读取器实例，以便它们使用一致的设置。

```
$ aws rds reboot-db-instance --db-instance-identifier instance-5138
```

```
{
  "DBInstanceIdentifier": "instance-5138",
  "DBInstanceStatus": "rebooting"
}
$ aws rds wait db-instance-available --db-instance-id instance-5138
```

以下示例确认所有读取器实例具有相同的 lower_case_table_names 参数设置。这些命令会检查每个读取器实例的 lower_case_table_names 设置值。使用读取器终端节点的同一命令表明，与读取器终端节点的每个连接都使用其中一个读取器实例，但该实例不可预测。

```
# Check lower_case_table_names setting on each reader instance.

$ mysql -h instance-5138.a12345.us-east-1.rds.amazonaws.com \
  -u my-user -p -e 'select @@aurora_server_id, @@lower_case_table_names'
+--------------------------+--------------------------+
| @@aurora_server_id       | @@lower_case_table_names |
+--------------------------+--------------------------+
| instance-5138            |                        1 |
+--------------------------+--------------------------+

$ mysql -h instance-2470.a12345.us-east-1.rds.amazonaws.com \
  -u my-user -p -e 'select @@aurora_server_id, @@lower_case_table_names'
+--------------------------+--------------------------+
| @@aurora_server_id       | @@lower_case_table_names |
+--------------------------+--------------------------+
| instance-2470            |                        1 |
+--------------------------+--------------------------+

# Check lower_case_table_names setting on the reader endpoint of the cluster.

$ mysql -h cluster-2393.cluster-ro-a12345.us-east-1.rds.amazonaws.com \
  -u my-user -p -e 'select @@aurora_server_id, @@lower_case_table_names'
+--------------------------+--------------------------+
| @@aurora_server_id       | @@lower_case_table_names |
+--------------------------+--------------------------+
| instance-5138            |                        1 |
+--------------------------+--------------------------+

# Run query on writer instance

$ mysql -h cluster-2393.cluster-a12345.us-east-1.rds.amazonaws.com \
  -u my-user -p -e 'select @@aurora_server_id, @@lower_case_table_names'
+--------------------------+--------------------------+
```

```
| @@aurora_server_id       | @@lower_case_table_names |
+--------------------------+--------------------------+
| instance-9404            |                        1 |
+--------------------------+--------------------------+
```

随着参数更改应用到各处，我们可以看到设置 lower_case_table_names=1 的影响。无论表名是 foo、Foo 还是 FOO，查询都会将名称转换为 foo，并在各种情况下都访问同一个表。

```
mysql> use lctn;

mysql> select * from foo;
+--------------------------+
| s                        |
+--------------------------+
| Lowercase table name foo |
+--------------------------+

mysql> select * from Foo;
+--------------------------+
| s                        |
+--------------------------+
| Lowercase table name foo |
+--------------------------+

mysql> select * from FOO;
+--------------------------+
| s                        |
+--------------------------+
| Lowercase table name foo |
+--------------------------+
```

对 Amazon Aurora 数据库集群进行失效转移

可以对 Aurora 数据库集群执行手动失效转移，例如，当您要将预置的写入器数据库实例替换为 Aurora Serverless v2 写入器实例时。

Aurora 通过两种方式之一失效转移到新的主数据库实例：

- 通过将现有读取器数据库实例提升为新的主实例
- 创建新的主实例

如果数据库集群具有一个或多个读取器数据库，则在故障事件期间，某个读取器将被提升为主实例。要提高数据库集群的可用性，建议您在两个或更多的不同可用区中创建至少一个或多个读取器实例。有关失效转移机制的更多信息，请参阅 Aurora 数据库集群的容错能力。

可以使用 AWS Management Console、AWS CLI 或 RDS API 来执行手动失效转移。

控制台

对数据库集群进行失效转移

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择数据库，然后在要进行失效转移的数据库集群中选择一个数据库实例。
3. 对于 Actions（操作），请选择 Failover（故障转移）。

   此时会显示确认页面。
4. 选择 Failover（故障转移）。

   数据库页面显示数据库集群状态为正在失效转移。失效转移完成后，状态将恢复为可用，并显示新的和以前的主数据库实例的角色。

AWS CLI

要使用 AWS CLI 对数据库集群进行失效转移，请调用 failover-db-cluster 命令。指定以下参数：

- `--db-cluster-identifier` – 要进行失效转移的数据库集群。
- `--target-db-instance-identifier` – 要提升为主数据库实例的数据库实例的名称。

Example

对于 Linux、macOS 或 Unix：

```
aws rds failover-db-cluster \
    --db-cluster-identifier mydbcluster \
    --target-db-instance-identifier mydbcluster-instance-2
```

对于 Windows：

```
aws rds failover-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --target-db-instance-identifier mydbcluster-instance-2
```

RDS API

要使用 Amazon RDS API 修改数据库集群，请调用 FailoverDBCluster 操作。指定以下参数：

- DBClusterIdentifier
- TargetDBInstanceIdentifier

删除 Aurora 数据库集群和数据库实例

当您不再需要 Aurora 数据库集群时，可以将其删除。删除集群会删除包含您所有数据的集群卷。删除集群之前，您可以保存数据的快照。您可以稍后还原快照，以创建包含相同数据的新集群。

您还可以从集群中删除数据库实例，同时保留集群本身及其包含的数据。如果集群不繁忙，或者您不需要多个数据库实例的计算容量，则删除数据库实例可以帮助您降低费用。

主题

- 删除 Aurora 数据库集群
- Aurora 集群的删除保护
- 删除已停止的 Aurora 集群
- 删除作为只读副本的 Aurora MySQL 集群
- 删除集群时的最终快照
- 从 Aurora 数据库集群中删除数据库实例

删除 Aurora 数据库集群

Aurora 不提供删除数据库集群的单步方法。此设计选择旨在防止您意外丢失数据或使应用程序离线。Aurora 应用程序通常是任务关键型应用程序，需要高可用性。因此，Aurora 通过添加和删除数据库实例，可以轻松地向上和向下扩展集群的容量。删除数据库集群本身需要您进行单独删除。

使用以下常规程序从集群中删除所有数据库实例，然后删除集群本身。

1. 删除集群中的所有读取器实例。使用 从 Aurora 数据库集群中删除数据库实例 中的程序。

   如果集群有任何读取器实例，删除其中一个实例只会降低集群的计算能力。删除读取器实例首先可确保集群在整个程序中始终可用，并且不会执行不必要的故障切换操作。
2. 从集群中删除写入器实例。再次使用 从 Aurora 数据库集群中删除数据库实例 中的程序。

   当您删除数据库实例时，集群及其关联的集群卷仍保留，即使在删除所有数据库实例后也是如此。
3. 删除数据库集群。
   - AWS Management Console – 选择集群，然后从操作菜单中选择删除。您可以选择以下选项来保留集群中的数据，以备日后需要时使用：
     - 创建集群卷的最终快照。默认设置是创建最终快照。
     - 保留自动备份。默认设置是不保留自动备份。

> **Note**
>
> 不会保留 Aurora Serverless v1 数据库集群的自动备份。

Aurora 还会要求您确认是否要删除集群。

- CLI 和 API – 调用 `delete-db-cluster` CLI 命令或 `DeleteDBCluster` API 操作。您可以选择以下选项来保留集群中的数据，以备日后需要时使用：
  - 创建集群卷的最终快照。
  - 保留自动备份。

> **Note**
>
> 不会保留 Aurora Serverless v1 数据库集群的自动备份。

主题

- 删除空 Aurora 集群
- 删除具有单个数据库实例的 Aurora 集群
- 删除包含多个数据库实例的 Aurora 集群

删除空 Aurora 集群

您可以使用 AWS Management Console、AWS CLI 或 Amazon RDS API 删除空的数据库集群。

> **Tip**
>
> 您可以保留没有数据库实例的集群，这样就能够在不产生集群 CPU 费用的前提下保留数据。您可以通过为集群创建一个或多个新的数据库实例来快速开始使用集群。您可以在集群没有任何关联的数据库实例的情况下对集群执行 Aurora 特定的管理操作。您无法访问数据或执行任何需要连接到数据库实例的操作。

控制台

删除数据库集群

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择数据库，然后选择要删除的数据库集群。
3. 对于操作，选择删除。
4. 要为数据库集群创建最终数据库快照，请选择是否创建最终快照？。这是默认设置。
5. 如果选择创建最终快照，请输入最终快照名称。
6. 要保留自动备份，请选择 Retain automated backups (保留自动备份)。这不是默认设置。
7. 在框中输入 **`delete me`**。
8. 选择删除。

CLI

要使用 AWS CLI 删除空 Aurora 数据库集群，请调用 delete-db-cluster 命令。

假设空集群 `deleteme-zero-instances` 仅用于开发和测试，不包含任何重要数据。在这种情况下，删除集群时不需要保留集群卷的快照。以下示例展示了集群不包含任何数据库实例，并随后在不创建最终快照或保留自动备份的情况下删除空集群。

```
$ aws rds describe-db-clusters --db-cluster-identifier deleteme-zero-instances --output
 text \
  --query '*[].["Cluster:",DBClusterIdentifier,DBClusterMembers[*].
["Instance:",DBInstanceIdentifier,IsClusterWriter]]
Cluster:        deleteme-zero-instances

$ aws rds delete-db-cluster --db-cluster-identifier deleteme-zero-instances \
  --skip-final-snapshot \
  --delete-automated-backups
{
  "DBClusterIdentifier": "deleteme-zero-instances",
  "Status": "available",
  "Engine": "aurora-mysql"
}
```

RDS API

要使用 Amazon RDS API 删除空的 Aurora 数据库集群，请调用 DeleteDBCluster 操作。

删除具有单个数据库实例的 Aurora 集群

您可以删除最后一个数据库实例，即使数据库集群启用了删除保护也是如此。在这种情况下，数据库集群本身仍然存在，您的数据将被保留。您可以将新的数据库实例附加到集群以再次访问数据。

以下示例显示当集群仍有任何关联的数据库实例时，该 `delete-db-cluster` 命令不起作用。此集群有单个写入器数据库实例。当我们检查集群中的数据库实例时，我们会检查每个实例的 `IsClusterWriter` 属性。集群可能有零个或一个写入器数据库实例。`true` 的值表示写入器数据库实例。`false` 的值表示读取器数据库实例。集群可能有零个、一个或多个读取数据库实例。在这种情况下，我们使用 `delete-db-instance` 命令删除写入器数据库实例。一旦数据库实例进入 `deleting` 状态，我们也可以删除集群。另外对于此示例，假设集群不包含任何值得保留的数据。因此，我们不会创建集群卷的快照或保留自动备份。

```
$ aws rds delete-db-cluster --db-cluster-identifier deleteme-writer-only --skip-final-
snapshot
An error occurred (InvalidDBClusterStateFault) when calling the DeleteDBCluster
 operation:
  Cluster cannot be deleted, it still contains DB instances in non-deleting state.

$ aws rds describe-db-clusters --db-cluster-identifier deleteme-writer-only \
  --query '*[].[DBClusterIdentifier,Status,DBClusterMembers[*].
[DBInstanceIdentifier,IsClusterWriter]]'
[
    [
        "deleteme-writer-only",
        "available",
        [
            [
                "instance-2130",
                true
            ]
        ]
    ]
]

$ aws rds delete-db-instance --db-instance-identifier instance-2130
{
  "DBInstanceIdentifier": "instance-2130",
```

```
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql"
}

$ aws rds delete-db-cluster --db-cluster-identifier deleteme-writer-only \
  --skip-final-snapshot \
  --delete-automated-backups
{
  "DBClusterIdentifier": "deleteme-writer-only",
  "Status": "available",
  "Engine": "aurora-mysql"
}
```

删除包含多个数据库实例的 Aurora 集群

如果您的集群包含多个数据库实例，通常会有一个写入器实例和一个或多个读取器实例。读取器实例通过在写入器实例遇到问题时处于备用状态以进行接管，来帮助实现高可用性。您还可以使用读取器实例向上扩展集群以处理读取密集型工作负载，而不会增加写入程序实例的开销。

要删除具有多个读取器数据库实例的集群，首先删除读取器实例，然后删除写入器实例。删除写入器实例会原地保留集群及其数据。您可以通过单独的操作删除集群。

- 有关删除 Aurora 数据库实例的程序，请参阅 从 Aurora 数据库集群中删除数据库实例。
- 有关在 Aurora 集群中删除写入器数据库实例的程序，请参阅 删除具有单个数据库实例的 Aurora 集群。
- 有关删除空 Aurora 集群的程序，请参阅 删除空 Aurora 集群。

此示例 CLI 说明如何删除包含写入器数据库实例和单个读取器数据库实例的集群。`describe-db-clusters` 输出显示 `instance-7384` 是写入器实例，而 `instance-1039` 是读取器实例。该示例首先删除读取器实例，因为在读取器实例仍存在的情况下删除写入器实例将导致故障转移操作。如果您也计划删除读取器实例，将读取器实例提升到写入器没有意义。同样，假设这些 `db.t2.small` 实例仅用于开发和测试，因此删除操作会跳过最终快照且不保留自动备份。

```
$ aws rds delete-db-cluster --db-cluster-identifier deleteme-writer-and-reader --skip-
final-snapshot

An error occurred (InvalidDBClusterStateFault) when calling the DeleteDBCluster
 operation:
  Cluster cannot be deleted, it still contains DB instances in non-deleting state.
```

```
$ aws rds describe-db-clusters --db-cluster-identifier deleteme-writer-and-reader --
output text \
  --query '*[].["Cluster:",DBClusterIdentifier,DBClusterMembers[*].
["Instance:",DBInstanceIdentifier,IsClusterWriter]]
Cluster:        deleteme-writer-and-reader
Instance:       instance-1039  False
Instance:       instance-7384   True

$ aws rds delete-db-instance --db-instance-identifier instance-1039
{
  "DBInstanceIdentifier": "instance-1039",
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql"
}

$ aws rds delete-db-instance --db-instance-identifier instance-7384
{
  "DBInstanceIdentifier": "instance-7384",
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql"
}

$ aws rds delete-db-cluster --db-cluster-identifier deleteme-writer-and-reader \
  --skip-final-snapshot \
  --delete-automated-backups
{
  "DBClusterIdentifier": "deleteme-writer-and-reader",
  "Status": "available",
  "Engine": "aurora-mysql"
}
```

以下示例说明如何删除包含写入器数据库实例和多个读取器数据库实例的数据库集群。它使用 describe-db-clusters 命令的简明输出来获取写入器和读取器实例的报告。同样，我们在删除写入数据库实例之前会删除所有读取器数据库实例。无需考虑我们删除读取器数据库实例的顺序。

假设此包含多个数据库实例的集群确实包含值得保留的数据。因此，本示例中的 delete-db-cluster 命令包括 --no-skip-final-snapshot 和 --final-db-snapshot-identifier 参数，用于指定要创建的快照的详细信息。它还包括用于保留自动备份的 --no-delete-automated-backups 参数。

```
$ aws rds describe-db-clusters --db-cluster-identifier deleteme-multiple-readers --
output text \
```

```
  --query '*[].["Cluster:",DBClusterIdentifier,DBClusterMembers[*].
["Instance:",DBInstanceIdentifier,IsClusterWriter]]
Cluster:        deleteme-multiple-readers
Instance:       instance-1010   False
Instance:       instance-5410   False
Instance:       instance-9948   False
Instance:       instance-8451   True

$ aws rds delete-db-instance --db-instance-identifier instance-1010
{
  "DBInstanceIdentifier": "instance-1010",
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql"
}

$ aws rds delete-db-instance --db-instance-identifier instance-5410
{
  "DBInstanceIdentifier": "instance-5410",
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql"
}

$ aws rds delete-db-instance --db-instance-identifier instance-9948
{
  "DBInstanceIdentifier": "instance-9948",
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql"
}

$ aws rds delete-db-instance --db-instance-identifier instance-8451
{
  "DBInstanceIdentifier": "instance-8451",
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql"
}

$ aws rds delete-db-cluster --db-cluster-identifier deleteme-multiple-readers \
  --no-delete-automated-backups \
  --no-skip-final-snapshot \
  --final-db-snapshot-identifier deleteme-multiple-readers-final-snapshot
{
  "DBClusterIdentifier": "deleteme-multiple-readers",
  "Status": "available",
  "Engine": "aurora-mysql"
```

```
}
```

以下示例说明如何确认 Aurora 创建了请求的快照。您可以通过指定特定快照的标识符 deleteme-multiple-readers-final-snapshot 来请求其详细信息。您还可以通过指定集群标识符 deleteme-multiple-readers 来获取已删除集群的所有快照的报告。这两个命令都返回有关同一快照的信息。

```
$ aws rds describe-db-cluster-snapshots \
  --db-cluster-snapshot-identifier deleteme-multiple-readers-final-snapshot
{
    "DBClusterSnapshots": [
        {
            "AvailabilityZones": [],
            "DBClusterSnapshotIdentifier": "deleteme-multiple-readers-final-snapshot",
            "DBClusterIdentifier": "deleteme-multiple-readers",
            "SnapshotCreateTime": "11T01:40:07.354000+00:00",
            "Engine": "aurora-mysql",
...

$ aws rds describe-db-cluster-snapshots --db-cluster-identifier deleteme-multiple-
readers
{
    "DBClusterSnapshots": [
        {
            "AvailabilityZones": [],
            "DBClusterSnapshotIdentifier": "deleteme-multiple-readers-final-snapshot",
            "DBClusterIdentifier": "deleteme-multiple-readers",
            "SnapshotCreateTime": "11T01:40:07.354000+00:00",
            "Engine": "aurora-mysql",
...
```

Aurora 集群的删除保护

您无法删除已启用删除保护的集群。您可以删除集群中的数据库实例，但不能删除集群本身。这样，包含所有数据的集群卷就可以免遭意外删除。Aurora 会为数据库集群强制执行删除保护，无论您尝试使用控制台、AWS CLI 还是 RDS API 删除集群。

在使用 AWS Management Console 创建生产数据库集群时，将默认启用删除保护。但是，如果您使用 AWS CLI 或 API 创建集群，将默认禁用删除保护。启用或禁用删除保护不会导致中断。要能够删除集群，请修改集群并禁用删除保护。有关打开和关闭删除保护的更多信息，请参阅 使用控制台、CLI 和 API 修改数据库集群。

> **Tip**
>
> 即使删除了所有数据库实例，您也可以通过在集群中创建新的数据库实例来访问数据。

删除已停止的 Aurora 集群

如果集群处于 `stopped` 状态，则无法将其删除。在这种情况下，请在删除集群之前启动集群。有关更多信息，请参阅“启动 Aurora 数据库集群”。

删除作为只读副本的 Aurora MySQL 集群

对于 Aurora MySQL，如果同时满足以下两个条件，则无法删除数据库集群中的数据库实例：

- 该数据库集群是另一个 Aurora 数据库集群的只读副本。
- 该数据库实例是该数据库集群的唯一实例。

在这种情况下，要删除数据库实例，请先提升数据库集群，以使其不再是只读副本。升级完成后，可以删除数据库集群中的最终数据库实例。有关更多信息，请参阅“跨 AWS 区域复制 Amazon Aurora MySQL 数据库集群”。

删除集群时的最终快照

在本节中，我们将给出示例以展示如何在删除 Aurora 集群时选择是否拍摄最终快照。如果选择拍摄最终快照，但指定的名称与现有快照的名称相同，则操作将停止并显示错误消息。在这种情况下，请检查快照详细信息，以确认它是否代表您当前的详细信息，还是代表较旧的快照。如果现有快照没有您想要保留的最新数据，请重命名快照并重试，或者为 final snapshot 参数指定其他名称。

从 Aurora 数据库集群中删除数据库实例

作为删除整个集群过程的一部分，您可以从 Aurora 数据库集群中删除数据库实例。如果您的集群包含一定数量的数据库实例，则删除集群需要删除其中的每个数据库实例。您还可以在保持集群运行的同时从集群中删除一个或多个读取器实例。您可以在集群不繁忙时这样做，以减少计算容量和相关费用。

若要删除某个数据库实例，您必须指定该实例的名称。

您可以使用 AWS Management Console、AWS CLI 或 RDS API 删除数据库实例。

> **Note**
>
> 在删除 Aurora 副本时，将立即删除其实例终端节点，并将 Aurora 副本从读取器终端节点中删除。如果在正待删除的 Aurora 副本上运行语句，则有 3 分钟宽限期。现有语句可在此宽限期内完成。当此宽限期结束后，将关闭并删除 Aurora 副本。

对于 Aurora 数据库集群，删除数据库实例不一定会删除整个集群。您可以在集群不繁忙时删除 Aurora 集群中的数据库实例，以减少计算容量和相关费用。有关具有一个数据库实例或零个数据库实例的 Aurora 集群的特殊情况的信息，请参阅 删除具有单个数据库实例的 Aurora 集群 和 删除空 Aurora 集群。

> **Note**
>
> 当数据库集群启用了删除保护时，您无法删除该集群。有关更多信息，请参阅“Aurora 集群的删除保护”。
> 您可以通过修改数据库集群来禁用删除保护。有关更多信息，请参阅 修改 Amazon Aurora 数据库集群。

控制台

删除数据库集群中的数据库实例

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Databases (数据库)，然后选择要删除的数据库实例。
3. 对于 Actions，选择 Delete。
4. 在框中输入 **`delete me`**。
5. 选择删除。

AWS CLI

要使用 AWS CLI 删除数据库实例，请调用 delete-db-instance 命令并指定 `--db-instance-identifier` 值。

Example

对于 Linux、macOS 或 Unix：

```
aws rds delete-db-instance \
    --db-instance-identifier mydbinstance
```

对于 Windows：

```
aws rds delete-db-instance ^
    --db-instance-identifier mydbinstance
```

RDS API

要使用 Amazon RDS API 删除数据库实例，请调用 DeleteDBInstance 操作并指定 `DBInstanceIdentifier` 参数。

> **Note**
>
> 当数据库实例的状态为 `deleting` 时，其 CA 证书值不会显示在 RDS 控制台中或 AWS CLI 命令或 RDS API 操作的输出中。有关 CA 证书的更多信息，请参阅 使用 SSL/TLS 加密与数据库集群的连接。

为 Amazon Aurora 和 Amazon RDS 资源添加标签

Amazon RDS 标签是由您定义的名称-值对，与某种 Amazon RDS 资源（例如，数据库实例或数据库快照）关联。此名称也叫键。您可以选择为键提供值。

可以使用 AWS Management Console、AWS CLI 或 Amazon RDS API 添加、列出和删除 Amazon RDS 资源上的标签。在使用 CLI 或 API 时，确保提供要使用的 RDS 资源的 Amazon 资源名称（ARN）。有关构造 ARN 的详细信息，请参阅构建 Amazon RDS 的 ARN。

可以使用标签向您的 Aurora 和 Amazon RDS 资源添加元数据。您可以使用标签自行添加有关数据库实例、快照、Aurora 集群等资源的符号。此举可以帮助您记录 Aurora 和 Amazon RDS 资源。您还可以将标签与自动维护程序结合使用。

特别是，您可以将这些标签用于 IAM policy。您可以使用这些标签管理对 Aurora 和 Amazon RDS 资源的访问，并控制可将什么操作应用于这些资源。您还可以将具有类似标签的资源的费用分组在一起，使用标签来跟踪成本。

您可以标记以下 Aurora 和 Amazon RDS 资源：

- 数据库实例数
- 数据库集群
- Aurora 全局集群
- 数据库集群端点
- 只读副本
- 数据库快照
- 数据库集群快照
- 预留数据库实例
- 事件订阅
- 数据库选项组
- 数据库参数组
- 数据库集群参数组
- 数据库子网组
- RDS 代理
- RDS Proxy 端点

- 蓝绿部署
- 零 ETL 集成

> **Note**
>
> 目前，您无法使用 AWS Management Console 标记 RDS 代理和 RDS 代理端点。

主题

- 为什么要使用 Amazon RDS 资源标签？
- Amazon RDS 资源标签的工作原理
- 标记 Amazon RDS 资源的最佳实践
- 复制标签到数据库集群快照
- 在 Amazon RDS 中添加和删除标签
- 教程：使用标签指定要停止哪些 Aurora 数据库集群

为什么要使用 Amazon RDS 资源标签？

您可使用标签执行以下操作：

- 按应用程序、项目、部门、环境等对 RDS 资源进行分类。例如，您可以使用标签键定义类别，其中标签值是类别中的项目。您可以创建标签 `environment=prod`。或者，您可以定义标签键 `project` 和标签值 `Salix`，这表示 Amazon RDS 资源将分配给 Salix 项目。
- 自动执行资源管理任务。例如，您可以为已标记 `environment=prod` 的实例创建一个维护时段，该时段不同于已标记 `environment=test` 的实例的时段。您还可以为已标记 `environment=prod` 的实例配置自动数据库快照。
- 使用 IAM 策略控制对 RDS 资源的访问。您可以使用全局 `aws:ResourceTag/`*`tag-key`* 条件键完成此操作。例如，策略可能只允许 `DBAdmin` 组中的用户修改已标记 `environment=prod` 的数据库实例。有关使用 IAM 策略管理对已标记资源的访问的信息，请参阅 Amazon Aurora 的 Identity and Access Management，以及《AWS Identity and Access Management 用户指南》中的控制对 AWS 资源的访问。
- 根据标签监控资源。例如，您可以为已标记 `environment=prod` 的数据库实例创建 Amazon CloudWatch 控制面板。

- 通过将具有类似标签的资源的费用分组在一起来跟踪成本。例如，如果您使用 `project=Salix` 标记与 Salix 项目关联的 RDS 资源，则可以为该项目生成成本报告并为该项目分配费用。有关更多信息，请参阅 如何将 AWS 账单与 Amazon RDS 中的标签结合使用。

Amazon RDS 资源标签的工作原理

AWS不会对您的标签应用任何语义意义。标签严格按字符串进行解释。

主题

- Amazon RDS 中的标签集
- Amazon RDS 中的标签结构
- 符合标记条件的 Amazon RDS 资源
- 如何将 AWS 账单与 Amazon RDS 中的标签结合使用

Amazon RDS 中的标签集

每个 Amazon RDS 资源都有一个名为标签集的容器。该容器包含所有分配给该资源的标签。一个资源只有一个标签集。

一个标签集包含 0 至 50 个标签。如果向 RDS 资源添加一个标签，而该标签的键与某个现有资源标签相同，则新值将覆盖旧值。

Amazon RDS 中的标签结构

RDS 标签的结构如下所示：

标签键

键是标签的必需名称。该字符串值的长度必须在 1 到 128 个 Unicode 字符之间，并且不能带有前缀 `aws:` 或 `rds:`。字符串只能包含 Unicode 字母、数字、空格、_、.、:、/、=、+、- 和 @ 的集合。Java 正则表达式是 `"^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-@]*)$"`。标签键区分大小写。因此，键 `project` 和 `Project` 是不同的。

键对于标签集是唯一的。例如，标签集中不能有键相同但值不同的键-值对，如 `project=Trinity` 和 `project=Xanadu`。

标签值

值是标签的可选字符串值。该字符串值的长度必须在 1 到 256 个 Unicode 字符之间。字符串只能包含 Unicode 字母、数字、空格、_、.、:、/、=、+、- 和 @ 的集合。Java 正则表达式是 "^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-@]*)$"。标签值区分大小写。因此，值 prod 和 Prod 是不同的。

在标签集中，值无需具有唯一性，且可为空值。例如，在 project=Trinity 和 cost-center=Trinity 的标签集中，可以存在键-值对。

符合标记条件的 Amazon RDS 资源

您可以标记以下 Amazon RDS 资源：

- 数据库实例
- 数据库集群
- 数据库集群端点
- 只读副本
- 数据库快照
- 数据库集群快照
- 预留数据库实例
- 事件订阅
- 数据库选项组
- 数据库参数组
- 数据库集群参数组
- 数据库子网组
- RDS 代理
- RDS 代理端点

> **Note**
>
> 目前，您无法使用 AWS Management Console 标记 RDS 代理和 RDS 代理端点。

- 蓝绿部署
- 零 ETL 集成（预览版）

如何将 AWS 账单与 Amazon RDS 中的标签结合使用

使用标签来组织 AWS 账单以反映您自身的成本结构。要执行此操作，请注册以获取包含标签键值的 AWS 账户账单。然后，如需查看组合资源的成本，请按有同样标签键值的资源组织您的账单信息。例如，您可以将特定的应用程序名称用作几个资源的标签，然后组织账单信息，以查看在数个服务中的使用该应用程序的总成本。有关更多信息，请参阅 AWS Billing 用户指南 中的使用成本分配标签。

成本分配标签如何与数据库集群快照结合使用

您可以向数据库集群快照添加标签。但是，您的账单不会反映此分组。必须满足以下条件，才能将成本分配标签应用于数据库集群快照：

- 标签必须附加到父数据库实例。
- 父数据库实例必须与数据库集群快照在同一 AWS 账户中。
- 父数据库实例必须与数据库集群快照在同一 AWS 区域中。

如果数据库集群快照与父数据库实例不在同一区域中，或者如果删除了父数据库实例，则这些快照将被视为孤立快照。孤立的数据库快照不支持成本分配标签。孤立快照的成本会汇总到单个未标记的行项目中。满足以下条件时，跨账户数据库集群快照将不会被视为孤立快照：

- 这些快照与父数据库实例在同一区域中。
- 父数据库实例由源账户所有。

> **Note**
>
> 如果父数据库实例由其他账户所有，则成本分配标签不适用于目标账户中的跨账户快照。

标记 Amazon RDS 资源的最佳实践

在使用标签时，建议您遵循以下最佳实践：

- 记录组织中所有团队所遵循的标签使用惯例。具体而言，请确保名称既具有描述性又保持一致。例如，对格式 `environment:prod` 进行标准化，而不是使用 `env:production` 标记某些资源。

> **Important**
>
> 请勿在标签中存储个人身份信息（PII）或其他机密或敏感信息。

- 自动添加标记以确保一致性。例如，您可以使用以下方法：
  - 在 AWS CloudFormation 模板中包含标签。在使用模板创建资源时，系统会自动标记资源。
  - 使用 AWS Lambda 函数定义和应用标签。
  - 创建一个 SSM 文档，其中包含向 RDS 资源添加标签的步骤。
- 仅在必要时使用标签。您最多可以为单个 RDS 资源添加 50 个标签，但最佳实践是避免不必要的标签激增和复杂性。
- 定期检查标签的相关性和准确性。根据需要删除或修改已过时的标签。
- 考虑在 AWS Management Console中使用 AWS 标签编辑器创建标签。您可以使用标签编辑器同时向多个受支持的 AWS 资源（包括 RDS 资源）添加标签。有关更多信息，请参阅 AWS Resource Groups 用户指南中的标签编辑器。

复制标签到数据库集群快照

在创建或还原数据库集群时，您可以指定将集群中的标签复制到数据库集群的快照。复制标签可确保数据库快照的元数据与源数据库集群的元数据匹配。它还确保数据库快照的任何访问策略也与源数据库集群的访问策略相匹配。默认情况下不复制标签。

您可以为以下操作指定将标签复制到数据库快照：

- 创建数据库集群
- 还原数据库集群。
- 创建只读副本
- 复制数据库集群快照

> **Note**
>
> 在某些情况下，您可以为 AWS CLI 命令 create-db-snapshot 的 `--tags` 参数包含一个值。或者，您可以为 CreateDBSnapshot API 操作提供至少一个标签。在这些情况下，RDS 不会将标签从源数据库实例复制到新的数据库快照。即使源数据库实例已开启 `--copy-tags-to-snapshot`（CopyTagsToSnapshot）选项，此功能也适用。
> 如果您采用此方法，则可以从数据库快照创建数据库实例的副本。这种方法可避免添加不适用于新数据库实例的标签。您可以使用 AWS CLI `create-db-snapshot` 命令（或 `CreateDBSnapshot` RDS API 操作）创建数据库快照。创建数据库快照后，可以按本主题后面所述添加标签。

在 Amazon RDS 中添加和删除标签

您可执行以下操作：

- 在创建资源的同时创建标签，例如在运行 AWS CLI 命令 `create-db-instance` 时。
- 使用命令 `add-tags-to-resource` 向现有资源添加标签。
- 使用命令 `list-tags-for-resource` 列出与特定资源关联的标签。
- 使用命令 `add-tags-to-resource` 更新标签。
- 使用命令 `remove-tags-from-resource` 从资源中删除标签。

以下步骤说明如何对与数据库实例和 Aurora 数据库集群相关的资源执行典型的标记操作。注意，对标签进行缓存以用于授权。因此，当您在 Amazon RDS 资源上添加或更新标签时，可能需要等待几分钟，之后修改才能生效。

控制台

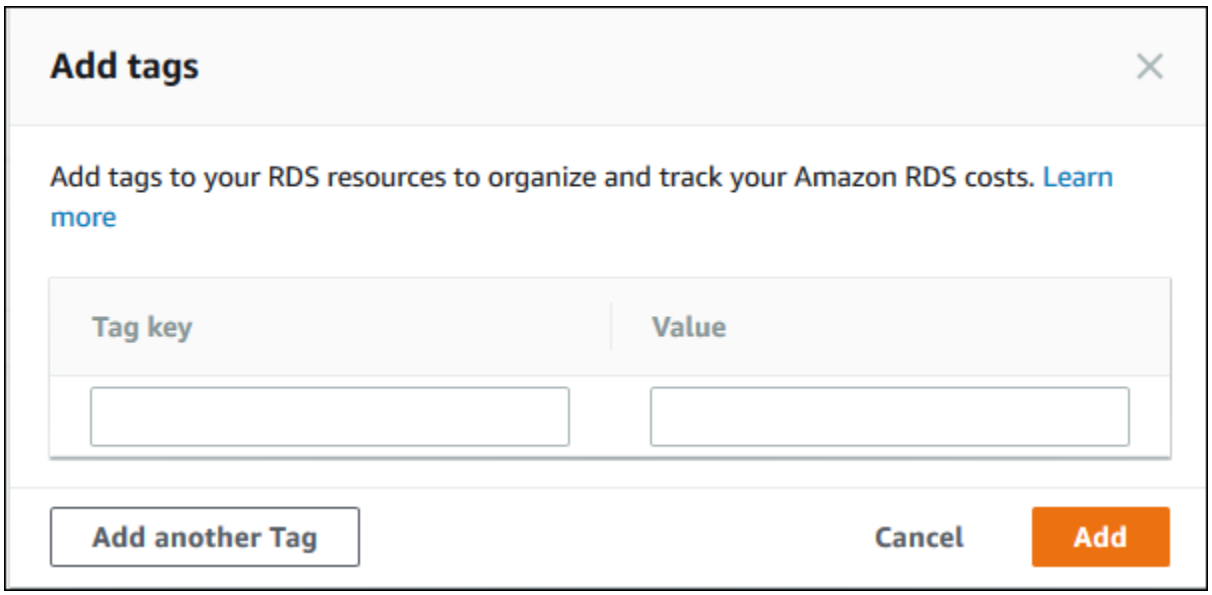
为 Amazon RDS 资源加标签的过程对于所有资源均类似。以下过程展示如何为 Amazon RDS 数据库实例加标签。

要向数据库实例添加标签，请执行以下操作：

1. 登录AWS Management Console并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Databases (数据库)。

   > **Note**
   >
   > 要在数据库窗格中筛选数据库实例的列表，对于筛选数据库，输入文本字符串。只会显示包含该字符串的数据库实例。

3. 选择要标记的数据库实例的名称以显示其详细信息。
4. 在详细信息部分中，向下滚动到标签部分。
5. 选择 Add。将显示添加标签窗口。



6. 为标签键和值输入一个值。
7. 要添加其他标签，您可以选择添加其他标签，并为其标签键和值输入一个值。

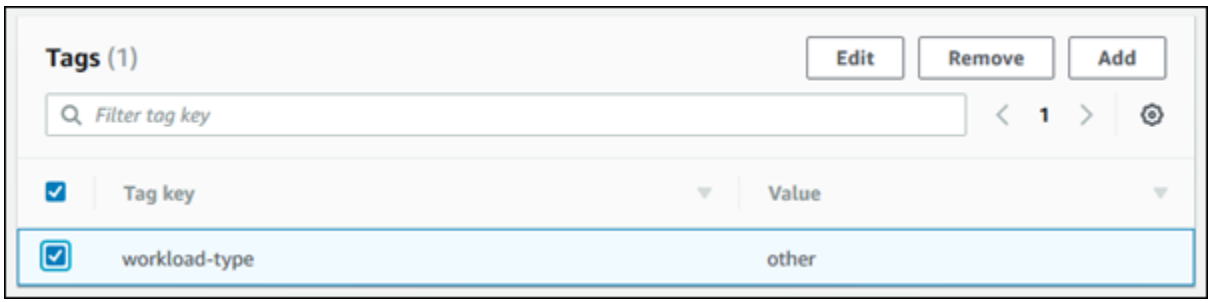
   将该步骤重复执行所需的次数。
8. 选择 Add。

删除数据库实例的标签

1. 登录AWS Management Console并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Databases (数据库)。

> **Note**
>
> 要在数据库窗格中筛选数据库实例的列表，请在筛选数据库框输入文本字符串。只会显示包含该字符串的数据库实例。

3. 选择数据库实例的名称以显示其详细信息。
4. 在详细信息部分中，向下滚动到标签部分。
5. 选择要删除的标签。



6. 选择 Delete (删除)，然后在 Delete tags (删除标签) 窗口中选择 Delete (删除)。

AWS CLI

可以使用 AWS CLI 为数据库实例添加、列出或删除标签。

- 要将一个或多个标签添加到 Amazon RDS 资源，请使用 AWS CLI 命令 `add-tags-to-resource`。
- 要列出 Amazon RDS 资源上的标签，请使用 AWS CLI 命令 `list-tags-for-resource`。
- 要从 Amazon RDS 资源中删除一个或多个标签，请使用 AWS CLI 命令 `remove-tags-from-resource`。

要了解有关如何构建所需 ARN 的更多信息，请参阅构建 Amazon RDS 的 ARN。

RDS API

您可使用 Amazon RDS API 为数据库实例添加、列出或删除标签。

- 要向 Amazon RDS 资源添加标签，请使用 `AddTagsToResource` 操作。
- 要列出分配给 Amazon RDS 资源的标签，请使用 `ListTagsForResource`。
- 要从 Amazon RDS 资源删除标签，请使用 `RemoveTagsFromResource` 操作。

要了解有关如何构建所需 ARN 的更多信息，请参阅构建 Amazon RDS 的 ARN。

在通过 Amazon RDS API 使用 XML 时，标签会使用如下架构：

```
<Tagging>
      <TagSet>
          <Tag>
              <Key>Project</Key>
              <Value>Trinity</Value>
```

```
        </Tag>
        <Tag>
            <Key>User</Key>
            <Value>Jones</Value>
        </Tag>
    </TagSet>
</Tagging>
```

下表提供了允许使用的 XML 标签及其特征的列表。Key 和 Value 的值区分大小写。例如，project=Trinity 和 PROJECT=Trinity 是不同的标签。

标签元素	描述
标签集	标签集是分配给 Amazon RDS 资源的所有标签的容器。每个资源只能有一个标签集。您只可以通过 Amazon RDS API 使用标签集。
标签	标签是用户定义的键值对。一个标签集中可以有 1 到 50 个标签。
密钥	密钥是标签必需的名称。有关限制，请参阅 Amazon RDS 中的标签结构。 该字符串值的长度可以在 1 到 128 个 Unicode 字符之间，并且不能带有前缀 aws: 或 rds:。该字符串只能包含 Unicode 字母、数字、空格、“_”、“.”、“/”、“=”、“+”、“-”的集合（Java 正则表达式：“^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-]*)$”）。 密钥在标签集中必须具有唯一性。例如，标签集中不能有键相同但值不同的键-值对，如 project/Trinity 和 project/Xanadu。
值	值是标签的可选内容。有关限制，请参阅 Amazon RDS 中的标签结构。 该字符串值的长度可以在 1 到 256 个 Unicode 字符之间，并且不能带有前缀 aws: 或 rds:。该字符串只能包含 Unicode 字母、数字、空格、“_”、“.”、“/”、“=”、“+”、“-”的集合（Java 正则表达式：“^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-]*)$”）。 在标签集中，值不必具有唯一性，且可为空值。例如，在“项目/Trinity”和“成本 - 中心/Trinity”的一个标签集中，可以存在键值对。

教程：使用标签指定要停止哪些 Aurora 数据库集群

假设您正在开发或测试环境中创建大量 Aurora 数据库集群。您需要将所有这些集群保留几天。一些集群会在夜间运行测试。另一些集群可以在夜间停止运行并在第二天重新开始运行。以下示例说明如何为适合在夜间停止运行的集群分配标签。之后，该示例显示脚本如何检测哪些集群具有该标签并随后停止这些集群。在此示例中，键值对的值部分无关紧要。存在`stoppable`标签表示集群具有此用户定义的属性。

指定要停止的 Aurora 数据库集群

1. 确定您要指定为“可停止”的集群的 ARN。

   用于标记的命令和 API 使用 ARN。这样一来，它们就可以在 AWS 区域、AWS 账户和可能具有相同短名称的不同类型资源之间无缝运作。您可以在集群上运行的 CLI 命令中指定 ARN 代替集群 ID。将自己集群的名称替换为 *dev-test-cluster*。在使用 ARN 参数的后续命令中，替换自己集群的 ARN。ARN 包含您自己的 AWS 账户 ID 和集群所在的 AWS 区域的名称。

   ```
   $ aws rds describe-db-clusters --db-cluster-identifier dev-test-cluster \
     --query "*[].{DBClusterArn:DBClusterArn}" --output text
   arn:aws:rds:us-east-1:123456789:cluster:dev-test-cluster
   ```

2. 将标签 `stoppable` 添加至此集群。

   由您选择此标签的名称。这种方法意味着，您可以避免设计一种在名称中对所有相关信息进行编码的命名约定。在此类约定中，您可以在数据库实例名称或其他资源的名称中对信息进行编码。由于此示例将标签视为存在或不存在的属性，因此它忽略了 `Value=` 参数的 `--tags` 部分。

   ```
   $ aws rds add-tags-to-resource \
     --resource-name arn:aws:rds:us-east-1:123456789:cluster:dev-test-cluster \
     --tags Key=stoppable
   ```

3. 确认集群中存在该标签。

   以下命令以 JSON 格式和以制表符分隔的纯文本检索集群的标签信息。

   ```
   $ aws rds list-tags-for-resource \
     --resource-name arn:aws:rds:us-east-1:123456789:cluster:dev-test-cluster
   {
       "TagList": [
           {
               "Key": "stoppable",
   ```

```
            "Value": ""

        }
    ]
}
$ aws rds list-tags-for-resource \
  --resource-name arn:aws:rds:us-east-1:123456789:cluster:dev-test-cluster --output
 text
TAGLIST stoppable
```

4. 要停止所有指定为stoppable的集群，请准备所有集群的列表。循环浏览列表，并检查每个集群是否都具有相关属性标签。

   此 Linux 示例使用 shell 脚本将群集 ARN 列表保存到临时文件中，然后为每个集群执行 CLI 命令。

```
$ aws rds describe-db-clusters --query "*[].[DBClusterArn]" --output text >/tmp/
cluster_arns.lst
$ for arn in $(cat /tmp/cluster_arns.lst)
do
  match="$(aws rds list-tags-for-resource --resource-name $arn --output text | grep
 'TAGLIST\tstoppable')"
  if [[ ! -z "$match" ]]
  then
      echo "Cluster $arn is tagged as stoppable. Stopping it now."
# Note that you can specify the full ARN value as the parameter instead of the
 short ID 'dev-test-cluster'.
      aws rds stop-db-cluster --db-cluster-identifier $arn
  fi
done

Cluster arn:aws:rds:us-east-1:123456789:cluster:dev-test-cluster is tagged as
 stoppable. Stopping it now.
{
    "DBCluster": {
        "AllocatedStorage": 1,
        "AvailabilityZones": [
            "us-east-1e",
            "us-east-1c",
            "us-east-1d"
        ],
        "BackupRetentionPeriod": 1,
        "DBClusterIdentifier": "dev-test-cluster",
```

```
        ...
```

您可以在每天结束时运行此类脚本，以确保停止非必要的集群。还可以使用实用程序（例如 cron）安排任务，如每晚执行此类检查。例如，您可以这样做，以防某些集群被误运行。在此，您可以对准备待检查集群列表的命令进行微调。

以下命令生成集群列表，但只生成处于available状态的集群的列表。该脚本可以忽略已停止的集群，因为它们将具有不同的状态值，如stopped或stopping。

```
$ aws rds describe-db-clusters \
  --query '*[].{DBClusterArn:DBClusterArn,Status:Status}|[?Status == `available`]|[].
{DBClusterArn:DBClusterArn}' \
  --output text
arn:aws:rds:us-east-1:123456789:cluster:cluster-2447
arn:aws:rds:us-east-1:123456789:cluster:cluster-3395
arn:aws:rds:us-east-1:123456789:cluster:dev-test-cluster
arn:aws:rds:us-east-1:123456789:cluster:pg2-cluster
```

Tip

您可以将分配标签的过程与查找具有这些标签的集群的过程结合使用，以其他方式降低成本。我们以用于开发和测试的 Aurora 数据库集群为例。在此，您可以指定在每天结束时删除一些集群，或者仅删除其读取器数据库实例。或者，您可以指定某些集群在预期使用率较低的时期，将其数据库实例更改为小型数据库实例类。

Amazon RDS 中的 Amazon 资源名称（ARN）

Amazon Web Services 中创建的资源分别使用 Amazon Resource Name (ARN) 进行唯一标识。对于某些 Amazon RDS 操作，您必须通过指定 Amazon RDS 资源的 ARN 来唯一标识该资源。例如，当您创建 RDS 数据库实例只读副本时，必须提供源数据库实例的 ARN。

有关构造 ARN 和获取现有 ARN 的信息，请参阅以下主题。

主题

- 构建 Amazon RDS 的 ARN
- 获取 Amazon RDS 的现有 ARN

构建 Amazon RDS 的 ARN

Amazon Web Services 中创建的资源分别使用 Amazon Resource Name (ARN) 进行唯一标识。您可以使用以下语法为 Amazon RDS 资源构建 ARN。

`arn:aws:rds:<region>:<account number>:<resourcetype>:<name>`

区域名称	区域	端点	协议
美国东部（俄亥俄州）	us-east-2	rds.us-east-2.amazonaws.com	HTTPS
		rds-fips.us-east-2.api.aws	HTTPS
		rds.us-east-2.api.aws	HTTPS
		rds-fips.us-east-2.amazonaws.com	HTTPS
美国东部（弗吉尼亚州北部）	us-east-1	rds.us-east-1.amazonaws.com	HTTPS
		rds-fips.us-east-1.api.aws	HTTPS
		rds-fips.us-east-1.amazonaws.com	HTTPS
		rds.us-east-1.api.aws	HTTPS
美国西部（加利福	us-west-1	rds.us-west-1.amazonaws.com	HTTPS
		rds.us-west-1.api.aws	HTTPS

区域名称	区域	端点	协议
尼亚北部）		rds-fips.us-west-1.amazonaws.com	HTTPS
		rds-fips.us-west-1.api.aws	HTTPS
美国西部（俄勒冈州）	us-west-2	rds.us-west-2.amazonaws.com	HTTPS
		rds-fips.us-west-2.amazonaws.com	HTTPS
		rds.us-west-2.api.aws	HTTPS
		rds-fips.us-west-2.api.aws	HTTPS
非洲（开普敦）	af-south-1	rds.af-south-1.amazonaws.com	HTTPS
		rds.af-south-1.api.aws	HTTPS
亚太地区（香港）	ap-east-1	rds.ap-east-1.amazonaws.com	HTTPS
		rds.ap-east-1.api.aws	HTTPS
亚太地区（海得拉巴）	ap-south-2	rds.ap-south-2.amazonaws.com	HTTPS
		rds.ap-south-2.api.aws	HTTPS
亚太地区（雅加达）	ap-southeast-3	rds.ap-southeast-3.amazonaws.com	HTTPS
		rds.ap-southeast-3.api.aws	HTTPS
亚太地区（马来西亚）	ap-southeast-5	rds.ap-southeast-5.amazonaws.com	HTTPS
亚太地区（墨尔本）	ap-southeast-4	rds.ap-southeast-4.amazonaws.com	HTTPS
		rds.ap-southeast-4.api.aws	HTTPS
亚太地区（孟买）	ap-south-1	rds.ap-south-1.amazonaws.com	HTTPS
		rds.ap-south-1.api.aws	HTTPS

区域名称	区域	端点	协议
亚太地区（大阪）	ap-northeast-3	rds.ap-northeast-3.amazonaws.com	HTTPS
		rds.ap-northeast-3.api.aws	HTTPS
亚太地区（首尔）	ap-northeast-2	rds.ap-northeast-2.amazonaws.com	HTTPS
		rds.ap-northeast-2.api.aws	HTTPS
亚太地区（新加坡）	ap-southeast-1	rds.ap-southeast-1.amazonaws.com	HTTPS
		rds.ap-southeast-1.api.aws	HTTPS
亚太地区（悉尼）	ap-southeast-2	rds.ap-southeast-2.amazonaws.com	HTTPS
		rds.ap-southeast-2.api.aws	HTTPS
亚太地区（台北）	ap-east-2	rds.ap-east-2.amazonaws.com	HTTPS
亚太地区（泰国）	ap-southeast-7	rds.ap-southeast-7.amazonaws.com	HTTPS
亚太地区（东京）	ap-northeast-1	rds.ap-northeast-1.amazonaws.com	HTTPS
		rds.ap-northeast-1.api.aws	HTTPS
加拿大（中部）	ca-central-1	rds.ca-central-1.amazonaws.com	HTTPS
		rds.ca-central-1.api.aws	HTTPS
		rds-fips.ca-central-1.api.aws	HTTPS
		rds-fips.ca-central-1.amazonaws.com	HTTPS
加拿大西部（卡尔加里）	ca-west-1	rds.ca-west-1.amazonaws.com	HTTPS
		rds-fips.ca-west-1.amazonaws.com	HTTPS

区域名称	区域	端点	协议
欧洲地区（法兰克福）	eu-central-1	rds.eu-central-1.amazonaws.com rds.eu-central-1.api.aws	HTTPS HTTPS
欧洲地区（爱尔兰）	eu-west-1	rds.eu-west-1.amazonaws.com rds.eu-west-1.api.aws	HTTPS HTTPS
欧洲地区（伦敦）	eu-west-2	rds.eu-west-2.amazonaws.com rds.eu-west-2.api.aws	HTTPS HTTPS
欧洲地区（米兰）	eu-south-1	rds.eu-south-1.amazonaws.com rds.eu-south-1.api.aws	HTTPS HTTPS
欧洲地区（巴黎）	eu-west-3	rds.eu-west-3.amazonaws.com rds.eu-west-3.api.aws	HTTPS HTTPS
欧洲（西班牙）	eu-south-2	rds.eu-south-2.amazonaws.com rds.eu-south-2.api.aws	HTTPS HTTPS
欧洲地区（斯德哥尔摩）	eu-north-1	rds.eu-north-1.amazonaws.com rds.eu-north-1.api.aws	HTTPS HTTPS
欧洲（苏黎世）	eu-central-2	rds.eu-central-2.amazonaws.com rds.eu-central-2.api.aws	HTTPS HTTPS
以色列（特拉维夫）	il-central-1	rds.il-central-1.amazonaws.com rds.il-central-1.api.aws	HTTPS HTTPS
墨西哥（中部）	mx-central-1	rds.mx-central-1.amazonaws.com	HTTPS

区域名称	区域	端点	协议
中东（巴林）	me-south-1	rds.me-south-1.amazonaws.com rds.me-south-1.api.aws	HTTPS HTTPS
中东（阿联酋）	me-central-1	rds.me-central-1.amazonaws.com rds.me-central-1.api.aws	HTTPS HTTPS
南美洲（圣保罗）	sa-east-1	rds.sa-east-1.amazonaws.com rds.sa-east-1.api.aws	HTTPS HTTPS
AWS GovCloud（美国东部）	us-gov-east-1	rds.us-gov-east-1.amazonaws.com rds.us-gov-east-1.api.aws	HTTPS HTTPS
AWS GovCloud（美国西部）	us-gov-west-1	rds.us-gov-west-1.amazonaws.com rds.us-gov-west-1.api.aws	HTTPS HTTPS

下表显示在构建特定 Amazon RDS 资源类型的 ARN 时应使用的格式。

资源类型	ARN 格式
数据库实例	arn:aws:rds:*<region>*:*<account>* :db:*<name>* 例如： `arn:aws:rds: us-east-2 :123456789012 :db:my-mysql-instance-1`
数据库群集	arn:aws:rds:*<region>*:*<account>* :cluster:*<name>* 例如：

资源类型	ARN 格式
	`arn:aws:rds:` *us-east-2* `:`*123456789012* `:cluster:` *my-aurora-cluster-1*
事件订阅	arn:aws:rds:*<region>*:*<account>* :es:*<name>* 例如： `arn:aws:rds:` *us-east-2* `:`*123456789012* `:es:`*my-subscription*
数据库参数组	arn:aws:rds:*<region>*:*<account>* :pg:*<name>* 例如： `arn:aws:rds:` *us-east-2* `:`*123456789012* `:pg:`*my-param-enable-logs*
数据库集群参数组	arn:aws:rds:*<region>*:*<account>* :cluster-pg:*<name>* 例如： `arn:aws:rds:` *us-east-2* `:`*123456789012* `:cluster-pg:` *my-cluster-param-timezone*
预留数据库实例	arn:aws:rds:*<region>*:*<account>* :ri:*<name>* 例如： `arn:aws:rds:` *us-east-2* `:`*123456789012* `:ri:`*my-reserved-postgresql*
数据库安全组	arn:aws:rds:*<region>*:*<account>* :secgrp:*<name>* 例如： `arn:aws:rds:` *us-east-2* `:`*123456789012* `:secgrp:`*my-public*

资源类型	ARN 格式
自动数据库快照	arn:aws:rds:*<region>*:*<account>* :snapshot:rds:*<name>* 例如： `arn:aws:rds: us-east-2 :123456789012 :snapshot:rds: my-mysql-db-2019-07-22-07-23`
自动数据库集群快照	arn:aws:rds:*<region>*:*<account>* :cluster-snapshot:rds:*<name>* 例如： `arn:aws:rds: us-east-2 :123456789012 :cluster-snapshot:rds: my-aurora-cluster-2019-07-22-16-16`
手动数据库快照	arn:aws:rds:*<region>*:*<account>* :snapshot:*<name>* 例如： `arn:aws:rds: us-east-2 :123456789012 :snapshot: my-mysql-db-snap`
手动数据库集群快照	arn:aws:rds:*<region>*:*<account>* :cluster-snapshot:*<name>* 例如： `arn:aws:rds: us-east-2 :123456789012 :cluster-snapshot: my-aurora-cluster-snap`
数据库子网组	arn:aws:rds:*<region>*:*<account>* :subgrp:*<name>* 例如： `arn:aws:rds: us-east-2 :123456789012 :subgrp:my-subnet-10`

获取 Amazon RDS 的现有 ARN

您可以使用 AWS Management Console、AWS Command Line Interface (AWS CLI) 或 RDS API 来获取 RDS 资源的 ARN。

控制台

要从 AWS Management Console中获取一个 ARN，请导航到要获取 ARN 的资源，然后查看该资源的详细信息。

例如，您可以从数据库集群详细信息的配置选项卡获取数据库集群的 ARN。

AWS CLI

要从 AWS CLI 为特定 RDS 资源获取 ARN，请对该资源使用 `describe` 命令。下表显示了每个 AWS CLI 命令，以及与命令配合使用以获取 ARN 的 ARN 属性。

AWS CLI command	ARN 属性
describe-event-subscriptions	EventSubscriptionArn
describe-certificates	CertificateArn
describe-db-parameter-groups	DBParameterGroupArn
describe-db-cluster-parameter-groups	DBClusterParameterGroupArn
describe-db-instances	DBInstanceArn
describe-db-security-groups	DBSecurityGroupArn
describe-db-snapshots	DBSnapshotArn
describe-events	SourceArn
describe-reserved-db-instances	ReservedDBInstanceArn
describe-db-subnet-groups	DBSubnetGroupArn
describe-db-clusters	DBClusterArn

AWS CLI command	ARN 属性
describe-db-cluster-snapshots	DBClusterSnapshotArn

例如，以下 AWS CLI 命令获取数据库实例的 ARN。

Example

对于 Linux、macOS 或 Unix：

```
aws rds describe-db-instances \
--db-instance-identifier DBInstanceIdentifier \
--region us-west-2 \
--query "*[].{DBInstanceIdentifier:DBInstanceIdentifier,DBInstanceArn:DBInstanceArn}"
```

对于 Windows：

```
aws rds describe-db-instances ^
--db-instance-identifier DBInstanceIdentifier ^
--region us-west-2 ^
--query "*[].{DBInstanceIdentifier:DBInstanceIdentifier,DBInstanceArn:DBInstanceArn}"
```

该命令的输出如下所示：

```
[
    {
        "DBInstanceArn": "arn:aws:rds:us-west-2:account_id:db:instance_id",
        "DBInstanceIdentifier": "instance_id"
    }
]
```

RDS API

要为特定 RDS 资源获取 ARN，您可以调用以下 RDS API 操作并使用如下所示的 ARN 属性。

RDS API 操作	ARN 属性
DescribeEventSubscriptions	EventSubscriptionArn

RDS API 操作	ARN 属性
DescribeCertificates	CertificateArn
DescribeDBParameterGroups	DBParameterGroupArn
DescribeDBClusterParameterGroups	DBClusterParameterGroupArn
DescribeDBInstances	DBInstanceArn
DescribeDBSecurityGroups	DBSecurityGroupArn
DescribeDBSnapshots	DBSnapshotArn
DescribeEvents	SourceArn
DescribeReservedDBInstances	ReservedDBInstanceArn
DescribeDBSubnetGroups	DBSubnetGroupArn
DescribeDBClusters	DBClusterArn
DescribeDBClusterSnapshots	DBClusterSnapshotArn

Amazon Aurora 更新

Amazon Aurora 定期发布更新。更新将在系统维护时段内应用于 Amazon Aurora 数据库集群。应用更新的时间取决于数据库集群的区域和维护时段设置以及更新的类型。更新要求重新启动数据库，因此，通常会有 20–30 秒的停机时间。此停机时间过后，您可以继续使用数据库集群。您可以从 AWS Management Console查看或更改维护时段设置。

> **Note**
>
> 重启数据库实例所需的时间取决于崩溃恢复过程、重启时的数据库活动以及特定数据库引擎的行为。为优化重新启动的时间，建议在重启过程中尽可能减少数据库活动。减少数据库活动可减少中转事务的回滚活动。

有关 Amazon Aurora 的操作系统更新的信息，请参阅 ???。

某些更新特定于 Aurora 支持的数据库引擎。有关数据库引擎更新的更多信息，请参阅下表。

数据库引擎	更新
Amazon Aurora MySQL	请参阅 Amazon Aurora MySQL 的数据库引擎更新
Amazon Aurora PostgreSQL	请参阅 Amazon Aurora PostgreSQL 的数据库引擎更新

确定您的 Amazon Aurora 版本

Amazon Aurora 包含某些 Aurora 通用功能，这些功能适用于所有 Aurora 数据库集群。Aurora 包含其自身支持的某个数据库引擎的其他特定功能。这些功能仅适用于使用该数据库引擎的 Aurora 数据库集群，例如 Aurora PostgreSQL。

Aurora 数据库实例提供两个版本号：Aurora 版本号和数据库引擎版本号。Aurora 版本号使用以下格式。

```
<major version>.<minor version>.<patch version>
```

要从使用特定数据库引擎的 Aurora 数据库实例中获取 Aurora 版本号，请使用以下查询之一。

数据库引擎	查询
Amazon Aurora MySQL	`SELECT AURORA_VERSION();` `SHOW @@aurora_version;`
Amazon Aurora PostgreSQL	`SELECT AURORA_VERSION();`

Amazon Aurora 的 Amazon RDS 扩展支持

借助 RDS 扩展支持，您可以在 Aurora 标准支持终止日期后，继续在主要引擎版本上运行数据库，但需要额外付费。

只能通过在首次创建或还原数据库实例时启用 RDS 扩展支持，将数据库注册到 RDS 扩展支持。除非您要还原现有数据库实例，否则您无法更新这些数据库实例上的 RDS 扩展支持注册状态。

如果您在创建或还原数据库实例期间启用了 RDS 扩展支持，则在 Aurora 标准支持终止日期之后，Amazon Aurora 会自动将数据库实例注册到 RDS 扩展支持。自动注册到 RDS 扩展支持不会更改数据库引擎，也不会影响数据库实例的正常运行时间或性能。

RDS 扩展支持提供以下更新和技术支持：

- 数据库实例或数据库集群（包括数据库引擎）的 critical and high CVEs 的安全更新
- 针对关键问题的错误修复和补丁
- 能够在标准 Amazon RDS 服务等级协议范围内建立支持案例并获得故障排除帮助

这项付费服务让您有更多时间升级到支持的主要引擎版本。例如，Aurora MySQL 版本 2 的 Aurora 标准支持终止日期为 2024 年 10 月 31 日。但是，在该日期之前，您还没有准备好手动升级到 Aurora MySQL 版本 3。在这一情况下，Amazon Aurora 会在 2024 年 10 月 31 日自动将您的集群注册到 RDS 扩展支持，并且您可以继续运行 Aurora MySQL 版本 2。从 2024 年 12 月 1 日起，Amazon Aurora 将自动向您收取 RDS 扩展支持费用。

对于主要引擎版本，在社区生命周期终止日期之后，RDS 扩展支持可提供长达 3 年的服务（对于 Aurora MySQL 版本 2，为 3 年零 4 个月）。此时间之后，如果您尚未将主要引擎版本升级到支持的版本，Amazon Aurora 将自动升级您的主要引擎版本。建议您尽快升级到支持的主要引擎版本。

有关 Aurora 标准支持终止日期和 RDS 扩展支持终止日期的更多信息，请参阅 Aurora MySQL 主要版本的发布日历和 Aurora PostgreSQL 主要版本的发布日历。

主题

- Amazon RDS 扩展支持概述
- Amazon RDS 扩展支持费用
- 提供 Amazon RDS Extended Support 版本
- Amazon Aurora 和客户对于 Amazon RDS 扩展支持的责任
- 使用 Amazon RDS 扩展支持创建Aurora 数据库集群或全局集群

- 查看Aurora 数据库集群或全局集群在 Amazon RDS 扩展支持中的注册情况
- 在 Amazon RDS 扩展支持中查看引擎版本的支持日期
- 使用 Amazon RDS 扩展支持还原Aurora 数据集群或全局集群

Amazon RDS 扩展支持概述

在 Aurora 标准支持终止日期之后，如果您在创建或还原数据库实例期间没有禁用 RDS 扩展支持，则 Amazon Aurora 会自动将它们注册到 RDS 扩展支持。如果您尚未运行在 Aurora 标准支持终止日期之前发布的最后一个次要版本，Aurora 会自动将您的数据库实例升级到该次要版本。直至主要引擎版本的 Aurora 标准支持终止日期之后，Amazon Aurora 才会升级您的次要版本。

您可以使用已达到 Aurora 标准支持终止日期的主要引擎版本创建新的数据库。Aurora 会自动在 RDS 扩展支持中注册这些新数据库，并向您收取此服务的费用。

如果您在 Aurora 标准支持终止日期之前 升级到仍处于 Aurora 标准支持之下的引擎，Amazon Aurora 不会将您的引擎注册到 RDS 扩展支持。

如果您试图还原与超过 Aurora 标准支持终止日期但未注册到 RDS 扩展支持的引擎兼容的数据库快照，则 Amazon Aurora 将尝试升级快照，使其与仍处于 Aurora 标准支持之下的最新引擎版本兼容。如果还原失败，则 Amazon Aurora 将使用与快照兼容的版本，自动将您的引擎注册到 RDS 扩展支持。

您可以随时终止 RDS 扩展支持的注册。要终止注册，请将每个已注册的引擎升级到仍处于 Aurora 标准支持之下的更高引擎版本。RDS 扩展支持注册的终止将在您完成升级到仍处于 Aurora 标准支持之下的更高引擎版本的当天生效。

有关 Aurora 标准支持终止日期和 RDS 扩展支持终止日期的更多信息，请参阅 Aurora MySQL 主要版本的发布日历和 Aurora PostgreSQL 主要版本的发布日历。

Amazon RDS 扩展支持费用

从 Aurora 标准支持终止日期的第二天开始，您将为在 RDS 扩展支持中注册的所有引擎付费。有关 Aurora 标准支持终止日期，请参阅 Amazon Aurora 主要版本。

当您执行以下操作之一时，RDS 扩展支持的额外费用将自动停止：

- 升级到标准支持涵盖范围内的引擎版本。
- 删除在 Aurora 标准支持终止日期之后运行主要版本的数据库。

如果您的目标引擎版本将来进入 RDS 扩展支持，将重新开始收费。

例如 Aurora PostgreSQL 11 将于 2024 年 3 月 1 日起进入扩展支持，但要等到 2024 年 4 月 1 日才会开始收费。您在 2024 年 4 月 30 日将 Aurora PostgreSQL 11 数据库升级到 Aurora PostgreSQL 12。您将只需在 Aurora PostgreSQL 11 上支付 30 天的扩展支持费用。在 RDS 标准支持终止日期 2025 年 2 月 28 日之后，您继续在此数据库实例上运行 Aurora PostgreSQL 12。从 2025 年 3 月 1 日起，将再次向您的数据库收取 RDS 扩展支持费用。

有关更多信息，请参阅 Amazon Aurora 定价。

避免支付 Amazon RDS 扩展支持费用

您可以通过阻止 Aurora 在 Aurora 标准支持终止日期后创建或还原 Aurora 数据库集群或全局集群，来避免支付 RDS 扩展支持费用。为此，请使用 AWS CLI 或 RDS API。

在 AWS CLI 中，为 `--engine-lifecycle-support` 选项指定 `open-source-rds-extended-support-disabled`。在 RDS API 中，为 `LifeCycleSupport` 参数指定 `open-source-rds-extended-support-disabled`。有关更多信息，请参阅创建Aurora 数据库集群或全局集群或还原Aurora 数据集群或全局集群。

提供 Amazon RDS Extended Support 版本

RDS 扩展支持可用于 Aurora MySQL 和 Aurora PostgreSQL。有关更多信息，请参阅 Amazon Aurora 主要版本。

RDS Extended Support 仅适用于某些次要版本。只有在主要版本达到社区生命周期终止日期之后，次要版本才会被指定为支持 RDS 扩展支持。有关更多信息，请参阅《Aurora MySQL Release Notes》中的 Release calendar for Aurora MySQL 和《Aurora PostgreSQL Release Notes》中的 Release calendar for Aurora PostgreSQL。

RDS Extended Support 仅适用于 Aurora Serverless v2。此功能在 Aurora Serverless v1 上不可用。

您还可以使用 AWS CLI 或 RDS API 查看有关引擎版本支持日期的信息。有关更多信息，请参阅 在 Amazon RDS 扩展支持中查看引擎版本的支持日期。

Amazon Aurora 和客户对于 Amazon RDS 扩展支持的责任

以下内容描述了 Amazon Aurora 对于 RDS 扩展支持的责任以及您对于 RDS 扩展支持的责任。

主题

- Amazon Aurora 的责任
- 您的责任

Amazon Aurora 的责任

在 Aurora 标准支持终止日期之后，Amazon Aurora 将为注册了 RDS 扩展支持的引擎提供补丁、错误修复和升级。这将持续长达 3 年，或者直到您停止使用引擎为止，以先发生者为准。

这些补丁将用于根据美国国家漏洞数据库（NVD）的 CVSS 严重性评级定义的严重和高 CVE。有关更多信息，请参阅漏洞指标。

您的责任

您负责为注册了 RDS 扩展支持的 Aurora 数据库集群或全局集群应用补丁、错误修复和升级。Amazon Aurora 保留随时更改、替换或撤回此类补丁、错误修复和升级的权利。如果需要某个补丁来解决安全或关键稳定性问题，Amazon Aurora 将保留使用该补丁更新您的 Aurora 数据库集群或全局集群或要求您安装该补丁的权利。

您还负责在 RDS 扩展支持终止日期之前，将引擎升级到更高的引擎版本。RDS 的扩展支持终止日期通常是 社区生命周期终止之后 3 年。。有关数据库主要引擎版本的 RDS 扩展支持终止日期，请参阅Amazon Aurora 主要版本。

如果您不升级引擎，则在 RDS 扩展支持终止日期之后，Amazon Aurora 会尝试将您的引擎升级到 Aurora 标准支持所支持的更新引擎版本。如果升级失败，Amazon Aurora 将保留以下权利：删除在 Aurora 标准支持终止日期之后正在运行该引擎的 Aurora 数据库集群或全局集群。但是，在这样做之前，Amazon Aurora 将保留您来自该引擎的数据。

使用 Amazon RDS 扩展支持创建Aurora 数据库集群或全局集群

在创建Aurora 数据库集群或全局集群时，请在控制台中选择启用 RDS 扩展支持，或者使用 AWS CLI 中的扩展支持选项或 RDS API 中的参数。当您在 Amazon RDS 扩展支持中注册 Aurora 数据库集群或全局集群时，它将在 Aurora 数据库集群或全局集群的生命周期内永久注册 RDS 扩展支持。

如果您使用控制台，则必须选择启用 RDS 扩展支持。默认情况下未选择该设置。

如果您使用 AWS CLI 或 RDS API，但未指定 RDS 扩展支持设置，则 Amazon RDS 默认为启用 RDS 扩展支持。当您使用 AWS CloudFormation 或其它服务自动执行操作时，此默认行为会在 Aurora 标准支持终止日期之后保持数据库的可用性。

可以使用 AWS CLI 或 RDS API 来创建 Aurora 数据库集群或全局集群，来防止注册 RDS 扩展支持。

主题

- RDS 扩展支持行为
- 使用 RDS 扩展支持时的注意事项
- 使用 RDS 扩展支持创建Aurora 数据库集群或全局集群

RDS 扩展支持行为

下表汇总了当主引擎版本达到 Aurora 标准支持终止时会发生的情况。

RDS 扩展支持状态*	行为
已启用	Amazon RDS 将向您收取 RDS 扩展支持费用。
已禁用	Amazon RDS 将您的 Aurora 数据库集群或全局集群升级到支持的引擎版本。此升级在 Aurora 标准支持终止日期或之后不久进行。

* 在 RDS 控制台中，RDS 扩展支持状态显示为“是”或“否”。在 AWS CLI 或 RDS API 中，RDS 扩展支持状态显示为 `open-source-rds-extended-support` 或 `open-source-rds-extended-support-disabled`。

使用 RDS 扩展支持时的注意事项

在创建Aurora 数据库集群或全局集群之前，请考虑以下事项：

- 在 Aurora 标准支持终止日期过去之后，您可以阻止创建新的 Aurora 数据库集群或新的全局集群，并避免支付 RDS 扩展支持费用。为此，请使用 AWS CLI 或 RDS API。在 AWS CLI 中，为 `--engine-lifecycle-support` 选项指定 `open-source-rds-extended-support-disabled`。在 RDS API 中，为 `LifeCycleSupport` 参数指定 `open-source-rds-extended-support-disabled`。如果您指定 `open-source-rds-extended-support-disabled` 且 Aurora 标准支持终止日期已过，则创建 Aurora 数据库集群或全局集群始终会失败。
- RDS 扩展支持在集群级别进行设置。集群成员在 RDS 控制台、AWS CLI 中的 `--engine-lifecycle-support`，RDS API 中的 `EngineLifecycleSupport`，始终具有相同的 RDS 扩展支持设置。

有关更多信息，请参阅 Amazon Aurora 版本。

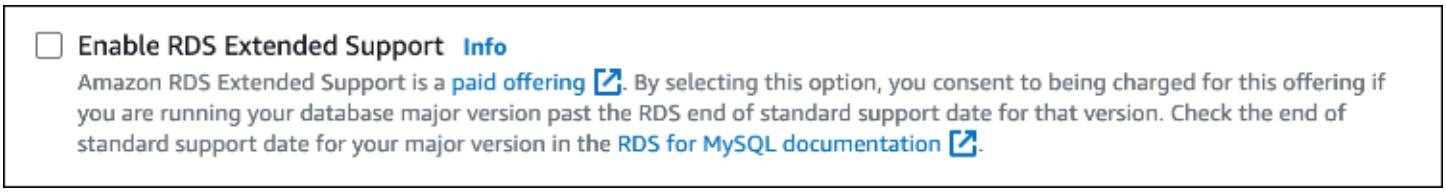
使用 RDS 扩展支持创建Aurora 数据库集群或全局集群

您可以使用 AWS Management Console、AWS CLI 或 RDS API 创建Aurora 数据库集群或全局集群

控制台

在创建 Aurora 数据库集群或全局集群时，请在引擎选项部分中选择启用 RDS 扩展支持。默认情况下未选择该设置。

下图显示了启用 RDS 扩展支持设置：



AWS CLI

在您运行 create-db-cluster 或 create-global-cluster AWS CLI 命令时，请通过为 `--engine-lifecycle-support` 选项指定 `open-source-rds-extended-support` 来选择 RDS 扩展支持。默认情况下，此选项设置为 `open-source-rds-extended-support`。

要防止在 Aurora 标准支持终止日期之后创建新的 Aurora 数据库集群或全局集群，请为 `--engine-lifecycle-support` 选项指定 `open-source-rds-extended-support-disabled`。这样，您就可以避免支付任何关联的 RDS 扩展支持费用。

RDS API

当您使用 CreateDBCluster 或 CreateGlobalCluster Amazon RDS API 操作时，请通过将 `EngineLifecycleSupport` 参数设置为 `open-source-rds-extended-support` 来选择 RDS 扩展支持。默认情况下，此参数设置为 `open-source-rds-extended-support`。

要防止在 Aurora 标准支持终止日期之后创建新的 Aurora 数据库集群或全局集群，请为 `EngineLifecycleSupport` 参数指定 `open-source-rds-extended-support-disabled`。这样，您就可以避免支付任何关联的 RDS 扩展支持费用。

有关更多信息，请参阅以下主题：

- 要创建 Aurora 数据库集群，请按创建 Amazon Aurora 数据库集群中数据库引擎的相关说明操作。
- 要创建全局集群，请按创建 Amazon Aurora Global Database中数据库引擎的相关说明操作。

查看Aurora 数据库集群或全局集群在 Amazon RDS 扩展支持中的注册情况

您可以使用 AWS Management Console、AWS CLI 或 RDS API 查看 Aurora 数据库集群或全局集群在 RDS 扩展支持中的注册情况。

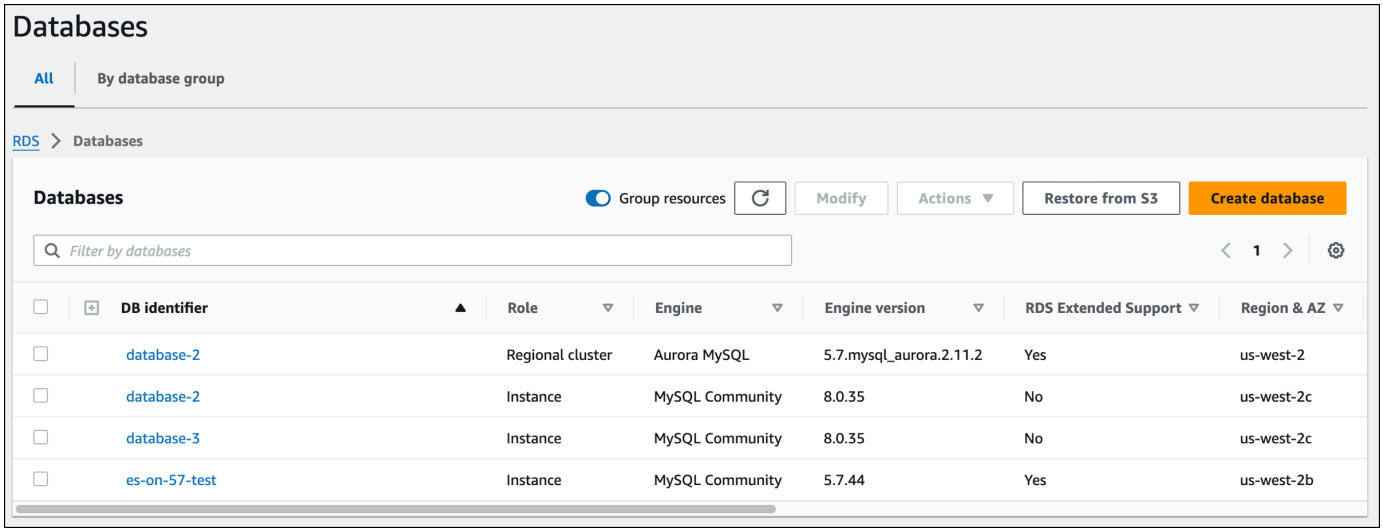
> **Note**
>
> 控制台中的 RDS 扩展支持列、AWS CLI 中的 `--engine-lifecycle-support` 选项和 RDS API 中的 `EngineLifecycleSupport` 参数仅表示在 RDS 扩展支持中进行注册。RDS 扩展支持仅在数据库引擎版本已达到 Aurora 标准支持终止时才开始收费。有关更多信息，请参阅 Amazon Aurora 主要版本。
>
> 例如，您有一个 Aurora PostgreSQL 11 数据库，该数据库注册了 RDS 扩展支持。2024 年 4 月 1 日，Amazon RDS 开始向您收取此数据库的 RDS 扩展支持费用。2024 年 7 月 31 日，您将此数据库升级到了 Aurora PostgreSQL 12。此数据库的 RDS 扩展支持状态保持为已启用。但是，对该数据库的 RDS 扩展支持收费已停止，因为 Aurora PostgreSQL 12 尚未达到 Aurora 标准支持终止。在 2025 年 3 月 1 日之前，Amazon RDS 都不会向您收取该数据库的 RDS 扩展支持费用，该日期是对 Aurora PostgreSQL 12 的 Aurora 标准支持终止的时间。

控制台

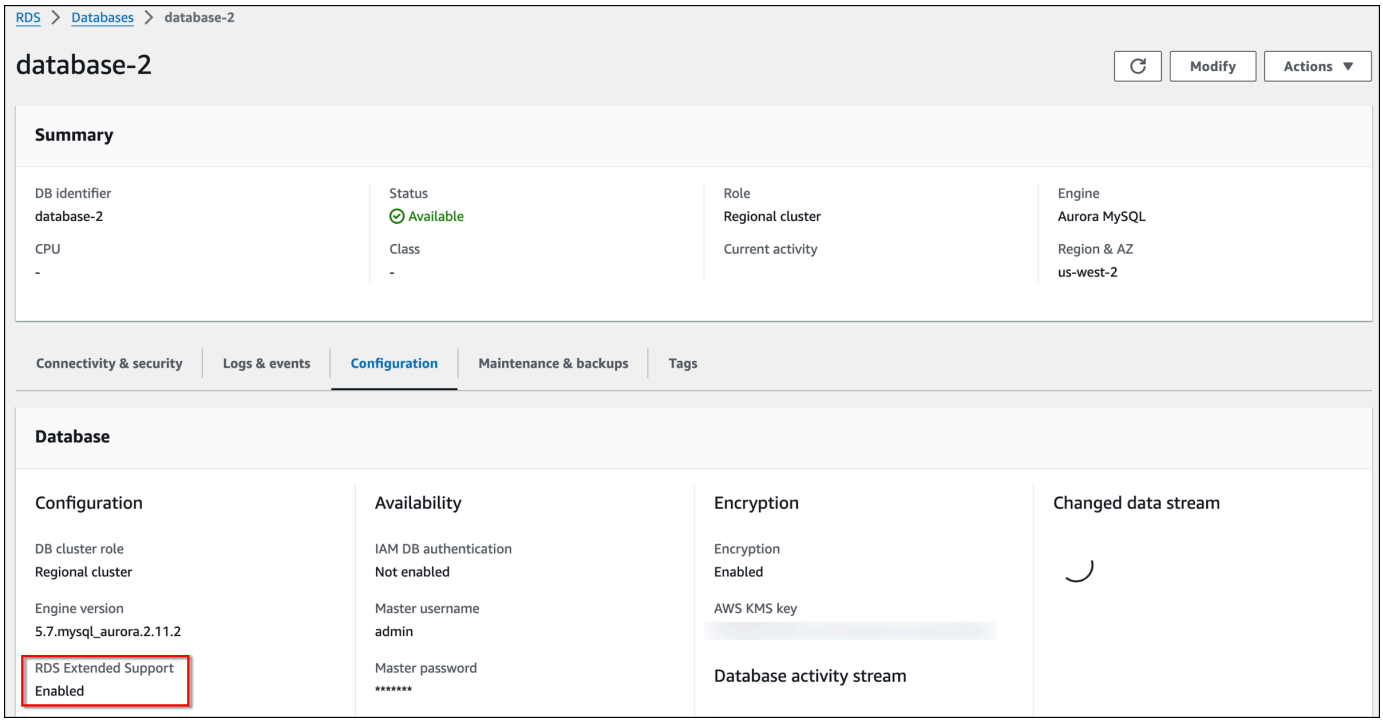
查看Aurora 数据库集群或全局集群在 RDS 扩展支持中的注册情况

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Databases（数据库）。RDS 扩展支持下的值指示Aurora 数据库集群或全局集群是否已在 RDS 扩展支持中注册。如果没有显示任何值，则 RDS 扩展支持不适用于您的数据库。

   > **Tip**
   >
   > 如果未出现 RDS 扩展支持列，请选择首选项图标，然后打开 RDS 扩展支持。



3. 您还可以在配置选项卡上查看每个数据库的注册情况。在数据库标识符下选择一个数据库。在配置选项卡上，在扩展支持下查看数据库是否已注册。



AWS CLI

要使用 AWS CLI 查看您的数据库在 RDS 扩展支持中的注册情况，请运行 describe-db-clusters 或 describe-global-clusters命令。

如果 RDS 扩展支持对数据库可用，则响应中包含参数 `EngineLifecycleSupport`。值 `open-source-rds-extended-support` 指示 Aurora 数据库集群或全局集群已在 RDS 扩展支持中注册。值 `open-source-rds-extended-support-disabled` 指示已禁用在 RDS 扩展支持中注册 Aurora 数据库集群或全局集群。

示例

以下命令会返回所有 Aurora 数据库集群的信息：

```
aws rds describe-db-clusters
```

以下响应显示在 Aurora 数据库集群 `database-1` 上运行的 Aurora PostgreSQL 引擎已注册到 RDS 扩展支持：

```
{
    "DBClusterIdentifier": "database-1",
    ...
    "Engine": "aurora-postgresql",
    ...
    "EngineLifecycleSupport": "open-source-rds-extended-support"
}
```

RDS API

要使用 Amazon RDS API 查看您的数据库在 RDS 扩展支持中的注册情况，请使用 DescribeDBClusters 或 DescribeGlobalClusters 操作。

如果 RDS 扩展支持对数据库可用，则响应中包含参数 `EngineLifecycleSupport`。值 `open-source-rds-extended-support` 指示 Aurora 数据库集群或全局集群已在 RDS 扩展支持中注册。值 `open-source-rds-extended-support-disabled` 指示已禁用在 RDS 扩展支持中注册 Aurora 数据库集群或全局集群。

在 Amazon RDS 扩展支持中查看引擎版本的支持日期

您可以使用 AWS CLI 或 RDS API 在 Amazon RDS 扩展支持中查看有关 Aurora 数据库集群或全局集群的引擎版本的支持日期的信息。利用此信息，您可以更好地规划升级。

AWS CLI 命令和 RDS API 操作返回 Aurora 标准支持和 RDS 扩展支持的开始日期和结束日期。这些日期也可以在主要引擎版本表中找到。有关更多信息，请参阅 Amazon Aurora 主要版本。

AWS CLI

要使用 AWS CLI 查看主要引擎版本的 Aurora 标准支持和 RDS 扩展支持的开始日期和结束日期，请运行 describe-db-major-engine-versions 命令。

此命令将返回以下相关参数：

- `SupportedEngineLifecycles` – 此参数是一个对象数组，其中包括 `LifecycleSupportName`、`LifecycleSupportStartDate` 和 `LifecycleSupportEndDate`。
- `LifecycleSupportName` – 此参数表示引擎版本所处的支持类型：Aurora 标准支持（`open-source-rds-standard-support`）或 RDS 扩展支持（`open-source-rds-extended-support`）。
- `LifecycleSupportStartDate` – 此参数根据 `LifecycleSupportName` 的值，列出主要引擎版本的 Aurora 标准支持或 RDS 扩展支持的开始日期。
- `LifecycleSupportEndDate` – 此参数根据 `LifecycleSupportName` 的值，列出主要引擎版本的 Aurora 标准支持或 RDS 扩展支持的结束日期。

示例

响应示例显示了 Aurora MySQL 版本 2（MySQL 5.7）的支持的引擎生命周期 `open-source-rds-standard-support` 和 `open-source-rds-extended-support` 的开始日期和结束日期。已提供 Aurora MySQL 版本 2（MySQL 5.7）的 RDS 扩展支持。

```
{
    "DBMajorEngineVersions": [
        {
            "Engine": "aurora-mysql",
            "MajorEngineVersion": "5.7",
            "SupportedEngineLifecycles": [
                {
                    "LifecycleSupportName": "open-source-rds-standard-support",
                    "LifecycleSupportStartDate": "2018-02-06T00:00:00+00:00",
                    "LifecycleSupportEndDate": "2024-10-31T23:59:59.999000+00:00"
                },
                {
                    "LifecycleSupportName": "open-source-rds-extended-support",
                    "LifecycleSupportStartDate": "2024-11-01T00:00:00+00:00",
                    "LifecycleSupportEndDate": "2027-02-28T23:59:59.999000+00:00"
```

```
                }
            ]
        },
        ...
    ]
}
```

RDS API

要使用 RDS API 查看主要引擎版本的 Aurora 标准支持和 RDS 扩展支持的开始日期和结束日期，请使用 DescribeDBMajorEngineVersions 操作。

如果已为某个引擎版本提供 RDS 扩展支持，则响应将包含参数 `SupportedEngineLifeCycles` 作为带两个对象的数组。一个对象包含 Aurora 标准支持的开始日期和结束日期。第二个对象包含 RDS 扩展支持的开始日期和结束日期。

如果没有为某个引擎版本提供 RDS 扩展支持，则响应仅包含参数 `SupportedEngineLifeCycles` 作为带单个对象的数组。该对象包含 Aurora 标准支持的开始日期和结束日期。

使用 Amazon RDS 扩展支持还原Aurora 数据集群或全局集群

在还原Aurora 数据集群或全局集群时，请在控制台中选择启用 RDS 扩展支持，或者使用 AWS CLI 中的扩展支持选项或使用 RDS API 中的参数。当您在 RDS 扩展支持中注册 Aurora 数据库集群或全局集群时，它将在 Aurora 数据库集群或全局集群的生命周期内永久注册 RDS 扩展支持。

RDS 扩展支持设置的默认值取决于您是使用控制台、AWS CLI 还是 RDS API 来还原数据库。如果您使用控制台，但未选择启用 RDS 扩展支持，并且您要还原的主引擎版本已超过 Aurora 标准支持终止，则 Amazon Aurora 会自动将您的数据库实例升级到更高的引擎版本。如果您使用 AWS CLI 或 RDS API，但未指定 RDS 扩展支持设置，则 Amazon RDS 默认为启用 RDS 扩展支持。当您使用 AWS CloudFormation 或其他服务自动执行操作时，此默认行为会在 Aurora 标准支持终止日期之后保持数据库的可用性。可以使用 AWS CLI 或 RDS API 来禁用 RDS 扩展支持。

主题

- RDS 扩展支持行为
- 使用 RDS 扩展支持时的注意事项
- 使用 RDS 扩展支持还原 Aurora 数据库集群或全局集群

RDS 扩展支持行为

下表汇总了当要还原的 Aurora 数据库集群或全局集群的主引擎版本已达到 Aurora 标准支持终止时会发生的情况。

RDS 扩展支持状态*	行为
已启用	Amazon RDS 将向您收取 RDS 扩展支持费用。
已禁用	还原完成后，Amazon RDS 会自动将 Aurora 数据库集群或全局集群升级到更高的引擎版本（在将来的维护时段中）。

* 在 RDS 控制台中，RDS 扩展支持状态显示为“是”或“否”。在 AWS CLI 或 RDS API 中，RDS 扩展支持状态显示为 `open-source-rds-extended-support` 或 `open-source-rds-extended-support-disabled`。

使用 RDS 扩展支持时的注意事项

在还原Aurora 数据集群或全局集群之前，请考虑以下事项：

- 在 Aurora 标准支持终止日期过去之后，如果您想从 Amazon S3 还原 Aurora 数据集群或全局集群，您只能使用 AWS CLI 或 RDS API 执行还原操作。使用 restore-db-cluster-from-s3 AWS CLI 命令中的 `--engine-lifecycle-support` 选项，或 RestoreDBClusterFromS3 RDS API 操作中的 `EngineLifecycleSupport` 参数。
- 如果您想阻止 Aurora 将您的数据库还原到 RDS 扩展支持版本，请在 AWS CLI 或 RDS API 中指定 `open-source-rds-extended-support-disabled`。这样，您就可以避免支付任何关联的 RDS 扩展支持费用。

  如果您指定此设置，Amazon Aurora 会自动将您还原的数据库升级到新的、受支持的主要版本。如果升级未通过升级前检查，Amazon Aurora 将安全地回滚到 RDS 扩展支持引擎版本。该数据库将保持 RDS 扩展支持模式，并且在您手动升级数据库之前，Amazon Aurora 将向您收取 RDS 扩展支持费用。
- RDS 扩展支持在集群级别进行设置。集群成员在 RDS 控制台、AWS CLI 中的 `--engine-lifecycle-support`，RDS API 中的 `EngineLifecycleSupport`，始终具有相同的 RDS 扩展支持设置。

有关更多信息，请参阅 Amazon Aurora 版本。

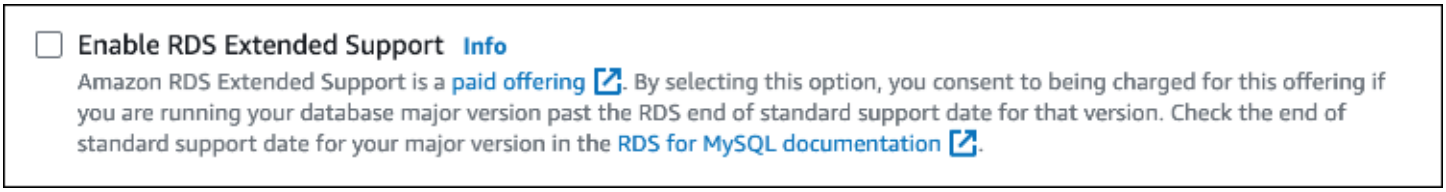
使用 RDS 扩展支持还原 Aurora 数据库集群或全局集群

您可以使用 AWS Management Console、AWS CLI 或 RDS API，借助 RDS 扩展支持版本还原Aurora 数据库集群或全局集群

控制台

在还原 Aurora 数据库集群或全局集群时，请在引擎选项部分选择启用 RDS 扩展支持。如果您未选择此设置，并且要还原的主引擎版本已超过 Aurora 标准支持终止，则 Amazon Aurora 会自动将您的 Aurora 数据库集群或全局集群升级到处于 Aurora 标准支持下的版本。

下图显示了启用 RDS 扩展支持设置：



AWS CLI

在您运行 restore-db-cluster-from-snapshot AWS CLI 命令时，请通过为 `--engine-lifecycle-support` 选项指定 `open-source-rds-extended-support` 来选择 RDS 扩展支持。

如果您想避免支付与 RDS 扩展支持关联的费用，请将 `--engine-lifecycle-support` 选项设置为 `open-source-rds-extended-support-disabled`。默认情况下，此选项设置为 `open-source-rds-extended-support`。

您还可以使用以下 AWS CLI 命令指定该值：

- restore-db-cluster-from-s3
- restore-db-cluster-to-point-in-time

RDS API

在使用 RestoreDBClusterFromSnapshot Amazon RDS API 操作时，请通过将 `EngineLifecycleSupport` 参数设置为 `open-source-rds-extended-support` 来选择 RDS 扩展支持。

如果您想避免支付与 RDS 扩展支持关联的费用，请将 `EngineLifecycleSupport` 参数设置为 `open-source-rds-extended-support-disabled`。默认情况下，此参数设置为 `open-source-rds-extended-support`。

您还可以使用以下 API 操作指定该值：

- RestoreDBClusterFromS3
- RestoreDBClusterToPointInTime

有关还原 Aurora 数据库集群的更多信息，请按照备份和还原 Amazon Aurora 数据库集群中数据库引擎的相关说明操作。

使用 Amazon Aurora 蓝绿部署进行数据库更新

蓝绿部署将生产数据库环境复制到单独的同步暂存环境。通过使用 Amazon Aurora 蓝绿部署，您可以在不影响生产环境的情况下对暂存环境中的数据库进行更改。例如，您可以升级主数据库引擎版本或次要数据库引擎版本，或在暂存环境中更改数据库参数。准备就绪后，可以将暂存环境提升为新的生产数据库环境，在此环境中，停机时间通常不到一分钟。

Amazon Aurora 通过在生产环境中克隆底层 Aurora 存储卷来创建暂存环境。暂存环境中的集群卷仅存储对该环境所做的增量更改。

> **Note**
>
> 目前，Aurora MySQL 和 Aurora PostgreSQL。有关 Amazon RDS 引擎可用性，请参阅《Amazon RDS 用户指南》中的使用 Amazon RDS 蓝绿部署进行数据库更新。

主题

- 适用于 Aurora 的 Amazon RDS 蓝绿部署概述
- 在 Amazon Aurora 中创建蓝绿部署
- 查看 Amazon Aurora 中的蓝绿部署
- 切换 Amazon Aurora 中的蓝绿部署
- 删除 Amazon Aurora 中的蓝绿部署

适用于 Aurora 的 Amazon RDS 蓝绿部署概述

通过使用 Amazon RDS 蓝绿部署，您可以进行数据库更改并测试，然后再在生产环境中实施这些更改。蓝绿部署会创建一个复制生产环境的暂存环境。在蓝绿部署中，蓝色环境是当前的生产环境。绿色环境是暂存环境。暂存环境使用逻辑复制与当前生产环境保持同步。

您可以在绿色环境中更改 Aurora 数据库集群，而不会影响生产工作负载。例如，您可以升级主要或次要数据库引擎版本或在暂存环境中更改数据库参数。您可以彻底测试绿色环境中的变化。准备就绪后，您可以切换环境，以将绿色环境提升为新的生产环境。切换通常需要不到一分钟，不会丢失数据，也无需更改应用程序。

由于绿色环境是生产环境拓扑的副本，因此数据库集群及其所有数据库实例都将在部署中复制。绿色环境还包括数据库集群使用的功能，例如数据库集群快照、Performance Insights、增强监控和 Aurora Serverless v2。

> **Note**
>
> - Aurora MySQL 和 Aurora PostgreSQL 支持蓝绿部署。有关 Amazon RDS 可用性，请参阅《Amazon RDS 用户指南》中的使用 Amazon RDS 蓝绿部署进行数据库更新。
> - 只有以下版本的适用于 Aurora PostgreSQL 的 Babelfish 支持蓝绿部署：
>   - 15.7 及更高版本
>   - 16.3 及更高版本

主题

- 区域和版本可用性
- 使用 Amazon RDS 蓝绿部署的优势
- 蓝绿部署的工作流
- 授权访问 Amazon Aurora 蓝绿部署操作
- Amazon Aurora 蓝绿部署的限制和注意事项
- Amazon Aurora 蓝绿部署最佳实践

区域和版本可用性

功能可用性和支持因每个数据库引擎的特定版本以及 AWS 区域而异。有关更多信息，请参阅 the section called “蓝绿部署”。

使用 Amazon RDS 蓝绿部署的优势

通过使用 Amazon RDS 蓝绿部署，您可以随时了解最新的安全补丁，提高数据库性能，并在短暂且可预测的停机时间内采用更新的数据库功能。蓝绿部署降低了数据库更新（例如主要或次要引擎版本升级）的风险和减少了停机时间。

蓝绿部署提供以下优势：

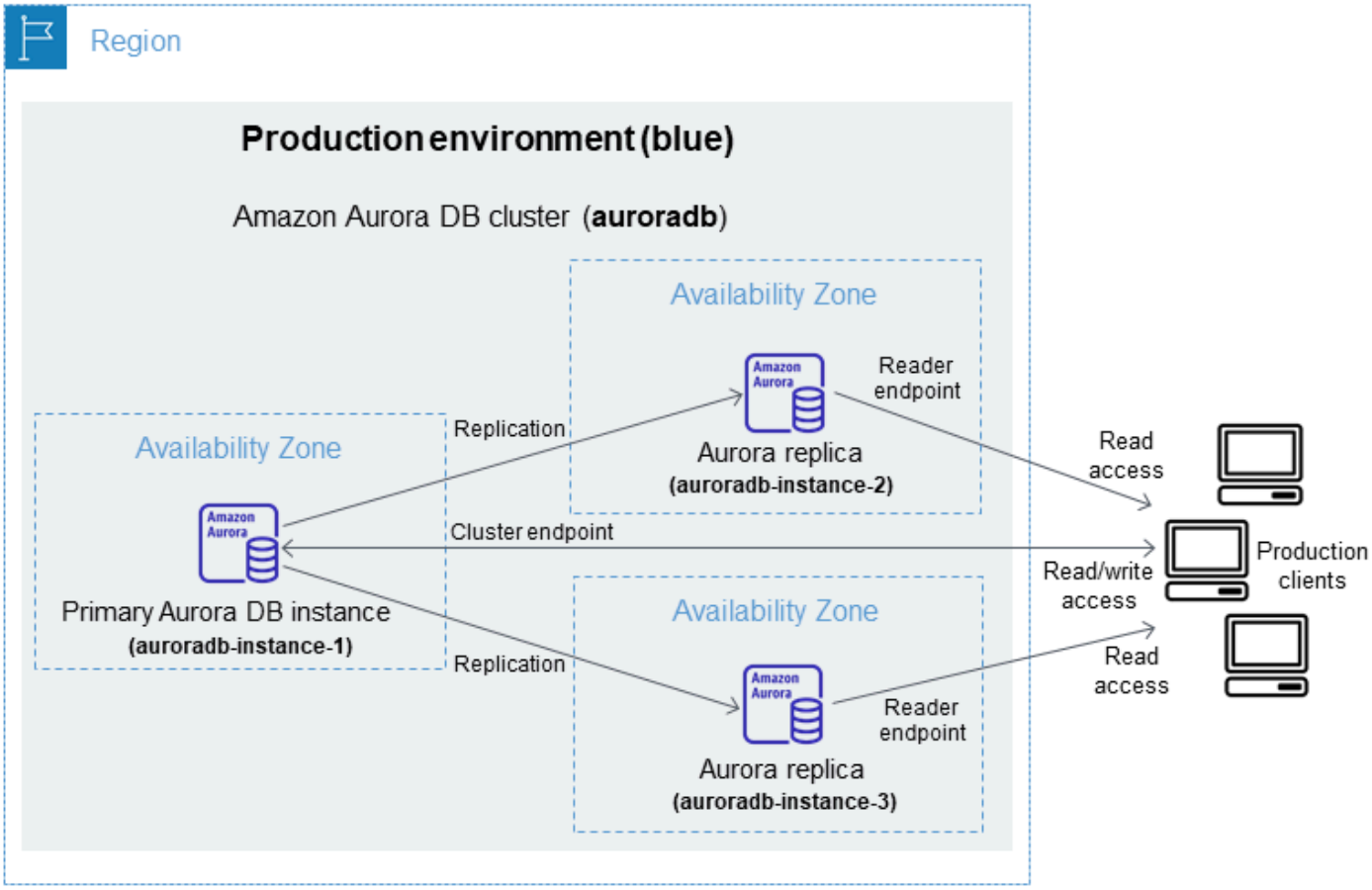
- 轻松创建生产就绪的暂存环境。
- 自动将数据库更改从生产环境复制到暂存环境。
- 在不影响生产环境的情况下在安全的暂存环境中测试数据库更改。
- 通过数据库补丁和系统更新保持最新状态。
- 实施和测试更新的数据库功能。
- 在不更改应用程序的情况下，将您的暂存环境切换为新的生产环境。
- 使用内置切换防护机制安全切换。
- 消除切换期间的数据丢失。
- 快速切换，通常不到一分钟，具体取决于您的工作负载。

蓝绿部署的工作流

使用蓝绿部署进行 Aurora 数据库集群更新时，请完成以下主要步骤。

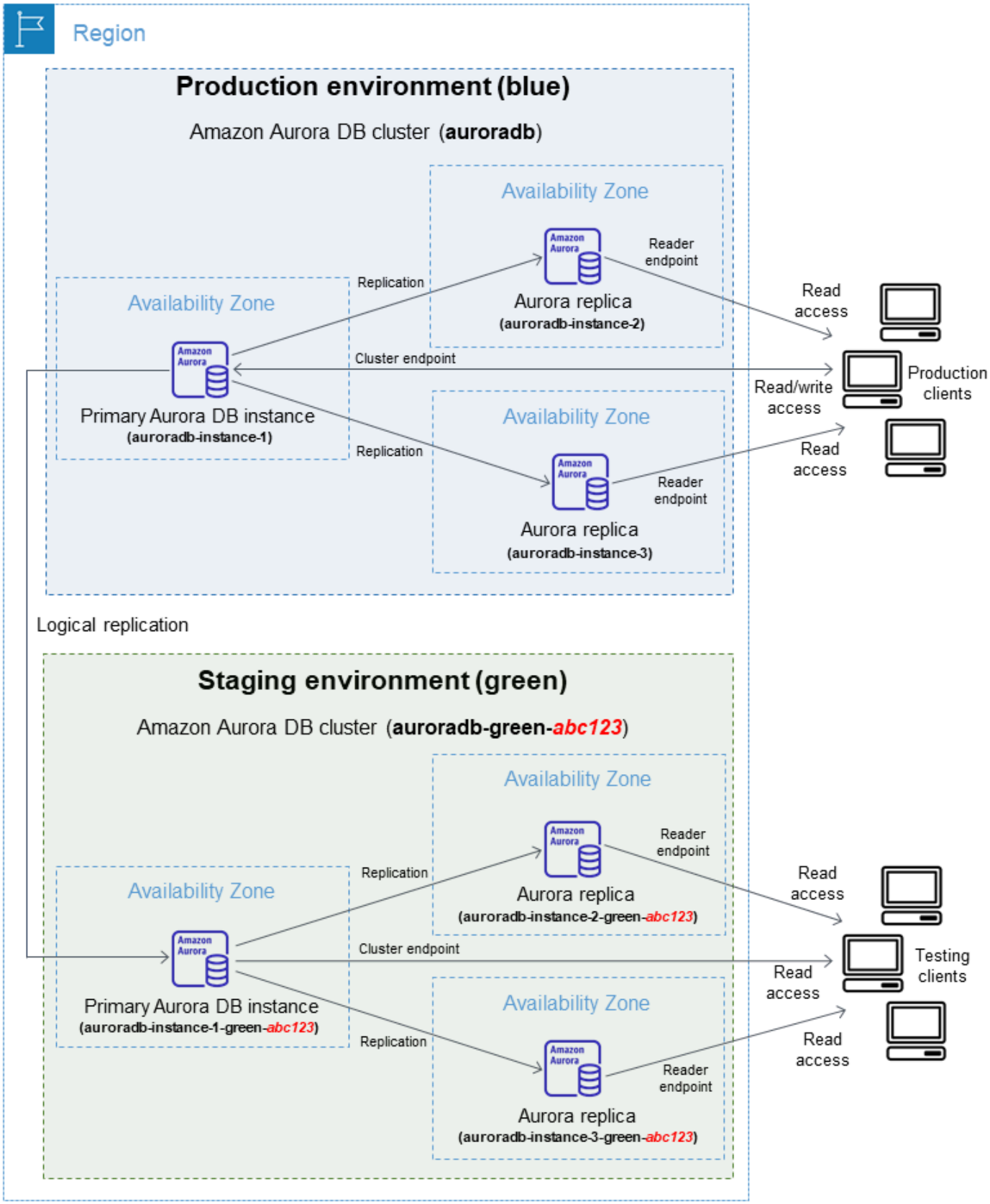
1. 确定需要更新的生产数据库集群。

   下图显示了一个生产数据库集群的示例。



2. 创建蓝绿部署。有关说明，请参阅 在 Amazon Aurora 中创建蓝绿部署。

   下图显示了步骤 1 中生产环境的蓝绿部署示例。在创建蓝绿部署时，RDS 会复制 Aurora 数据库集群的完整拓扑和配置以创建绿色环境。复制的数据库集群和数据库实例的名称附加了 `-green-random-characters`。映像中的暂存环境包含数据库集群（auroradb-green-*abc123*）。它还包含数据库集群中的三个数据库实例（auroradb-instance1-green-*abc123*、auroradb-instance2-green-*abc123* 和 auroradb-instance3-green-*abc123*）。


Region
Production environment (blue)
Amazon Aurora DB cluster (auroradb)
Availability Zone
Amazon Aurora
Reader endpoint
Aurora replica
(auroradb-instance-2)
Replication
Availability Zone
Cluster endpoint
Read access
Production clients
Read/write access
Primary Aurora DB instance
(auroradb-instance-1)
Availability Zone
Replication
Read access
Reader endpoint
Aurora replica
(auroradb-instance-3)
Logical replication
Staging environment (green)
Amazon Aurora DB cluster (auroradb-green-abc123)
Availability Zone
Amazon Aurora
Reader endpoint
Aurora replica
(auroradb-instance-2-green-abc123)
Replication
Availability Zone
Cluster endpoint
Read access
Testing clients
Read access
Primary Aurora DB instance
(auroradb-instance-1-green-abc123)
Availability Zone
Replication
Read access
Reader endpoint
Aurora replica
(auroradb-instance-3-green-abc123)

创建蓝绿部署时，可以为绿色环境中的数据库集群指定更高的数据库引擎版本和不同的数据库集群参数组。您还可以为数据库集群中的数据库实例指定不同的数据库参数组。

RDS 还配置从蓝色环境中的主数据库实例到绿色环境中的主数据库实例的复制。

> **Important**
>
> 对于 Aurora MySQL 版本 3，在创建蓝绿部署后，绿色环境中的数据库集群默认情况下不支持写入操作。但是，这一点不适用于具有 `CONNECTION_ADMIN` 权限的用户，包括 Aurora 主用户。拥有此权限的用户可以覆盖 `read_only` 行为。有关更多信息，请参阅 基于角色的权限模型。

3. 对暂存环境进行更改。

   例如，您可以对数据库进行模式更改，或者更改绿色环境中一个或多个数据库实例使用的数据库实例类。

   有关修改数据库集群的信息，请参阅修改 Amazon Aurora 数据库集群。

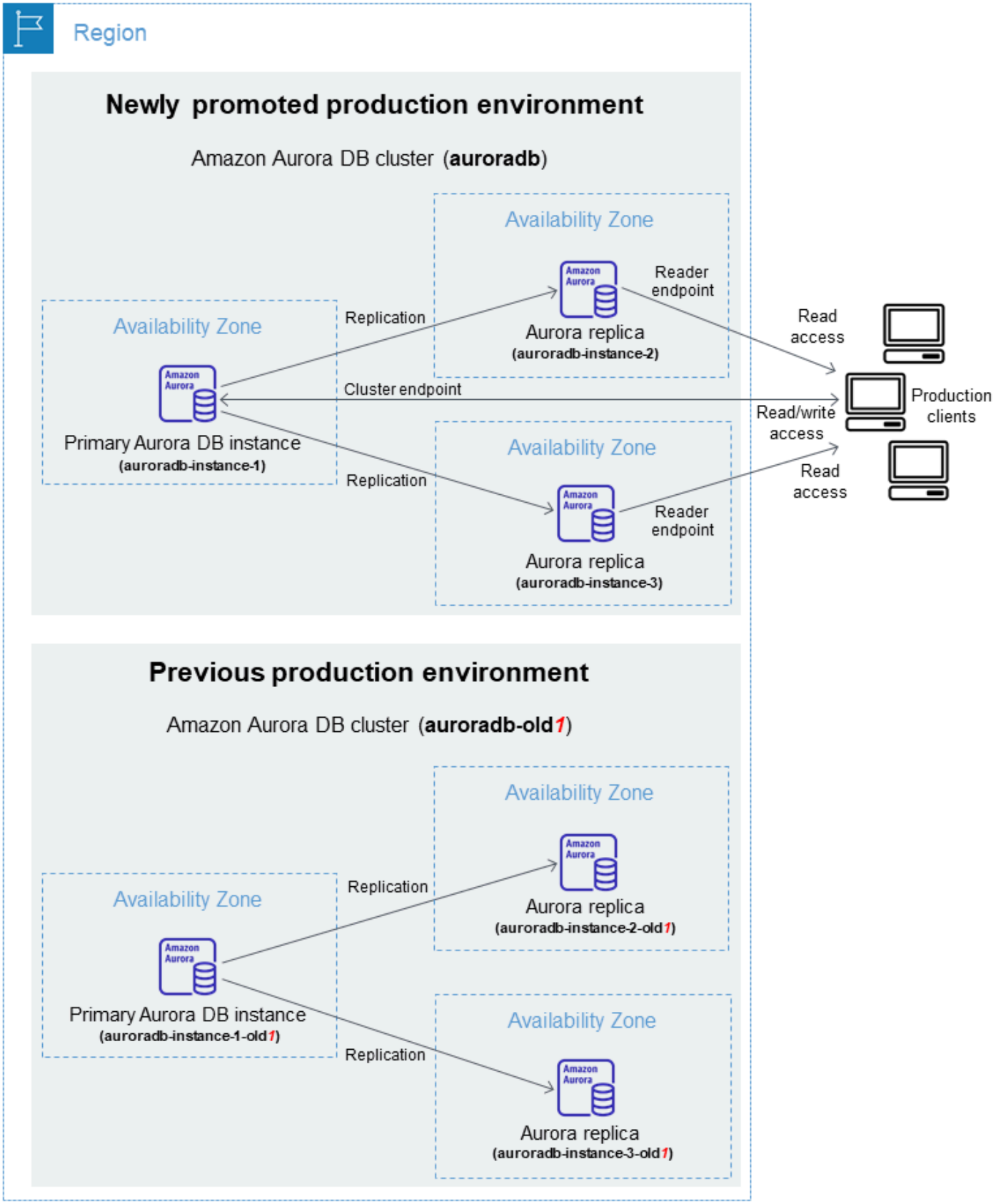
4. 测试您的暂存环境。

   在测试期间，我们建议您将绿色环境中的数据库保持为只读状态。在绿色环境中启用写入操作需谨慎，因为它们可能导致复制冲突。它们还可能导致切换后生产数据库中出现意外数据。要对 Aurora MySQL 启用写入操作，请将 `read_only` 参数设置为 `0`，然后重启数据库实例。对于 Aurora PostgreSQL，请在会话级别将 `default_transaction_read_only` 参数设置为 `off`。

5. 准备就绪后，切换以将暂存环境提升为新的生产环境。有关说明，请参阅 切换 Amazon Aurora 中的蓝绿部署。

   切换会导致停机。停机时间通常不到一分钟，但根据您的工作负载，停机时间可能会更长。

   下图显示了切换后的数据库集群。


Region
Newly promoted production environment
Amazon Aurora DB cluster (auroradb)
Availability Zone
Availability Zone
Amazon Aurora
Reader endpoint
Replication
Aurora replica
(auroradb-instance-2)
Read access
Cluster endpoint
Production clients
Primary Aurora DB instance
(auroradb-instance-1)
Read/write access
Availability Zone
Replication
Read access
Reader endpoint
Aurora replica
(auroradb-instance-3)
Previous production environment
Amazon Aurora DB cluster (auroradb-old1)
Availability Zone
Availability Zone
Replication
Aurora replica
(auroradb-instance-2-old1)
Primary Aurora DB instance
(auroradb-instance-1-old1)
Availability Zone
Replication
Aurora replica
(auroradb-instance-3-old1)

切换后，绿色环境中的 Aurora 数据库集群将成为新的生产数据库集群。当前生产环境中的名称和端点分配给新提升的生产环境，无需更改您的应用程序。因此，您的生产流量现在流向新的生产环境。蓝色环境中的数据库集群和数据库实例通过向当前名称附加 -old*n*（其中 *n* 是一个数字）来重命名。例如，假设蓝色环境中数据库实例的名称为 auroradb-instance-1。切换后，数据库实例名称可能为 auroradb-instance-1-old1。

在本例的映像中，切换期间会发生以下变化：

- 绿色环境数据库集群 auroradb-green-abc123 成为名为 auroradb 的生产数据库集群。
- 名为 auroradb-instance1-green-abc123 的绿色环境数据库实例成为生产数据库实例 auroradb-instance1。
- 名为 auroradb-instance2-green-abc123 的绿色环境数据库实例成为生产数据库实例 auroradb-instance2。
- 名为 auroradb-instance3-green-abc123 的绿色环境数据库实例成为生产数据库实例 auroradb-instance3。
- 名为 auroradb 的蓝色环境数据库集群成为 auroradb-old1。
- 名为 auroradb-instance1 的蓝色环境数据库实例成为 auroradb-instance1-old1。
- 名为 auroradb-instance2 的蓝色环境数据库实例成为 auroradb-instance2-old1。
- 名为 auroradb-instance3 的蓝色环境数据库实例成为 auroradb-instance3-old1。

6. 如果您不再需要蓝绿部署，可将其删除。有关说明，请参阅 删除 Amazon Aurora 中的蓝绿部署。

   切换后，之前的生产环境不会被删除，因此如有必要，您可以将其用于回归测试。

授权访问 Amazon Aurora 蓝绿部署操作

用户必须具有所需的权限才能执行与蓝绿部署相关的操作。您可以创建 IAM policy，以便为用户和角色授予权限以对所需的指定资源执行特定的 API 操作。然后，可以将这些策略附加到需要这些权限的 IAM 权限集或角色。有关更多信息，请参阅 Amazon Aurora 的 Identity and Access Management。

创建蓝绿部署的用户必须具有执行以下 RDS 操作的权限：

- rds:CreateBlueGreenDeployment
- rds:AddTagsToResource
- rds:CreateDBCluster
- rds:CreateDBInstance

- rds:CreateDBClusterEndpoint

切换蓝绿部署的用户必须具有执行以下 RDS 操作的权限：

- rds:SwitchoverBlueGreenDeployment
- rds:ModifyDBCluster
- rds:PromoteReadReplicaDBCluster

删除蓝绿部署的用户必须具有执行以下 RDS 操作的权限：

- rds:DeleteBlueGreenDeployment
- rds:DeleteDBCluster
- rds:DeleteDBInstance
- rds:DeleteDBClusterEndpoint

Aurora 代表您预调配和修改暂存环境中的资源。这些资源包括使用内部定义命名约定的数据库实例。因此，附加的 IAM 策略不能包含部分资源名称模式，例如 my-db-prefix-*。仅支持通配符（*）。通常，我们建议使用资源标签和其它支持的属性来控制对这些资源的访问权限，而不是使用通配符。有关更多信息，请参阅 Actions, resources, and condition keys for Amazon RDS。

Amazon Aurora 蓝绿部署的限制和注意事项

要在 Amazon RDS 中进行蓝绿部署，需要仔细考虑复制插槽、资源管理、实例大小以及对数据库性能的潜在影响等因素。以下各节提供的指导有助于您优化部署策略，以确保尽可能地减少停机时间、实现无缝过渡并有效地管理数据库环境。

主题

- 蓝绿部署的限制
- 蓝绿部署注意事项

蓝绿部署的限制

以下限制适用于蓝绿部署。

主题

- 蓝绿部署的一般限制
- Aurora MySQL 蓝绿部署的限制
- 采用逻辑复制的 Aurora PostgreSQL 蓝绿部署的

蓝绿部署的一般限制

以下一般限制适用于蓝绿部署：

- 您无法停止和启动属于蓝绿部署一部分的集群。
- 蓝绿部署不支持使用 AWS Secrets Manager 管理主用户密码。
- 如果您尝试对蓝色数据库集群强制执行回溯，则蓝绿部署会中断并阻止切换。
- 在切换期间，蓝色和绿色环境无法与 Amazon Redshift 进行零 ETL 集成。您必须先删除集成，接着切换，然后重新创建集成。
- 创建蓝绿部署时，必须在绿色环境中禁用事件调度器（`event_scheduler` 参数）。这样可以防止在绿色环境中生成事件并导致不一致。
- 在蓝色数据库集群上配置的自动扩缩策略不会复制到绿色环境。无论它们最初是在蓝色还是绿色环境中设置的，切换之后都必须重新配置。
- 您无法将未加密的数据库集群更改为加密的数据库集群。此外，无法将加密的数据库集群更改为未加密的数据库集群。
- 无法将蓝色数据库集群更改为比其相应的绿色数据库集群更高的引擎版本。
- 蓝色环境和绿色环境中的资源必须位于同一个 AWS 账户中。
- 以下功能不支持蓝绿部署：
  - Amazon RDS 代理
  - 跨区域只读副本
  - Aurora Serverless v1 数据库集群
  - 属于 Aurora Global Database 的数据库集群
  - AWS CloudFormation

Aurora MySQL 蓝绿部署的限制

以下限制适用于 Aurora MySQL 蓝绿部署：

- 源数据库集群不能包含任何名为 `tmp` 的数据库。具有此名称的数据库将不会复制到绿色环境中。
- 蓝色数据库集群不能是外部二进制日志副本。

- 如果源数据库集群启用了回溯功能，则创建的绿色数据库集群不支持回溯功能。这是因为回溯功能不适用于二进制日志（binlog）复制，而二进制日志复制是蓝绿部署所必需的。有关更多信息，请参阅 the section called “回溯数据库集群”。
- 蓝绿部署不支持适用于 MySQL 的 AWS JDBC 驱动程序。有关更多信息，请参阅 GitHub 上的 Known Limitations。

采用逻辑复制的 Aurora PostgreSQL 蓝绿部署的

以下限制适用于使用逻辑复制的 Aurora PostgreSQL 蓝绿。

- Unlogged（未记录的）表不会复制到绿色环境，除非在蓝色数据库集群上将 `rds.logically_replicate_unlogged_tables` 参数设置为 1。在创建蓝绿部署后不要修改此参数值，以免未记录的表上可能出现复制错误。
- 蓝色数据库集群不能是逻辑源（发布者）或副本（订阅者）。
- 如果将蓝色数据库集群配置为外部数据包装器（FDW）扩展的外部服务器，则必须使用集群端点名称而不是 IP 地址。这会让配置在切换后仍能正常运行。
- 在蓝绿部署中，每个数据库都需要一个逻辑复制插槽。随着数据库数量增长，资源开销会增加，并可能导致复制滞后，尤其是在数据库集群没有充分扩展的情况下。影响取决于数据库工作负载和连接数等因素。要缓解这种情况，可以考虑纵向扩展数据库实例类，或减少源集群上的数据库数量。
- 适用于 Aurora PostgreSQL 的 Babelfish 仅版本 15.7 及更高的 15 版本，以及 16.3 及更高的 16 版本支持蓝绿部署。
- 如果要在 Aurora 副本中捕获执行计划，则必须在调用 `apg_plan_mgmt.create_replica_plan_capture` 函数时提供蓝色数据库集群端点。这样可以确保计划捕获在切换后继续进行。有关更多信息，请参阅 the section called “在副本中捕获 Aurora PostgreSQL 执行计划”。
- 绿色环境中的逻辑复制 apply process 是单线程的。如果蓝色环境生成大量写入流量，则绿色环境可能无法跟上。这可能会导致复制滞后或失败，特别是对于产生持续高写入吞吐量的工作负载。确保彻底测试工作负载。对于需要主要版本升级和处理大量写入工作负载的场景，可以考虑使用其它方法，例如使用 AWS Database Migration Service（AWS DMS）或自行管理的逻辑复制。
- 在执行 Aurora 的蓝绿部署期间，不支持在分区表上创建新分区。创建新分区涉及数据定义语言（DDL）操作（例如 `CREATE TABLE`），这些操作不会从蓝色环境复制到绿色环境。但是，现有的分区表及其数据将复制到绿色环境。
- 以下限制适用于 PostgreSQL 扩展：
  - 创建蓝绿部署时，必须在蓝色环境中禁用 `pg_partman` 扩展。该扩展将执行 `CREATE TABLE` 等 DDL 操作，这会中断从蓝色环境到绿色环境的逻辑复制。

- 创建蓝绿部署后，pg_cron 扩展必须在所有绿色数据库上保持禁用状态。该扩展具有以超级用户身份运行并绕过绿色环境只读设置的后台工作进程，这可能会导致复制冲突。
- 在所有绿色数据库上，apg_plan_mgmt 扩展必须将 apg_plan_mgmt.capture_plan_baselines 参数设置为 off，以免在蓝色环境中捕获相同的计划时发生主键冲突。有关更多信息，请参阅 the section called “Aurora PostgreSQL 查询计划管理概览”。
- 创建蓝绿部署时，必须在蓝色环境中禁用 pglogical 和 pgactive 扩展。将绿色环境切换为新的生产环境后，您可以重新启用这些扩展。此外，蓝色数据库不能是外部实例的逻辑订阅者。
- 如果您使用的是 pgAudit 扩展，则它必须保留在蓝色和绿色数据库实例的自定义数据库参数组上的共享库（shared_preload_libraries）中。有关更多信息，请参阅 the section called “设置 pgAudit 扩展”。

蓝绿部署的特定于逻辑复制的限制

PostgreSQL 有某些与逻辑复制相关的限制，这会导致创建 Aurora PostgreSQL 数据库集群的蓝绿部署存在限制。

下表描述了适用于 Aurora PostgreSQL 的蓝绿部署的逻辑复制限制。有关更多信息，请参阅 PostgreSQL 逻辑复制文档中的限制。

限制	说明
数据定义语言（DDL）语句，（例如 CREATE TABLE 和 CREATE SCHEMA）不会从蓝色环境复制到绿色环境。	如果 Aurora 在蓝色环境中检测到 DDL 更改，则您的绿色数据库将进入复制已降级状态。必须删除蓝绿部署和所有绿色数据库，然后重新创建。
数据控制语言（DCL）语句（例如 GRANT 和 REVOKE）不会从	如果 Aurora 检测到有人试图在蓝色环境中执行 DCL 语句，您将看到一条警告消息。由于这是蓝绿部署过程的限制，因此没有可用于更改此行为的配置或 API。

限制	说明
蓝色环境复制到绿色环境。	
对序列对象执行的 NEXTVAL 操作在蓝色环境和绿色环境之间不同步。	在切换期间，Aurora 会增加绿色环境中的序列值，以匹配蓝色环境中的那些序列值。如果您有成千上万的序列，这可能会延迟切换。
蓝色环境中的大型对象不会复制到绿色环境中。这既包括现有的大型对象，也包括在蓝绿部署过程中任何新创建或修改的大型对象。	如果 Aurora 检测到在蓝色环境中创建或修改了存储在 pg_largeobject 系统表中的大型对象，则您的绿色数据库将进入复制已降级状态。必须删除蓝绿部署和所有绿色数据库，然后重新创建。
刷新实体化视图会导致复制中断。	在蓝色环境中刷新实体化视图，会中断向绿色环境的复制。请避免在蓝色环境中刷新实体化视图。切换后，您可以使用 REFRESH MATERIALIZED VIEW 命令手动刷新实体化视图，也可以按计划进行刷新。
不允许对没有主键的表执行 UPDATE 和 DELETE 操作。	在创建蓝绿部署之前，请确保所有表都有主键或使用 REPLICA IDENTITY FULL。但是，仅当主键或唯一键不存在时才使用 REPLICA IDENTITY FULL，因为它会影响复制性能。有关更多信息，请参阅 PostgreSQL 文档。

蓝绿部署注意事项

Amazon RDS 使用每种资源的 DbiResourceId 和 DbClusterResourceId 跟踪蓝绿部署中的资源。此资源 ID 是资源的 AWS 区域唯一的不可变标识符。

资源 ID 与数据库集群 ID 是分开的。每一个都列在 RDS 控制台的数据库配置中。

当您切换蓝绿部署时，资源的名称（集群 ID）会发生变化，但每个资源都保持相同的资源 ID。例如，在蓝色环境中，数据库集群标识符可能已经为 mycluster。切换后，同一个数据库集群可能重命名为 mycluster-old1。但是，数据库集群的资源 ID 在切换期间不会更改。因此，当将绿色资源切换为新的生产资源时，它们的资源 ID 与之前生产中的蓝色资源 ID 不匹配。

切换蓝绿部署后，请考虑将资源 ID 更新为新转换的生产资源的 ID，以获得与生产资源一起使用的集成功能和服务。具体而言，请考虑以下更新：

- 如果您使用 RDS API 和资源 ID 执行筛选，请在切换后调整筛选中使用的资源 ID。
- 如果您使用 CloudTrail 来审计资源，请调整 CloudTrail 的使用者，以便在切换后跟踪新的资源 ID。有关更多信息，请参阅 监控 AWS CloudTrail 中的 Amazon Aurora API 调用。
- 如果您将数据库活动流用于蓝色环境中的资源，请调整您的应用程序以在切换后监视新流的数据库事件。有关更多信息，请参阅 支持数据库活动流的区域和 Aurora 数据库引擎。
- 如果您使用性能详情API，请在切换后调整 API 调用中的资源 ID。有关更多信息，请参阅 在 Amazon Aurora 上使用性能详情监控数据库负载。

  您可以在切换后监控同名数据库，但它不包含切换之前的数据。
- 如果您在 IAM 策略中使用资源 ID，请确保在必要时添加新转换的资源的资源 ID。有关更多信息，请参阅 Amazon Aurora 的 Identity and Access Management。
- 如果您有与数据库集群关联的 IAM 角色，请务必在切换后重新关联它们。附加的角色不会自动复制到绿色环境。
- 如果您使用 IAM 数据库身份验证对数据库集群进行身份验证，请确保用于数据库访问的 IAM 策略同时包含在策略的 `Resource` 元素下方列出的蓝色和绿色数据库。这是在切换后连接到绿色数据库所必需的。有关更多信息，请参阅 the section called “创建和使用适用于 IAM 数据库访问的 IAM 策略”。
- 如果您想为属于蓝绿部署的数据库集群还原手动数据库集群快照，请确保通过检查快照拍摄时间来还原正确的数据库集群快照。有关更多信息，请参阅 从数据库集群快照还原。
- 切换后，AWS Database Migration Service（AWS DMS）复制任务无法恢复，因为蓝色环境中的检查点在绿色环境中无效。您必须使用新的检查点重新创建 DMS 任务后才能继续复制。
- Amazon Aurora 通过在蓝色环境中克隆底层 Aurora 存储卷来创建绿色环境。绿色集群卷仅存储对绿色环境所做的增量更改。如果您删除蓝色环境中的数据库集群，则绿色环境中底层 Aurora 存储卷的大小增长到完全大小。有关更多信息，请参阅 the section called “克隆 Aurora 数据库集群卷”。
- 当您在蓝绿部署的绿色环境中向数据库集群添加数据库实例时，当您切换时，新的数据库实例不会替换蓝色环境中的数据库实例。但是，新的数据库实例保留在数据库集群中，并成为新的生产环境中的数据库实例。
- 在蓝绿部署的绿色环境中删除数据库集群中的数据库实例时，您无法创建新的数据库实例来替换蓝绿部署中的该实例。

  如果您创建一个与已删除的数据库实例具有相同名称和相同 ARN 的新数据库实例，则它具有不同的 `DbiResourceId`，因此它不属于绿色环境。

如果您在绿色环境中删除数据库集群中的数据库实例，则会导致以下行为：

- 如果蓝色环境中存在同名的数据库实例，则不会将其切换到绿色环境中的数据库实例。不会通过将 `-old`*n* 附加到数据库实例名称来重命名此数据库实例。
- 切换后，任何指向蓝色环境中数据库实例的应用程序都将继续使用相同的数据库实例。

Amazon Aurora 蓝绿部署最佳实践

以下是蓝绿部署的最佳实践。

主题

- 蓝绿部署的一般最佳实践
- Aurora MySQL 蓝绿部署最佳实践
- Aurora PostgreSQL 蓝绿部署最佳实践

蓝绿部署的一般最佳实践

在创建蓝绿部署时，请考虑以下一般最佳实践。

- 切换之前，在绿色环境中全面测试 Aurora 数据库集群。
- 使绿色环境中的数据库保持只读。建议您在绿色环境中谨慎启用写入操作，因为它们可能导致复制冲突。它们还可能导致切换后生产数据库中出现意外数据。
- 如果您使用蓝绿部署来实现架构更改，则仅进行与复制兼容的更改。

  例如，您可以在表末尾添加新列，而无需中断从蓝色部署到绿色部署的复制。但是，模式更改（例如重命名列或重命名表）会中断向绿色部署的复制。

  有关与复制兼容的更改的更多信息，请参阅 MySQL 文档中的在源和副本上使用不同的表定义进行复制以及 PostgreSQL 逻辑复制文档中的限制。
- 为两个环境中的所有连接使用集群端点、读取器端点或自定义端点。请勿使用带有静态或排除列表的实例端点或自定义端点。
- 切换蓝绿部署时，请遵循切换最佳实践。有关更多信息，请参阅 the section called “切换最佳实践”。

Aurora MySQL 蓝绿部署最佳实践

从 Aurora MySQL 数据库集群创建蓝绿部署时，请考虑以下最佳实践。

- 如果绿色环境出现副本滞后，请考虑以下事项：
  - 如果不需要保留二进制日志，请将其禁用，或者在复制赶上进度之前暂时将其禁用。为此，将 `binlog_format` 数据库集群参数重新设置为 `0`，然后重启绿色写入器数据库实例。
  - 在绿色数据库参数组中临时将 `innodb_flush_log_at_trx_commit` 参数设置为 `0`。当复制赶上进度之后，请在切换之前恢复为默认值 `1`。如果使用临时参数值时发生意外关闭或崩溃，请重建绿色环境以避免未检测到的数据损坏。有关更多信息，请参阅the section called “配置刷新日志缓冲区的频率”。

Aurora PostgreSQL 蓝绿部署最佳实践

从 Aurora PostgreSQL 数据库集群创建蓝绿部署时，请考虑以下的最佳实践。

- 监控 Aurora PostgreSQL 逻辑复制直写缓存，并在必要时对缓存缓冲区进行调整。有关更多信息，请参阅 the section called “监控逻辑复制直写缓存”。
- 在蓝色环境中增加 `logical_decoding_work_mem` 数据库参数的值。这样做可以减少磁盘上的解码次数，改为使用内存。有关更多信息，请参阅 the section called “调整工作内存以进行逻辑解码”。
  - 您可以使用 `ReplicationSlotDiskUsage` CloudWatch 指标监控写入磁盘的事务溢出量。该指标提供对复制槽的磁盘使用情况的见解，有助于确定事务数据何时超过内存容量并存储在磁盘上。您可以使用 `FreeableMemory` CloudWatch 指标监控可用内存。有关更多信息，请参阅 the section called “Amazon Aurora 的实例级指标”。
  - 在 Aurora PostgreSQL 版本 14 及更高版本中，您可以使用 `pg_stat_replication_slots` 系统视图监控逻辑溢出文件的大小。
- 创建蓝绿部署之前，请将所有 PostgreSQL 扩展更新到最新版本。有关更多信息，请参阅 the section called “升级 PostgreSQL 扩展”。
- 如果您使用的是 `aws_s3` 扩展，请在创建绿色环境之后，通过 IAM 角色赋予绿色数据库集群访问 Amazon S3 的权限。这允许导入和导出命令在切换后继续运行。有关说明，请参阅the section called “设置 Amazon S3 存储桶的访问权限”。
- 如果您为绿色环境指定更高的引擎版本，请对所有数据库运行 `ANALYZE` 操作来刷新 `pg_statistic` 表。在主要版本升级期间不会传输优化程序统计数据，因此您必须重新生成所有统计数据，来避免出现性能问题。有关主要版本升级期间的其他最佳实践，请参阅the section called “执行主要版本升级”。
- 如果在源中使用触发器来操作数据，请避免将触发器配置为 `ENABLE REPLICA` 或 `ENABLE ALWAYS`。否则，复制系统会传播更改并执行触发器，从而导致重复。
- 长时间运行的事务可能会导致严重的副本延迟。要减少副本延迟，可考虑执行下列操作：

- 减少长期运行的事务和子事务，这些事务和子事务可能会延迟到绿色环境赶上蓝色环境之后再运行。
- 减少蓝色环境上的批量操作，直到绿色环境赶上蓝色环境之后再运行。
- 在创建蓝绿部署之前，对事务繁忙的表启动手动真空冻结操作。有关说明，请参阅执行手动 vacuum 冻结。
- 在 PostgreSQL 12 及更高版本中，请对大型表或繁忙表禁用 `index_cleanup` 参数，以提高蓝色数据库的常规维护效率。有关更多信息，请参阅尽快对表执行 vacuum 操作。

  > **Note**
  >
  > 在执行 vacuum 操作期间定期跳过索引清理可能会导致索引膨胀，从而降低扫描性能。最佳实践是仅在使用蓝绿部署时使用此方法。部署完成后，建议恢复定期的索引维护和清理。

- 如果蓝色和绿色数据库实例的大小不足以应付工作负载，则可能会出现副本延迟。确保您的数据库实例未达到该实例类型的资源限制。有关更多信息，请参阅 the section called “使用 CloudWatch 指标分析资源使用情况”。

- 复制缓慢会导致发送方和接收方经常重启，从而延迟同步。要确保它们保持活动状态，请在蓝色环境中将 `wal_sender_timeout` 参数设置为 `0`，在绿色环境中将 `wal_receiver_timeout` 参数设置为 `0`，从而禁用超时。
- 检查 UPDATE 和 DELETE 语句的性能，并评估在 WHERE 子句中使用的列上创建索引能否优化这些查询。在绿色环境中重播操作时，这可以提高性能。有关更多信息，请参阅the section called “检查谓词筛选条件是否存在生成等待的查询”。
- 如果您使用的是触发器，请确保它们不会影响名称以“rds”开头的 `pg_catalog.pg_publication`、`pg_catalog.pg_subscription` 和 `pg_catalog.pg_replication_slots` 对象的创建、更新及删除。
- 如果在源数据库集群上启用了 Babelfish，则绿色环境的目标数据库集群参数组中以下参数的设置必须与源数据库集群参数组中的设置相同：
  - rds.babelfish_status
  - babelfishpg_tds.tds_default_numeric_precision
  - babelfishpg_tds.tds_default_numeric_scale
  - babelfishpg_tsql.default_locale
  - babelfishpg_tsql.migration_mode
  - babelfishpg_tsql.server_collation_name

有关这些参数的更多信息，请参阅 the section called “Babelfish 的数据库集群参数组设置 ”。

在 Amazon Aurora 中创建蓝绿部署

RDS 将蓝色环境的拓扑和功能复制到暂存区域。如果蓝色数据库实例具有只读副本，它们将被复制为绿色实例的副本。所有绿色副本分配得到的存储与绿色主实例一致，而其他存储参数则继承自蓝色副本。

创建蓝绿部署时，您需要指定要在部署中复制的数据库集群。您选择的数据库集群是生产数据库集群，它将成为蓝色环境中的数据库集群。RDS 将蓝色环境的拓扑及其配置的功能复制到暂存区域。数据库集群复制到绿色环境，RDS 配置从蓝色环境中的数据库集群到绿色环境中的数据库集群的复制。RDS 还复制数据库集群中的所有数据库实例。

主题

- 准备进行蓝绿部署
- 创建蓝绿部署时指定更改
- 创建蓝绿部署
- 创建蓝绿部署的设置

准备进行蓝绿部署

在创建蓝绿部署之前，必须执行某些步骤，具体取决于您的 Aurora 数据库集群运行的引擎。

主题

- 准备 Aurora MySQL 数据库集群以进行蓝绿部署
- 准备 Aurora PostgreSQL 数据库集群以进行蓝绿部署

准备 Aurora MySQL 数据库集群以进行蓝绿部署

在为 Aurora MySQL 数据库集群创建蓝绿部署之前，该集群必须与开启二进制日志记录（`binlog_format`）的自定义数据库集群参数组相关联。从蓝色环境复制到绿色环境需要二进制日志记录。尽管任何二进制日志格式都有效，但我们建议使用 `ROW` 以降低复制不一致的风险。有关创建自定义数据库集群参数组和设置参数的信息，请参阅the section called “数据库集群参数组”。

> **Note**
>
> 启用二进制日志记录会增加集群的写入磁盘 I/O 操作数。您可以使用 `VolumeWriteIOPs` CloudWatch 指标监控 IOPS 使用情况。

启用二进制日志记录后，请务必重启数据库集群以使您的更改生效。蓝绿部署要求写入器实例与数据库集群参数组同步，否则创建将失败。有关更多信息，请参阅 重启 Aurora 集群内的数据库实例。

此外，建议将二进制日志保留期更改为 NULL 以外的其他值，以防止二进制日志文件被清除。有关更多信息，请参阅 the section called “设置和显示二进制日志配置”。

准备 Aurora PostgreSQL 数据库集群以进行蓝绿部署

在为 Aurora PostgreSQL 数据库集群创建蓝绿部署之前，请务必执行以下操作：

- 将集群与启用逻辑复制（`rds.logical_replication`）的自定义数据库集群参数组相关联。从蓝色环境复制到绿色环境需要逻辑复制。

  启用逻辑复制时，还需要调整某些集群参数，例如 `max_replication_slots`、`max_logical_replication_workers` 和 `max_worker_processes`。有关启用逻辑复制和调整这些参数的说明，请参阅the section called “设置逻辑复制”。

  此外，请确保 `synchronous_commit` 参数设置为 `on`。

  配置所需参数后，重启数据库集群以使您的更改生效。蓝绿部署要求写入器实例与数据库集群参数组同步，否则创建将失败。有关更多信息，请参阅 重启 Aurora 集群内的数据库实例。
- 确认您的数据库集群运行的 Aurora PostgreSQL 版本与蓝绿部署兼容。有关兼容版本列表，请参阅 the section called “Aurora PostgreSQL 的蓝绿部署”。
- 确保数据库集群中的所有表都有主键。PostgreSQL 逻辑复制不允许对没有主键的表执行 UPDATE 或 DELETE 操作。

创建蓝绿部署时指定更改

创建蓝绿部署时，可以在绿色环境中对数据库集群进行以下更改：

部署后，您可以在绿色环境中对数据库集群及其数据库实例进行其他修改。例如，您可以指定更高的引擎版本或不同的参数组。

有关修改数据库集群的信息，请参阅修改 Amazon Aurora 数据库集群。

主题

- 指定更高的引擎版本
- 指定其它数据库参数组

指定更高的引擎版本

如果要测试数据库引擎升级，可以指定更高的引擎版本。切换后，数据库将升级到您指定的主要或次要数据库引擎版本。

指定其它数据库参数组

指定与数据库集群使用的数据库集群参数组不同的数据库集群参数组。您可以测试参数更改如何影响绿色环境中的数据库集群，或者在升级时为新的主要数据库引擎版本指定参数组。

如果您指定不同的数据库集群参数组，则指定的参数组将与绿色环境中的数据库集群相关联。如果您未指定其他数据库集群参数组，则绿色环境中的数据库集群将和与蓝色数据库集群相同的参数组关联。

创建蓝绿部署

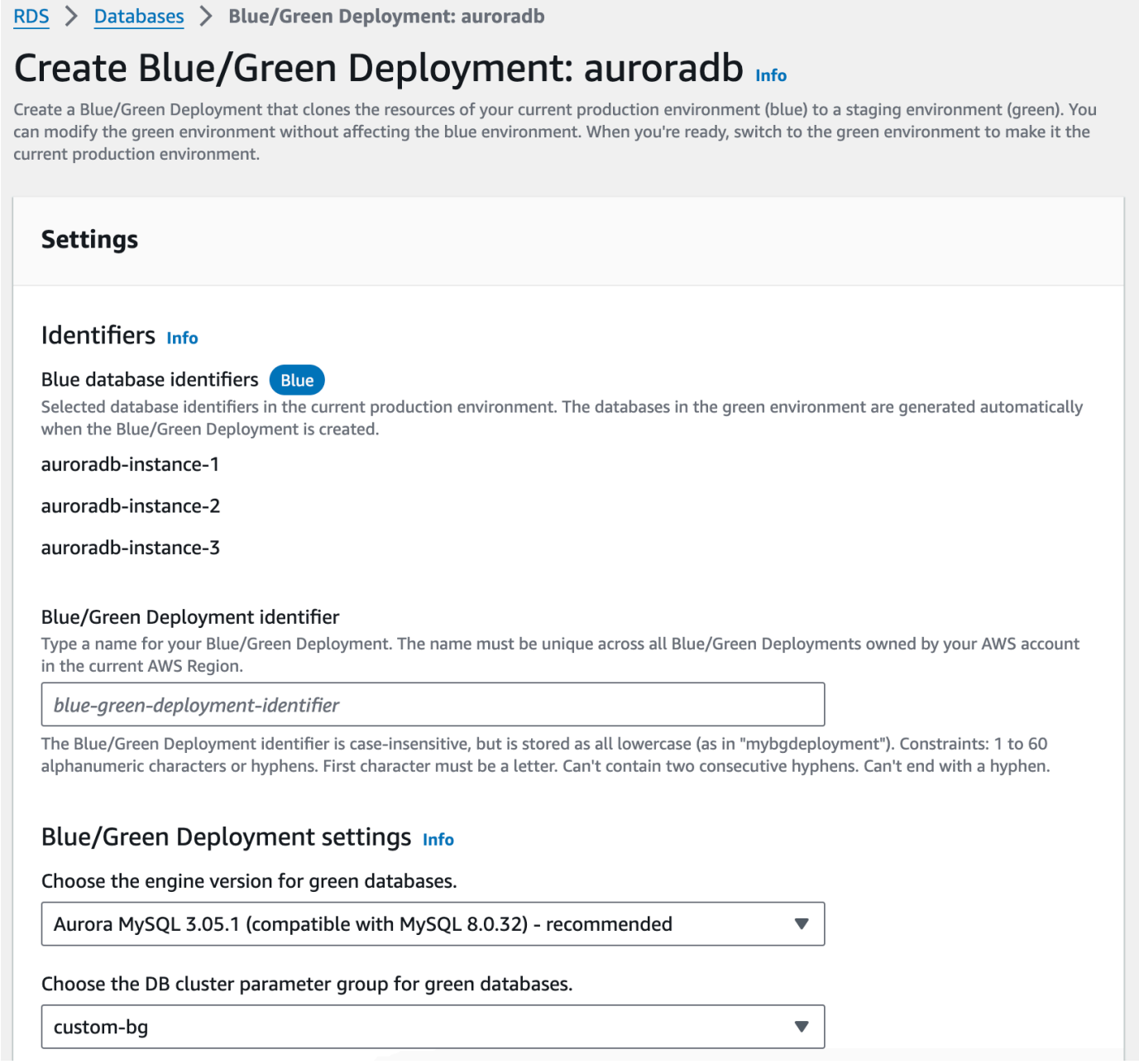
您可以使用 AWS Management Console、AWS CLI 或 RDS API 创建蓝绿部署。

控制台

创建蓝绿部署

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Databases（数据库），然后选择要复制到绿色环境的数据库集群。
3. 依次选择操作和创建蓝绿部署。

   将出现创建蓝绿部署页面。



4. 查看蓝色数据库标识符。确保它们与您在蓝色环境中预期的数据库实例相匹配。如果不符合预期，请选择 Cancel（取消）。
5. 对于蓝绿部署名称，输入蓝绿部署的名称。
6. 在剩余部分中，指定绿色环境的设置。有关每项设置的信息，请参阅 the section called “可用设置”。

   部署后，可以在绿色环境中对数据库进行其他修改。
7. 选择创建。

AWS CLI

要使用 AWS CLI 创建蓝绿部署，请使用 create-blue-green-deployment 命令。有关所有可用选项的信息，请参阅the section called “可用设置”。

Example

对于 Linux、macOS 或 Unix：

```
aws rds create-blue-green-deployment \
    --blue-green-deployment-name aurora-blue-green-deployment \
    --source arn:aws:rds:us-east-2:123456789012:cluster:auroradb \
    --target-engine-version 8.0 \
    --target-db-cluster-parameter-group-name mydbclusterparametergroup
```

对于 Windows：

```
aws rds create-blue-green-deployment ^
    --blue-green-deployment-name aurora-blue-green-deployment ^
    --source arn:aws:rds:us-east-2:123456789012:cluster:auroradb ^
    --target-engine-version 8.0 ^
    --target-db-cluster-parameter-group-name mydbclusterparametergroup
```

RDS API

要使用 Amazon RDS API 创建蓝绿部署，请使用 CreateBlueGreenDeployment 操作。有关各选项的信息，请参阅 the section called “可用设置”。

创建蓝绿部署的设置

下表说明创建蓝绿部署时可供您选择的设置。有关 AWS CLI 选项的更多信息，请参阅 create-blue-green-deployment。有关 RDS API 参数的更多信息，请参阅 CreateBlueGreenDeployment。

控制台设置	设置说明	CLI 选项和 RDS API 参数
蓝绿部署标识符	蓝绿部署的名称。	CLI 选项： `--blue-green-deployment-name`

控制台设置	设置说明	CLI 选项和 RDS API 参数
		API 参数： `BlueGreenDeploymentName`

蓝色数据库标识符	要复制到绿色环境的集群的标识符。使用 CLI 或 API 时，请指定集群的 Amazon 资源名称（ARN）。	CLI 选项：
		
		`--source`
		
		API 参数：
		
		`Source`

绿色数据库的数据库集群参数组	与绿色环境中的数据库相关联的参数组。	CLI 选项：
		
		`--target-db-cluster-parameter-group-name`
		
		API 参数：
		
		`TargetDBClusterParameterGroupName`

绿色数据库的引擎版本	将绿色环境中的集群升级到指定的数据库引擎版本。
	
	如果您选择使用逻辑复制的 Aurora PostgreSQL 数据库集群，请回顾并确认逻辑复制的限制。有关更多信息，请参阅 the section called “蓝绿部署的特定于逻辑复制的限制”。
		CLI 选项：
		
		`--target-engine-version`
		
		RDS API 参数：
		
		`TargetEngineVersion`

查看 Amazon Aurora 中的蓝绿部署

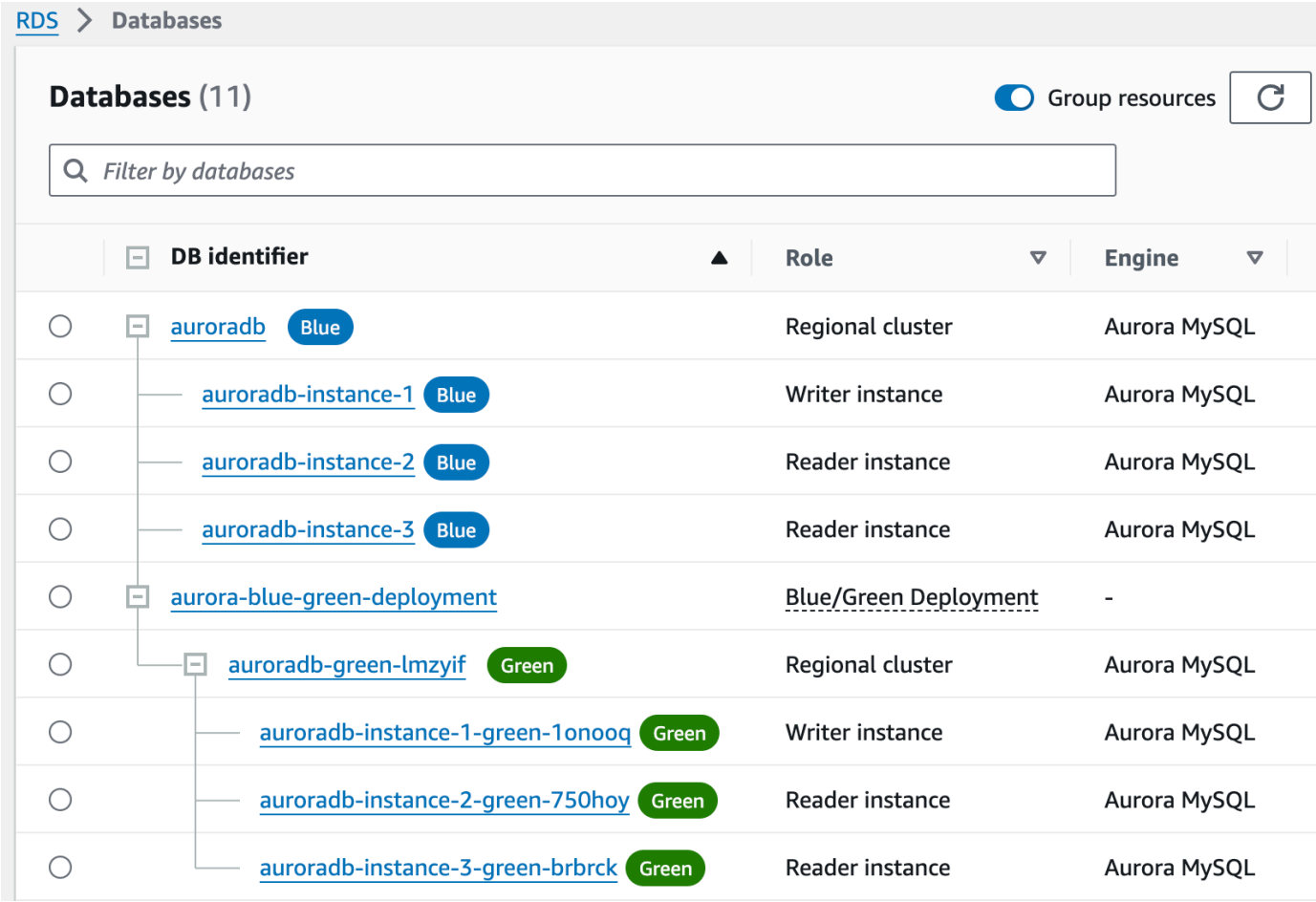
您可以使用 AWS Management Console、AWS CLI 或 RDS API 查看有关蓝绿部署的详细信息。

您还可以查看和订阅事件，以了解有关蓝绿部署的信息。有关更多信息，请参阅 蓝绿部署事件。

控制台

查看有关蓝绿部署的详细信息

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Databases（数据库），然后在列表中找到蓝绿部署。

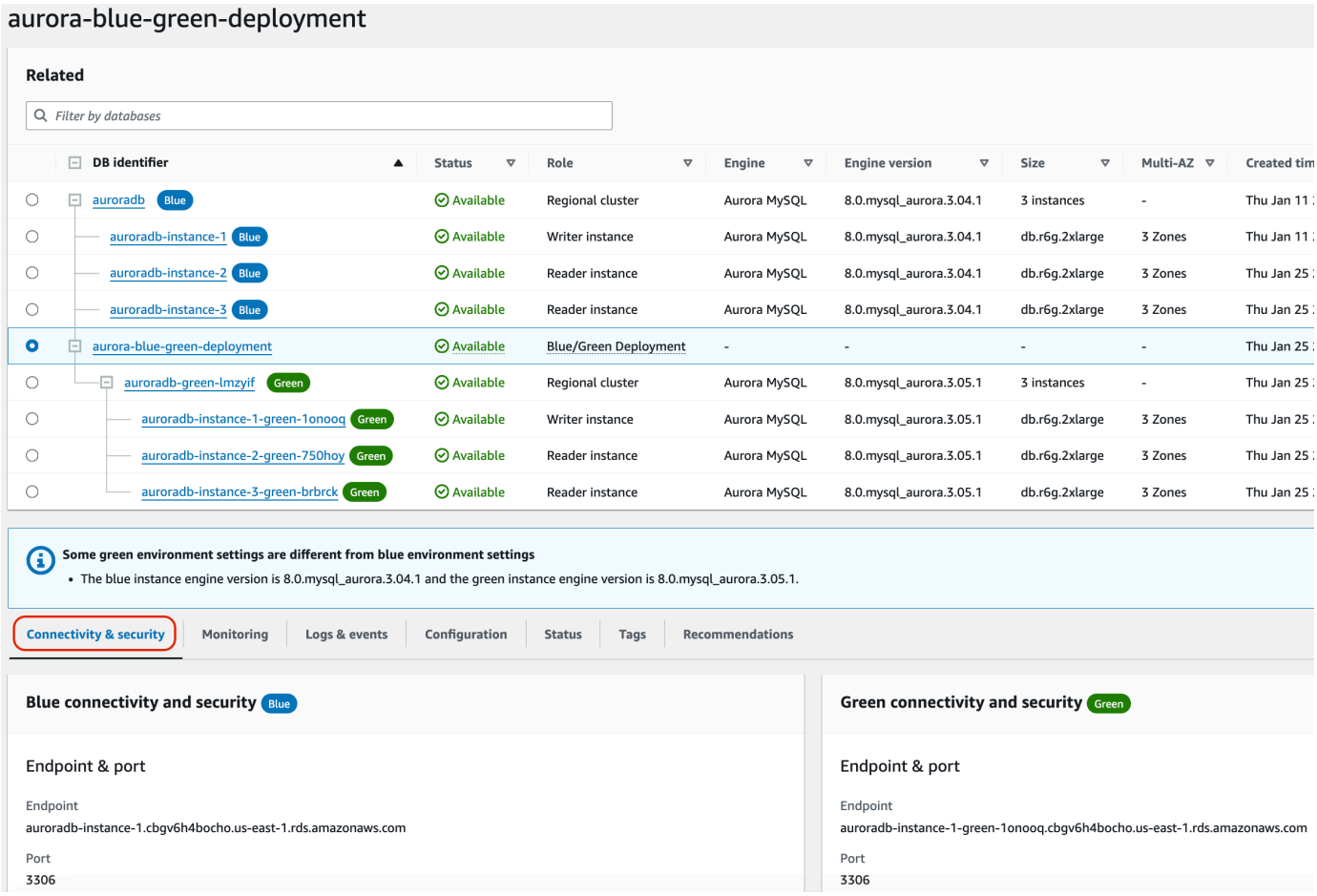


蓝绿部署的 Role（角色）值为 Blue/Green Deployment（蓝绿部署）。

3. 选择要查看的蓝绿部署的名称以显示其详细信息。

每个选项卡都有一个用于蓝色部署的部分和一个用于绿色部署的部分。例如，在配置选项卡上，如果在绿色环境中升级数据库引擎版本，则蓝色环境和绿色环境中的数据库引擎版本可能不同。

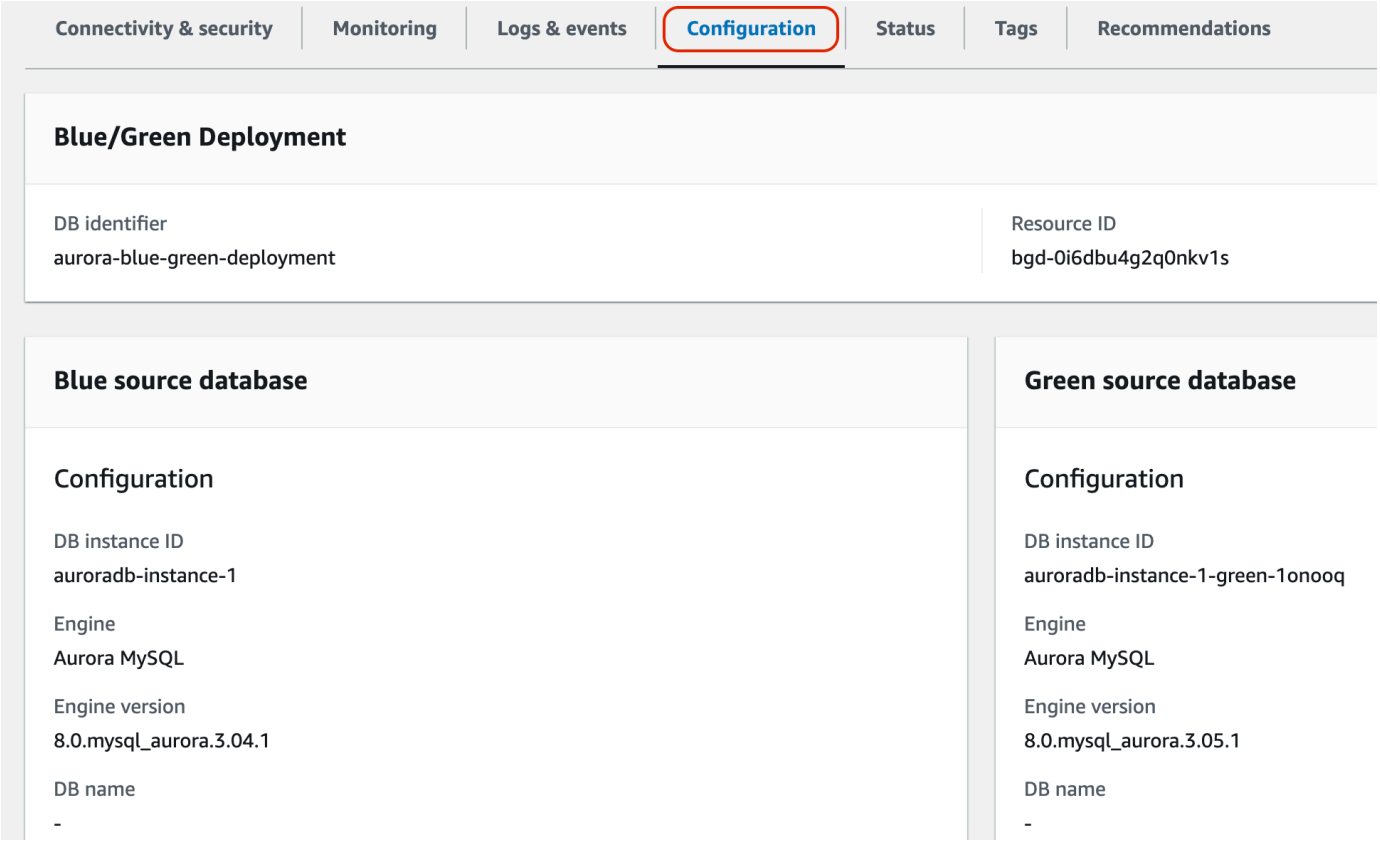
下图显示连接和安全选项卡的示例：



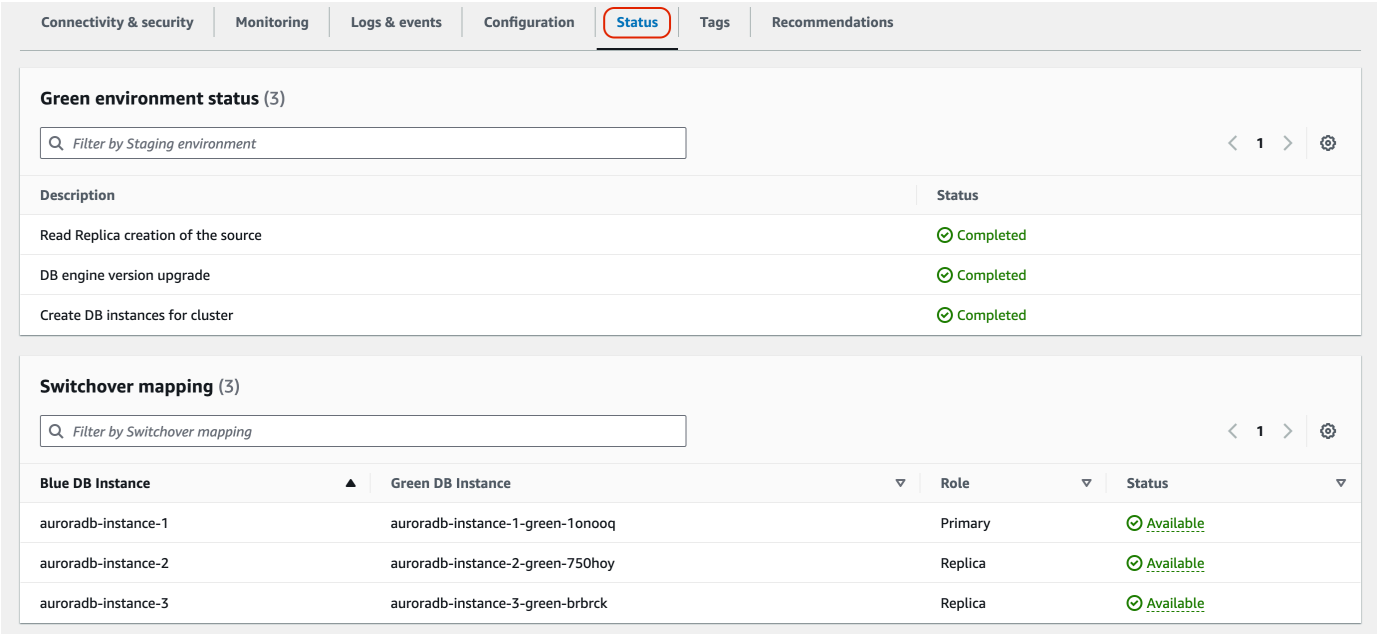
连接和安全选项卡还包括一个名为复制的部分，其中显示了复制的当前状态以及蓝色和绿色环境之间的副本滞后。如果复制状态为 `Replicating`，则表示蓝绿部署复制成功。

对于使用逻辑复制的 Aurora PostgreSQL 蓝绿，如果您在蓝色环境中进行了不受支持的 DDL 或大型对象更改，则复制状态会变成 `Replication degraded`。有关更多信息，请参阅 the section called “蓝绿部署的特定于逻辑复制的限制”。

下图显示配置选项卡的示例：



下图显示状态选项卡的示例：



AWS CLI

要使用 AWS CLI 查看有关蓝绿部署的详细信息，请使用 describe-blue-green-deployments 命令。

Example 通过筛选蓝绿部署的名称来查看有关蓝绿部署的详细信息

当您使用 describe-blue-green-deployments 命令时，可以按 `--blue-green-deployment-name` 进行筛选。

以下示例显示名为 *my-blue-green-deployment* 的蓝绿部署的详细信息。

```
aws rds describe-blue-green-deployments \
  --filters Name=blue-green-deployment-name,Values=my-blue-green-deployment
```

Example 通过指定蓝绿部署的标识符查看有关蓝绿部署的详细信息

使用 describe-blue-green-deployments 命令时，您可以指定 `--blue-green-deployment-identifier` 选项。

以下示例显示带有标识符 *bgd-1234567890abcdef* 的蓝绿部署的详细信息。

```
aws rds describe-blue-green-deployments \
  --blue-green-deployment-identifier bgd-1234567890abcdef
```

RDS API

要使用 Amazon RDS API 查看有关蓝绿部署的详细信息，请使用 DescribeBlueGreenDeployments 操作并指定 `BlueGreenDeploymentIdentifier`。

切换 Amazon Aurora 中的蓝绿部署

切换将绿色环境中的数据库集群（包括其数据库实例）转换为生产数据库集群。在切换之前，生产流量将路由到蓝色环境中的集群。切换后，生产流量将路由到绿色环境中的数据库集群。

切换 蓝绿部署不同于在蓝绿部署中提升 绿色数据库集群。如果您通过从操作菜单中选择提升来手动提升绿色数据库集群，则蓝色环境与绿色环境之间的复制会中断，蓝绿部署将进入无效配置状态。

主题

- 切换超时

- 切换防护机制
- 切换操作
- 切换最佳实践
- 在切换之前验证 CloudWatch 指标
- 在切换之前监控副本滞后
- 切换蓝绿部署
- 切换后

切换超时

您可以指定 30 秒与 3600 秒（一小时）之间的切换超时时间。如果切换所花的时间超过指定的持续时间，则所有更改都将回滚，并且不会对任一环境进行任何更改。默认的超时期间为 300 秒（5 分钟）。

切换防护机制

当您开始切换时，Amazon RDS 会运行一些基本检查，以测试蓝绿环境是否准备好进行切换。这些检查称为切换防护机制。如果环境还没有准备好进行切换，这些切换防护机制可以防止发生切换。因此，它们可以避免比预期更长的停机时间，并防止切换开始时可能导致的蓝色和绿色环境之间的数据丢失。

Amazon RDS 在绿色环境中运行以下防护机制检查：

- 复制运行状况 - 检查绿色数据库集群复制状态是否为正常运行。绿色数据库集群是蓝色数据库集群的副本。
- 复制滞后 - 检查绿色数据库集群的副本滞后是否在切换的允许范围内。允许的限制基于指定的超时期间。副本滞后表示绿色数据库集群滞后于其蓝色数据库集群的程度。有关更多信息，请参阅 the section called “在切换之前监控副本滞后”。
- 主动写入 - 确保绿色数据库集群上没有主动写入。

Amazon RDS 在蓝色环境中运行以下防护机制检查：

- 外部复制 – 对于 Aurora PostgreSQL，请确保蓝色环境不是自行管理的逻辑源（发布者）或副本（订阅用户）。如果是，建议您删除蓝色环境中所有数据库的自行管理的复制插槽和订阅，继续切换，然后重新创建它们以恢复复制。对于 Aurora MySQL，请确保蓝色数据库不是外部二进制日志副本。如果是，请确保它未在主动复制。

- 长时间运行的活动写入 - 确保蓝色数据库集群上没有长时间运行的活动写入，因为它们会增加副本滞后。
- 长时间运行的 DDL 语句 - 确保蓝色数据库集群上没有长时间运行的 DDL 语句，因为它们会增加副本滞后。
- 不支持的 PostgreSQL 更改 – 对于使用逻辑复制的 Aurora PostgreSQL，确保在蓝色环境中没有进行 DDL 更改，也没有添加或修改大型对象。有关更多信息，请参阅 the section called “蓝绿部署的特定于逻辑复制的限制”。

  如果 Amazon RDS 检测到不支持的 PostgreSQL 更改，它会将复制状态更改为 `Replication degraded`，并通知您蓝绿部署无法进行切换。要继续切换，建议您删除并重新创建蓝绿部署和所有绿色数据库。为此，请选择操作、使用绿色数据库删除。

切换操作

当您切换蓝绿部署时，RDS 会执行以下操作：

1. 运行防护机制检查，以验证蓝色和绿色环境是否已准备好进行切换。
2. 在两个环境中停止对数据库集群进行新的写入操作。
3. 在这两个环境中删除与数据库实例的连接，并且不允许新的连接。
4. 等待复制在绿色环境中赶上进度，以便绿色环境与蓝色环境同步。
5. 重命名这两个环境中的数据库集群和数据库实例。

   RDS 重命名绿色环境中的数据库集群和数据库实例，以匹配蓝色环境中相应的数据库集群和数据库实例。例如，假设蓝色环境中数据库实例的名称为 `mydb`。还假设绿色环境中相应的数据库实例的名称为 `mydb-green-abc123`。在切换期间，绿色环境中数据库实例的名称更改为 `mydb`。

   RDS 通过在当前名称后面附加 `-old`*n*（其中 *n* 是一个数字）来重命名蓝色环境中的数据库集群和数据库实例。例如，假设蓝色环境中数据库实例的名称为 `mydb`。切换后，数据库实例名称可能为 `mydb-old1`。

   RDS 还重命名绿色环境中的端点，以匹配蓝色环境中的相应端点，以便无需更改应用程序。
6. 允许在这两种环境中连接到数据库。
7. 允许在新的生产环境中对数据库集群进行写入操作。

   切换之后，先前的生产数据库集群仅允许读取操作。即使您对数据库集群启用写入，它仍为只读的，直到您删除蓝绿部署。

您可以使用 Amazon EventBridge 监控切换的状态。有关更多信息，请参阅 the section called “蓝绿部署事件”。

如果在蓝色环境中配置了标签，则这些标签将在切换期间复制到新的生产环境。有关标签的更多信息，请参阅 为 Amazon Aurora 和 Amazon RDS 资源添加标签。

如果切换开始后由于任何原因在完成之前停止，则所有更改都将回滚，并且不会对任一环境进行任何更改。

切换最佳实践

在切换之前，我们强烈建议您通过完成以下任务来遵循最佳实践：

- 在绿色环境中彻底测试资源。确保它们正常高效地运行。
- 监控相关的 Amazon CloudWatch 指标。有关更多信息，请参阅 the section called “在切换之前验证 CloudWatch 指标”。
- 确定最佳切换时间。

  在切换期间，两个环境中的数据库都会切断写入操作。确定生产环境中流量最低的时间。长时间运行的事务（例如活动的 DDL）会延长您的切换时间，从而延长生产工作负载的停机时间。

  如果您的数据库集群和数据库实例上有大量连接，请考虑在切换蓝绿部署之前，手动将连接减少到应用程序所需的最低数量。实现此目标的一种方法是创建一个脚本，该脚本监控蓝绿部署的状态，并在检测到状态已更改为 `SWITCHOVER_IN_PROGRESS` 时开始清理连接。
- 确保两个环境中的数据库集群和数据库实例均处于 `Available` 状态。
- 确保绿色环境中的数据库集群运行状况良好且正在复制。
- 确保您的网络和客户端配置不会将 DNS 缓存存活时间（TTL）增加到五秒以上，这是 Aurora DNS 区域的默认值。
  否则，应用程序将在切换后继续向蓝色环境发送写入流量。
- 对于使用逻辑复制的 Aurora PostgreSQL 蓝绿部署，执行以下操作：
  - 在切换之前，请查看逻辑复制限制并采取任何必需的操作。有关更多信息，请参阅 the section called “蓝绿部署的特定于逻辑复制的限制”。
  - 运行 `ANALYZE` 操作以刷新 `pg_statistics` 表。这样可以降低切换后出现性能问题的风险。

> **Note**
>
> 在切换期间，您无法修改切换中包含的任何数据库集群。

在切换之前验证 CloudWatch 指标

在切换蓝绿部署之前，建议您检查 Amazon CloudWatch 中以下指标的值。

- `DatabaseConnections` – 使用此指标估算蓝绿部署上的活动水平，并在切换之前确保该值处于部署的可接受水平。如果开启了性能详情，则 `DBLoad` 是更准确的指标。
- `ActiveTransactions` – 如果在任何数据库实例的数据库参数组中 `innodb_monitor_enable` 设置为 `all`，请使用此指标来查看是否存在大量可能阻止切换的活跃事务。

有关更多信息，请参阅 the section called “Aurora 的 CloudWatch 指标”。

在切换之前监控副本滞后

在您切换蓝绿部署之前，请确保副本滞后接近于零，以减少停机时间。

Aurora MySQL

对于 MySQL 蓝绿部署，请在绿色环境中检查 `AuroraBinlogReplicaLag` CloudWatch 指标，以确定当前的副本滞后。有关更多信息，请参阅 the section called “诊断并解决只读副本之间的滞后”。

Aurora PostgreSQL

对于的 PostgreSQL 蓝绿部署，请在蓝色环境中检查 `OldestReplicationSlotLag` CloudWatch 指标，以确定当前的副本滞后。有关更多信息，请参阅 the section called “Amazon Aurora 的实例级指标”。

此外，您还可以在蓝色环境中运行以下 SQL 查询：

```
SELECT slot_name,
       confirmed_flush_lsn as flushed,
       pg_current_wal_lsn(),
       (pg_current_wal_lsn() - confirmed_flush_lsn) AS lsn_distance
FROM pg_catalog.pg_replication_slots
```

```
WHERE slot_type = 'logical';

slot_name        |    flushed    | pg_current_wal_lsn | lsn_distance
-----------------+---------------+--------------------+------------
logical_replica1 | 47D97/CF32980 | 47D97/CF3BAC8      | 37192
```

`confirmed_flush_lsn` 表示发送到副本的最后一个日志序列号（LSN）。`pg_current_wal_lsn` 表示数据库现在的位置。如果 `lsn_distance` 为 0，则表示已捕获副本。

切换蓝绿部署

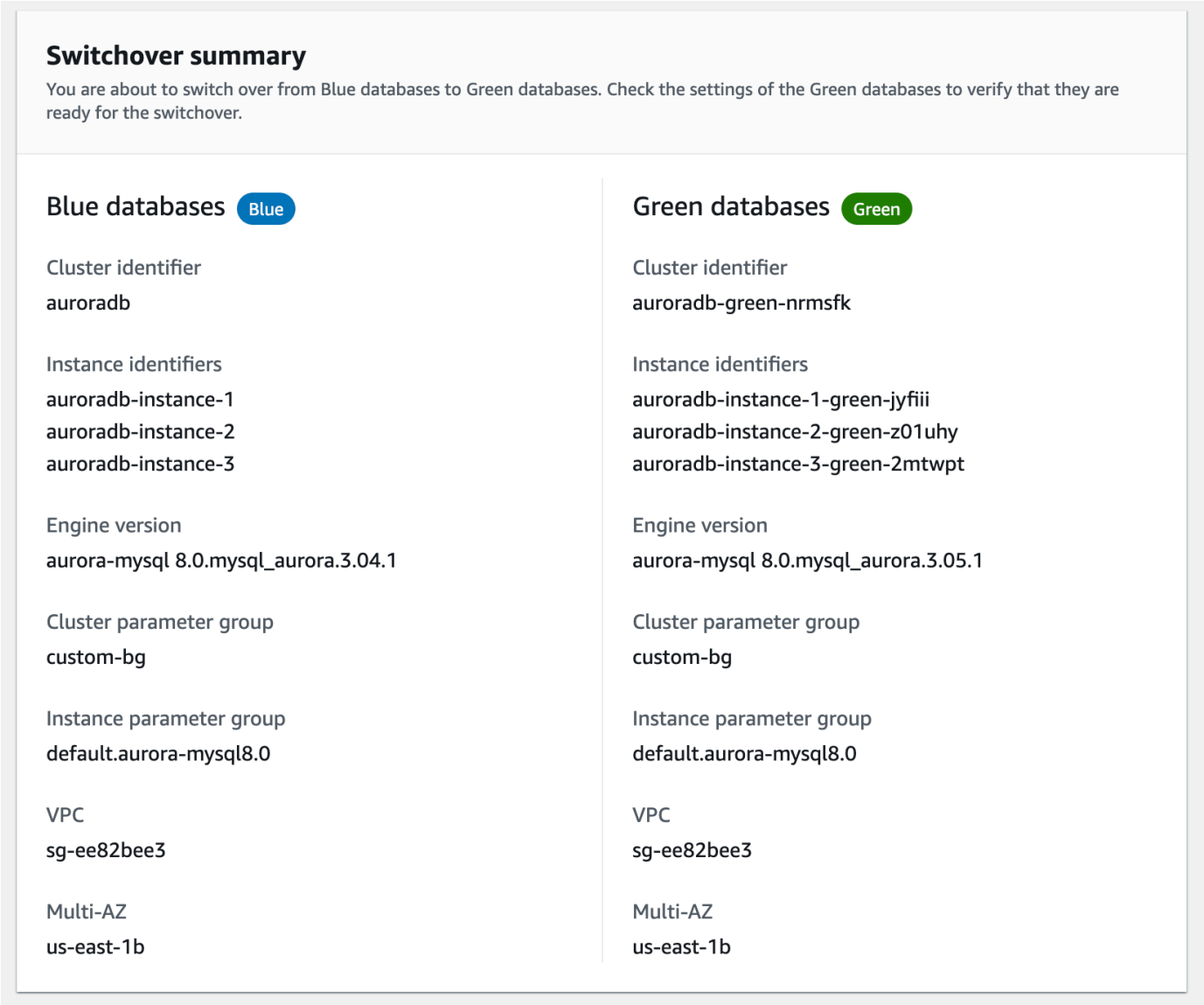
您可以使用 AWS Management Console、AWS CLI 或 RDS API 切换蓝绿部署。

控制台

切换蓝绿部署

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Databases（数据库），然后选择要切换的蓝绿部署。
3. 对于 Actions（操作），选择 Switch over（切换）。

   将出现 Switch over（切换）页面。



4. 在 Switch over（切换）页面上，查看切换摘要。确保两个环境中的资源都符合您的期望。如果不符合预期，请选择 Cancel（取消）。
5. 对于超时设置，输入切换的时间限制。
6. 如果您的集群运行 Aurora PostgreSQL，请查看并确认切换前建议。有关更多信息，请参阅 the section called “蓝绿部署的特定于逻辑复制的限制”。
7. 选择 Switch over（切换）。

AWS CLI

要使用 AWS CLI 切换蓝绿部署，请使用带有以下选项的 switchover-blue-green-deployment 命令：

- `--blue-green-deployment-identifier` – 指定蓝绿部署的资源 ID。

- `--switchover-timeout` – 指定切换的时间限制，以秒为单位。默认值为 300。

Example 切换蓝绿部署

对于 Linux、macOS 或 Unix：

```
aws rds switchover-blue-green-deployment \
    --blue-green-deployment-identifier bgd-1234567890abcdef \
    --switchover-timeout 600
```

对于 Windows：

```
aws rds switchover-blue-green-deployment ^
    --blue-green-deployment-identifier bgd-1234567890abcdef ^
    --switchover-timeout 600
```

RDS API

要使用 Amazon RDS API 切换蓝绿部署，请使用带有以下参数的 SwitchoverBlueGreenDeployment 操作：

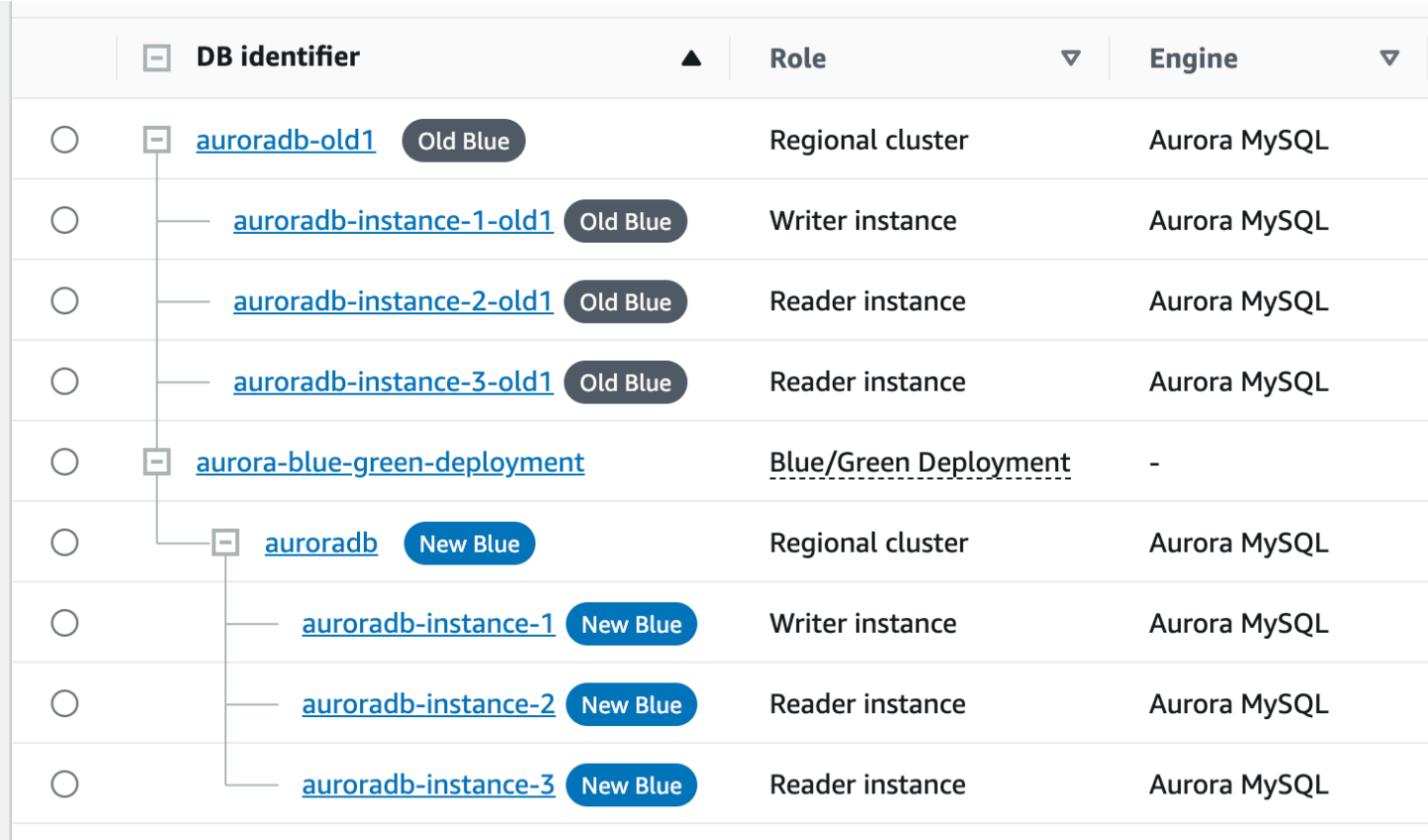
- `BlueGreenDeploymentIdentifier` – 指定蓝绿部署的资源 ID。
- `SwitchoverTimeout` – 指定切换的时间限制，以秒为单位。默认值为 300。

切换后

切换后，将保留先前蓝色环境中的数据库集群和数据库实例。标准成本适用于这些资源。蓝色和绿色环境之间的复制和二进制日志记录会停止。

RDS 通过在当前资源名称后面附加 `-old`*n*（其中 *n* 是一个数字）来重命名蓝色环境中的数据库集群和数据库实例。蓝色数据库集群被强制进入只读状态。即使您启用了写入操作，它仍为只读的，直到您删除蓝绿部署。RDS 会命名绿色环境 `-new`*n* 中的数据库集群和数据库实例。

如果您删除蓝绿部署资源，RDS 会保留 `-old`*n* 和 `-new`*n* 资源。



更新使用者的父节点

RDS 提供完全托管式只读副本。但是，它还提供用于设置自托管式副本（也称为外部副本）的选项。外部副本允许您使用第三方资源作为复制目标。

切换 Aurora MySQL 蓝绿部署后，如果蓝色数据库集群在切换之前有任何外部副本或二进制日志使用者，则必须在切换后更新其父节点以保持复制连续性。

更新父节点

1. 切换后，之前处于绿色环境中的写入器数据库实例会发出一个事件，其中包含主日志文件名和主日志位置。要找到该事件，请导航到 RDS 控制台，然后从左侧导航窗格中选择事件。
2. 在切换之前，按源为旧绿色写入器数据库实例名称的事件进行筛选。
3. 找到包含二进制日志坐标的事件。事件消息类似于：`Binary log coordinates in green environment after switchover: file mysql-bin-changelog.000003 and position 40134574`。
4. 确保使用者或副本已应用来自旧蓝色环境的所有二进制日志。然后，使用提供的二进制日志坐标在使用者上恢复复制。例如，如果您在 EC2 上运行 MySQL 副本，则可以使用下列命令：

MySQL 8.0.22 及更低的主要和次要版本

```
CHANGE MASTER TO MASTER_HOST='{new-writer-endpoint}', MASTER_LOG_FILE='mysql-bin-
changelog.000003', MASTER_LOG_POS=40134574;
```

MySQL 8.0.23 及更高的主要和次要版本

```
CHANGE REPLICATION SOURCE TO SOURCE_HOST='{new-writer-endpoint}',
 SOURCE_LOG_FILE='mysql-bin-changelog.000003', SOURCE_LOG_POS=40134574;
```

删除 Amazon Aurora 中的蓝绿部署

您可以在切换蓝绿部署之前或之后将其删除。

当您在切换蓝绿部署之前将其删除时，Amazon RDS 会在绿色环境中可选删除数据库集群：

- 如果您选择在绿色环境中删除数据库集群（`--delete-target`），则集群必须已关闭删除保护功能。
- 如果您未在绿色环境中删除数据库集群（`--no-delete-target`），则集群会被保留，但它不再是蓝绿部署的一部分。对于 Aurora MySQL，将继续在环境之间进行复制。对于 Aurora PostgreSQL，绿色环境将提升为独立环境，因此复制停止。

切换后，控制台中不提供用于删除绿色数据库的选项。使用 AWS CLI 删除蓝绿部署时，如果部署状态为 `SWITCHOVER_COMPLETED`，则无法指定 `--delete-target` 选项。

> **Important**
>
> 删除蓝绿部署后，RDS 会移除先前生产数据库集群的只读保护。如果对数据库集群禁用了 `read_only` 参数，它将再次开始支持写入操作。

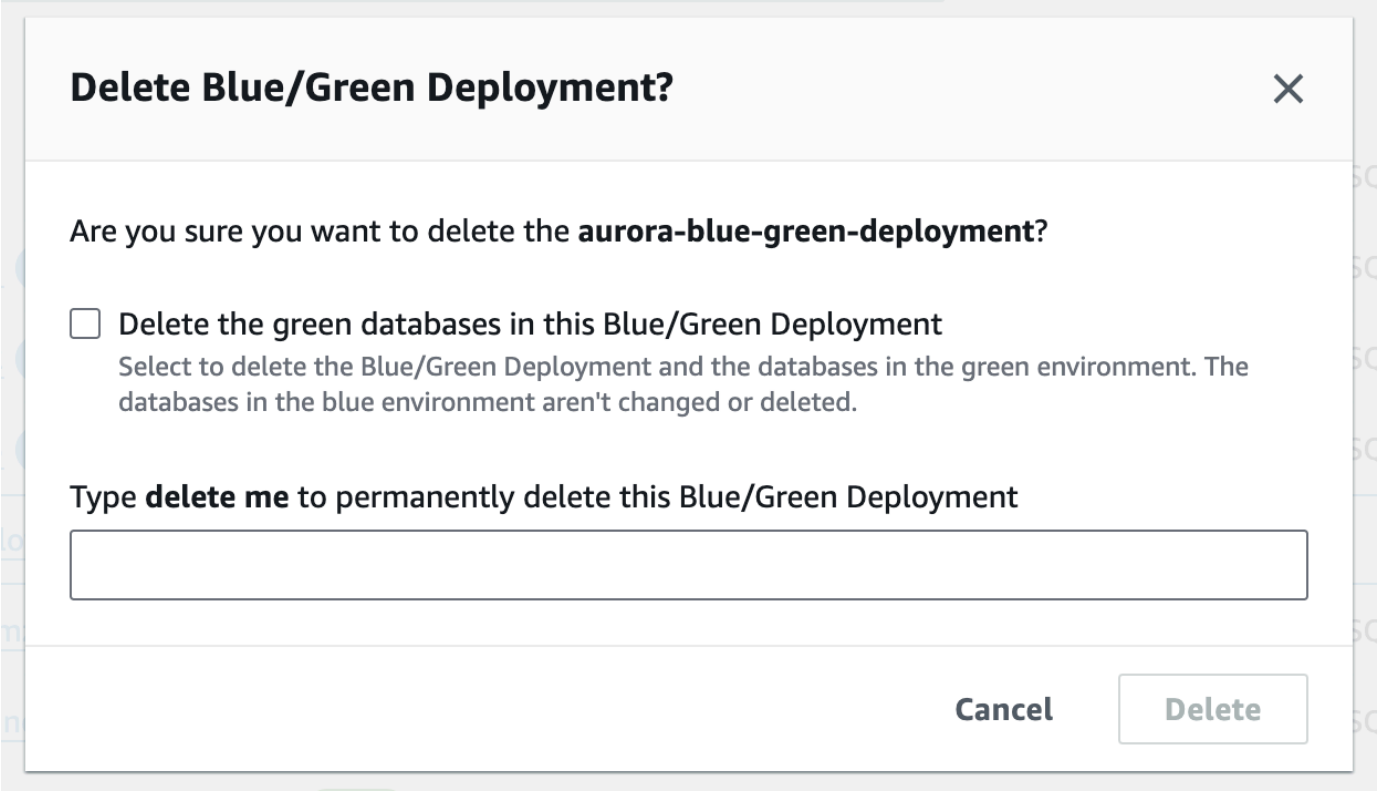
您可以使用 AWS Management Console、AWS CLI 或 RDS API 删除蓝绿部署。

控制台

删除蓝绿部署

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Databases（数据库），然后选择要删除的蓝绿部署。
3. 对于操作，选择删除。

   将显示是否删除蓝绿部署？窗口。



   要删除绿色数据库，请选择删除此蓝绿部署中的绿色数据库。
4. 在框中输入 **`delete me`**。
5. 选择删除。

AWS CLI

要使用 AWS CLI 删除蓝绿部署，请使用带有以下选项的 delete-blue-green-deployment 命令：

- `--blue-green-deployment-identifier` – 要删除的蓝绿部署的资源 ID。

- `--delete-target` – 指定删除绿色环境中的数据库集群。如果蓝绿部署的状态为 `SWITCHOVER_COMPLETED`，则无法指定此选项。
- `--no-delete-target` – 指定保留绿色环境中的数据库集群。

Example 删除绿色环境中的蓝绿部署和数据库集群

对于 Linux、macOS 或 Unix：

```
aws rds delete-blue-green-deployment \
    --blue-green-deployment-identifier bgd-1234567890abcdef \
    --delete-target
```

对于 Windows：

```
aws rds delete-blue-green-deployment ^
    --blue-green-deployment-identifier bgd-1234567890abcdef ^
    --delete-target
```

Example 删除蓝绿部署，但保留绿色环境中的数据库集群

对于 Linux、macOS 或 Unix：

```
aws rds delete-blue-green-deployment \
    --blue-green-deployment-identifier bgd-1234567890abcdef \
    --no-delete-target
```

对于 Windows：

```
aws rds delete-blue-green-deployment ^
    --blue-green-deployment-identifier bgd-1234567890abcdef ^
    --no-delete-target
```

RDS API

要使用 Amazon RDS API 删除蓝绿部署，请使用带有以下参数的 DeleteBlueGreenDeployment 操作：

- `BlueGreenDeploymentIdentifier` – 要删除的蓝绿部署的资源 ID。

- DeleteTarget – 指定 TRUE 以删除绿色环境中的数据库集群，或指定 FALSE 以保留它。如果蓝绿部署的状态为 SWITCHOVER_COMPLETED，则不能为 TRUE。

备份和还原 Amazon Aurora 数据库集群

这些主题提供有关备份和还原 Amazon Aurora 数据库集群的信息。

> **Tip**
>
> Aurora 高可用性功能和自动备份功能有助于确保您的数据安全，且无需您进行大量设置。在实施备份策略之前，请了解 Aurora 如何维护数据的多个副本，并帮助您跨多个数据库实例和AWS区域访问这些副本。有关详细信息，请参阅Amazon Aurora 的高可用性。

主题

- 备份和还原 Aurora 数据库集群的概述
- 保留自动备份
- 了解 Amazon Aurora 备份存储使用量
- 创建数据库集群快照
- 从数据库集群快照还原
- 数据库集群快照复制
- 共享数据库集群快照
- 将数据库集群数据导出到 Amazon S3
- 将数据库集群快照数据导出到 Amazon S3
- 将数据库集群还原到指定时间
- 删除数据库集群快照
- 教程：从数据库集群快照中还原 Amazon Aurora 数据库集群

备份和还原 Aurora 数据库集群的概述

以下主题介绍 Aurora 备份以及如何还原 Aurora 数据库集群。

目录

- 备份
  - 使用 AWS Backup
- 备份时段
- 还原数据
- 用于 Aurora 的数据库克隆
- 回溯

备份

Aurora 自动备份您的集群卷并将还原数据保留备份保留期的时长。Aurora 自动备份是连续和递增的，您可以快速还原到备份保留期内的任何时间点。在写入备份数据时，不会发生任何性能影响或数据库服务中断。在创建或修改数据库集群时，可指定备份保留期（1 天到 35 天）。Aurora 自动备份存储在 Amazon S3 中。有关保留自动备份的更多信息，请参阅保留自动备份

如果希望数据的保留期超出备份保留期，则可为集群卷中的数据创建快照。Aurora 数据库集群快照不会过期。您可以从该快照创建新的数据库集群。有关更多信息，请参阅 创建数据库集群快照。

> **Note**
>
> - 对于 Amazon Aurora 数据库集群，默认备份保留期为 1 天，不管创建数据库集群的方式如何。
> - 您无法在 Aurora 上禁用自动备份。Aurora 的备份保留期是由数据库集群管理的。

您的备份存储成本取决于您保存的 Aurora 备份和快照数据的数量以及保存时间。有关与 Aurora 备份和快照关联的存储的信息，请参阅了解 Amazon Aurora 备份存储使用量。有关 Aurora 备份存储的定价信息，请参阅 Amazon RDS for Aurora 定价。在删除与快照关联的 Aurora 集群后，存储该快照将产生 Aurora 的标准备份存储费用。

使用 AWS Backup

您可以使用 AWS Backup 来管理 Amazon Aurora 数据库集群的备份。

由 AWS Backup 管理的快照被视为手动数据库集群快照，但不计入 Aurora 的数据库集群快照限额。使用 AWS Backup 创建的快照的名称中具有 `awsbackup:job-AWS-Backup-job-number`。有关 AWS Backup 的更多信息，请参阅 AWS Backup 开发人员指南。

您也可以使用 AWS Backup 来管理 Amazon Aurora 数据库集群的自动备份。如果您的数据库集群与 AWS Backup 中的备份计划关联，则可以将该备份计划用于时间点恢复。由 AWS Backup 管理的自动（连续）备份的名称中具有 `continuous:cluster-AWS-Backup-job-number`。有关更多信息，请参阅 使用 AWS Backup 将数据库集群还原到指定时间。

备份时段

自动备份在每天的首选备份时段中进行。如果备份所需的时间超过了分配到备份时段的时间，则备份将在该时段结束后继续，直至完成。备份时段不能与数据库集群的每周维护时段重叠。

Aurora 自动备份是连续的增量备份，但备份时段用于创建在备份保留期内保留的每日系统备份。您可以复制此备份，以便在保留期之外保留。

> **Note**
>
> 使用 AWS Management Console 创建数据库集群时，无法指定备份时段。但是，您可以在使用 AWS CLI 或 RDS API 创建数据库集群时指定备份时段。

如果创建数据库集群时未指定首选备份时段，Aurora 将分配 30 分钟的默认备份时段。此时段是从每个 AWS 区域的 8 小时时间段中随机选择出来的。下表列出了已分配默认备份时段的各个 AWS 区域的时间块。

区域名称	区域	时间段
美国东部（弗吉尼亚州北部）	us-east-1	03:00–11:00 UTC
美国东部（俄亥俄州）	us-east-2	03:00–11:00 UTC
美国西部（加利福尼亚北部）	us-west-1	06:00–14:00 UTC

区域名称	区域	时间段
美国西部（俄勒冈州）	us-west-2	06:00–14:00 UTC
非洲（开普敦）	af-south-1	03:00–11:00 UTC
亚太地区（香港）	ap-east-1	06:00–14:00 UTC
亚太地区（海得拉巴）	ap-south-2	06:30–14:30 UTC
亚太地区（雅加达）	ap-southeast-3	08:00–16:00 UTC
亚太地区（马来西亚）	ap-southeast-5	09:00–17:00 UTC
亚太地区（墨尔本）	ap-southeast-4	11:00–19:00 UTC
亚太地区（孟买）	ap-south-1	16:30–00:30 UTC
Asia Pacific (Osaka)	ap-northeast-3	00:00–08:00 UTC
Asia Pacific (Seoul)	ap-northeast-2	13:00–21:00 UTC
亚太地区（新加坡）	ap-southeast-1	14:00–22:00 UTC
亚太地区（悉尼）	ap-southeast-2	12:00–20:00 UTC
亚太地区（东京)	ap-northeast-1	13:00–21:00 UTC
加拿大（中部）	ca-central-1	03:00–11:00 UTC
加拿大西部（卡尔加里）	ca-west-1	18:00–02:00 UTC
中国（北京）	cn-north-1	06:00–14:00 UTC
中国（宁夏）	cn-northwest-1	06:00–14:00 UTC
Europe (Frankfurt)	eu-central-1	20:00–04:00 UTC

区域名称	区域	时间段
欧洲（爱尔兰）	eu-west-1	22:00–06:00 UTC
欧洲地区（伦敦）	eu-west-2	22:00–06:00 UTC
欧洲（米兰）	eu-south-1	02:00–10:00 UTC
欧洲（巴黎）	eu-west-3	07:29–14:29 UTC
欧洲（西班牙）	eu-south-2	02:00–10:00 UTC
Europe (Stockholm)	eu-north-1	23:00–07:00 UTC
欧洲（苏黎世）	eu-central-2	02:00–10:00 UTC
以色列（特拉维夫）	il-central-1	03:00–11:00 UTC
中东（巴林）	me-south-1	06:00–14:00 UTC
中东（阿联酋）	me-central-1	05:00–13:00 UTC
南美洲（圣保罗）	sa-east-1	23:00–07:00 UTC
AWS GovCloud（美国东部）	us-gov-east-1	17:00–01:00 UTC
AWS GovCloud（美国西部）	us-gov-west-1	06:00–14:00 UTC

还原数据

您可以从 Aurora 保留的备份数据、您保存的数据库集群快照或保留的自动备份创建新的 Aurora 数据库集群以恢复数据。您可以将从备份数据创建的新数据库集群副本快速还原到备份保留期内的任何时间点。由于 Aurora 备份在备份保留期内是连续和增量备份，因此，您无需为了缩短还原时间而频繁创建数据快照。

数据库集群的最近可还原时间是可以还原数据库集群的最近时间点。对于活动的数据库集群，这通常在当前时间的 5 分钟之内，或者在保留自动备份的集群删除时间的 5 分钟之内。

最早可还原时间指定可将集群卷还原到的备份保留期内的最早时间点。

要确定数据库集群的最近或最早的可还原时间，请在 RDS 控制台上查找 `Latest restorable time` 或 `Earliest restorable time` 值。有关查看这些值的信息，请参阅查看保留的备份。

您可以通过检查 `Latest restorable time` 和 `Earliest restorable time` 值来确定数据库集群还原完成的时间。在还原操作完成之前，这些值将返回 NULL。如果 `Latest restorable time` 或 `Earliest restorable time` 返回 NULL，则无法请求备份或还原操作。

有关将数据库集群还原到指定时间的信息，请参阅将数据库集群还原到指定时间。

用于 Aurora 的数据库克隆

您还可以使用数据库克隆将 Aurora 数据库集群的数据库克隆到新的数据库集群，而不是还原数据库集群快照。克隆数据库仅在首次创建时使用很少的额外空间。仅当源数据库上或克隆数据库上的数据发生变化时才复制数据。您可以从同一个数据库集群中进行多次克隆，甚至可以为其他克隆创建额外的克隆。有关更多信息，请参阅 克隆 Amazon Aurora 数据库集群卷。

回溯

Aurora MySQL 现在支持将数据库集群“倒回”到特定时间，而无需从备份还原数据。有关更多信息，请参阅“回溯 Aurora 数据库集群”。

保留自动备份

当您删除预调配或 Aurora Serverless v2 数据库集群时，您可以保留自动备份。这允许您将数据库集群还原到备份保留期内的特定时间点，即使在集群被删除之后也是如此。

保留的自动备份包含系统快照和来自数据库集群的事务日志。它们还包括数据库集群属性，例如数据库实例类，这些属性是将其还原到活动集群所必需的。

您可以使用 AWS Management Console、RDS API 和 AWS CLI 还原或移除保留的自动备份。

> **Note**
>
> 您不能保留 Aurora Serverless v1 数据库集群的自动备份。

主题

- 保留期
- 查看保留的备份
- 保留成本
- 限制
- 删除保留的自动备份

保留期

保留的自动备份中的系统快照和事务日志与源数据库集群的系统快照和事务日志以同样的方式过期。源集群的保留期设置也适用于自动备份。因为没有为此集群创建任何新的快照或日志，所以保留的自动备份最终将完全过期。保留期结束后，您可以继续保留手动数据库集群快照，但所有自动备份都会过期。

您可以使用控制台、AWS CLI 或 RDS API 操作移除保留的自动备份。有关更多信息，请参阅 删除保留的自动备份。

与保留的自动备份不同，最终快照不会过期。我们强烈建议您制作最终快照，即使您保留自动备份也是如此，因为保留的自动备份最终将过期。

查看保留的备份

要在 RDS 控制台中查看保留的自动备份，请在导航窗格中选择自动备份，然后选择保留。要查看与保留的自动备份关联的单个快照，请在导航窗格中选择 Snapshots (快照)。或者，您可以通过描述来查看与保留的自动备份关联的各个快照。然后，您可以直接从这些快照之一还原数据库实例。

要使用 AWS CLI 对保留的自动备份进行描述，请使用以下命令：

```
aws rds describe-db-cluster-automated-backups --db-cluster-resource-
id DB_cluster_resource_ID
```

要使用 RDS API 对保留的自动备份进行描述，请使用 DbClusterResourceId 参数调用 DescribeDBClusterAutomatedBackups 操作。

保留成本

对于每个 Aurora 数据库集群，备份存储最高可达总 Aurora 数据库存储的 100%，不收取额外费用。如果您在删除数据库集群后保留自动备份，也会有多达一天时间不收取额外费用。保留超过一天的备份需要付费。

事务日志或实例元数据没有额外费用。备份的所有其他定价规则适用于可还原的集群。有关更多信息，请参阅 Amazon Aurora 定价页面。

限制

以下限制适用于保留的自动备份：

- 一个 AWS 区域中保留的自动备份的最大数量为 40。它不包含在数据库集群的限额中。您最多可以同时保留 40 个正在运行的数据库集群、40 个正在运行的数据库实例和 40 个为数据库集群保留的自动备份。

  有关更多信息，请参阅 Amazon Aurora 中的配额。
- 保留的自动备份不包含有关参数或选项组的信息。
- 您可以将已删除的集群还原到删除时保留期内的某个时间点。
- 您无法修改保留的自动备份，因为它由系统备份、事务日志和删除源集群时存在的数据库集群属性组成。

删除保留的自动备份

当不再需要保留的自动备份时，可以删除它们。

控制台

删除保留的自动备份

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Automated backups（自动备份）。
3. 选择已保留选项卡。



4. 选择要删除的保留自动备份。
5. 对于 Actions (操作)，选择 Delete (删除)。
6. 在确认页面上，输入 **`delete me`** 并选择 Delete (删除)。

AWS CLI

可以通过使用 AWS CLI 命令 delete-db-cluster-automated-backup 及以下选项删除保留的自动备份：

- `--db-cluster-resource-id` – 源数据库集群的资源标识符。

  可以通过运行 AWS CLI 命令 describe-db-cluster-automated-backups 查找保留自动备份的源数据库集群的资源标识符。

Example

此示例删除资源 ID 为 `cluster-123ABCEXAMPLE` 的源数据库集群的保留自动备份。

对于 Linux、macOS 或 Unix：

```
aws rds delete-db-cluster-automated-backup \
```

```
    --db-cluster-resource-id cluster-123ABCEXAMPLE
```

对于 Windows：

```
aws rds delete-db-cluster-automated-backup ^
    --db-cluster-resource-id cluster-123ABCEXAMPLE
```

RDS API

可以通过使用 Amazon RDS API 操作 DeleteDBClusterAutomatedBackup 及以下参数删除保留的自动备份。

- `DbClusterResourceId` – 源数据库集群的资源标识符。

  可以通过使用 Amazon RDS API 操作 DescribeDBClusterAutomatedBackups 查找保留自动备份的源数据库实例的资源标识符。

了解 Amazon Aurora 备份存储使用量

Amazon Aurora 维护两种类型的备份：自动（连续）备份和快照。

自动备份存储

集群的自动（连续）备份以增量方式存储指定保留期内的所有数据库更改，以便能够还原到该保留期内的任何时间点。保留期可以介于 1 到 35 天之间。自动备份是增量备份，根据还原到保留期内的任何时间所需的存储量进行收费。

Aurora 还提供免费备份使用量。此免费使用量等于最新的集群卷大小（由 `VolumeBytesUsed` Amazon CloudWatch 指标表示）。此使用量从计算出的自动备份使用量中扣除。对于保留期仅为 1 天的自动备份，也不会收取任何费用。

例如，您的自动备份的保留期为 7 天，您想要将集群还原到四天前的状态。Aurora 使用存储在自动备份中的增量数据来重新创建四天前这个具体时间的集群状态。

自动备份存储了所有必需的信息，以便能够在保留时段中的任何时间点还原集群。这意味着它会存储保留时段内的所有更改，包括写入新信息或删除现有信息。对于发生许多更改的数据库，自动备份的大小会逐渐增加。数据库停止发生更改后，由于先前存储的更改会退出保留时段，因此您可以预期自动备份的大小会减小。

自动备份的总计费使用量绝不会超过保留期内的累积集群卷大小。例如，如果您的保留期为 7 天，集群卷为每天 100 GB，则计费的自动备份使用量绝不会超过 700 GB（100 GB * 7）。

快照存储

数据库集群快照始终是完整备份，这些备份会在您创建它们时捕获集群卷的大小。无论您是手动还是通过 AWS Backups 计划拍摄快照，Aurora 都会将其视为手动快照。Aurora 为自动备份保留期内的快照提供不受限的免费存储空间。手动快照超出此期限后，将按 GB-月计费。自动系统快照将保持免费，除非您复制它们。由于自动备份不涵盖快照副本，因此 AWS 始终会对其收费。

有关 Aurora 备份的一般信息，请参阅备份。有关 Aurora 备份存储定价的信息，请参阅 Amazon Aurora 定价页面。

Aurora 备份存储的 Amazon CloudWatch 指标

您可以通过 CloudWatch 控制台，使用 Amazon CloudWatch 指标来监控 Aurora 集群并创建报告。您可以使用以下 CloudWatch 指标来检查和监控 Aurora 备份使用的存储量。这些指标是为每个 Aurora 数据库集群单独计算的。

- BackupRetentionPeriodStorageUsed – 表示目前用于存储自动备份的备份存储量（以字节为单位）。
  - 该值取决于集群卷的大小以及在保留期内对数据库集群的更改（写入和更新）次数。这是因为自动备份必须存储对集群所做的所有增量更改，才能还原到任何时间点。
  - 此指标并未减去 Aurora 提供的免费备份使用套餐。
  - 该指标针对当天记录的自动备份使用量发出一个每日数据点。
- SnapshotStorageUsed – 表示用于存储超出自动备份保留期的手动快照的备份存储量（以字节为单位）。
  - 该值取决于您在自动备份的保留期之外保留的快照数量以及每个快照的大小。
  - 每个快照的大小是您拍摄快照时的集群卷的大小。
  - 快照是完整备份，不是增量备份。
  - 该指标针对每个收费的快照发出一个每日数据点。要检索您每日的总体快照使用量，请计算一天内该指标的总和。
- TotalBackupStorageBilled – 表示给定集群的所有计费备份使用量的指标（以字节为单位）：

  ```
  BackupRetentionPeriodStorageUsed + SnapshotStorageUsed - free tier
  ```

  - 此指标针对 BackupRetentionPeriodStorageUsed 值减去 Aurora 提供的免费备份使用套餐，发出一个每日数据点。此免费套餐等于数据库集群卷的最新记录大小。此数据点表示自动备份的实际计费使用量。
  - 该指标将针对所有 SnapshotStorageUsed 值发出单独的每日数据点。
  - 要检索您每日的总体计费备份使用量，请计算一天内该指标的总和。这将所有计费的快照使用量与计费的自动备份使用量相加，得出您总的计费备份使用量。

有关如何使用 CloudWatch 指标的更多信息，请参阅 Amazon RDS 控制台中 Aurora 指标的可用性。

计算备份存储使用量

自动备份的使用量是通过查看必须存储的所有增量记录来计算的，存储这些增量记录的目的是为了能够还原到备份保留期内的任意时间点。这些更改不仅包括写入操作的数量，还包括数据修改的大小和范围。每种类型的操作（INSERT、UPDATE、DELETE）都会创建更改记录，必须保留这些记录才能进行时间点故障恢复。因此，尽管两个数据库可能具有相同数量的写入操作（IOPS），但它们的备份存储要求可能会因每个事务中所修改的数据量而大相径庭。

例如，您有一个保留期为 7 天的自动备份。保留期之前的集群卷大小为 100 GB，因此这是 Aurora 需要存储的最小容量。然后，您将在接下来的 7 天内进行以下活动，其中增量记录大小是存储数据库写入和更新产生的更改记录所需的存储量。

天	增量记录大小（GB）
1	10
2	15
3	25
4	20
5	10
6	25
7	30
总计	135

此数据意味着计算出的备份的自动备份使用量如下：

```
100 GB (volume size before retention period) + 135 GB (size of incremental records) =
 235 GB total backup usage
```

然后，计费的使用量减去免费使用量套餐。假设您的卷的最新大小为 200 GB：

```
235 GB total backup usage - 200 GB (latest volume size) = 35 GB billed backup usage
```

常见问题解答

什么时候会向我收取快照费用？

对于超出（早于）自动备份保留期的手动快照，将向您收取费用。

什么是手动快照？

手动快照是指适用以下条件之一的快照：

- 由您手动请求
- 由自动备份服务（例如 AWS Backup）拍摄
- 从自动系统快照中复制，在保留期之外进行保存

如果我删除数据库集群，我的手动快照会怎样？

手动快照在删除之前不会过期。

当您删除数据库集群时，您之前拍摄的手动快照将继续存在。如果这些快照以前因为处于自动备份保留期内而不被计费，那么现在它们已不在保留期内，所有快照都开始按其整个使用量计费。

如何降低备份存储成本？

有几种方法可以减少备份使用量相关成本：

- 删除超出自动备份保留期的手动快照。其中包括您拍摄的快照，以及您的 AWS Backup 计划可能会拍摄的快照。请务必检查您的 AWS Backup 计划，确保其不会在出乎您意料的保留期之外保留快照。
- 评估您对数据库的写入和更新，看看是否可以减少更改次数。由于我们的自动备份会存储保留期内的所有增量更改，因此减少更新次数也可以减少自动备份费用。
- 评估缩短自动备份保留期是否合理。缩短保留期意味着备份存储增量数据的天数更少，从而可以降低总体备份成本。但是，缩短保留期也可能会导致某些快照开始计费，因为这些快照现在已超出保留期。在决定缩短保留期是否为正确做法之前，请务必检查可能产生的所有额外快照费用。

备份存储是如何计费的？

备份存储按 GB/月计费。

这意味着备份存储使用量按给定月份使用量的加权平均值计费。以下是总共 30 天的某个月份的几个示例：

- 当月 30 天的计费备份使用量均为 100 GB。您的费用如下：

```
(100 GB * 30) / 30 = 100 GB-month
```

- 当月前 15 天的计费备份使用量为 100 GB，后 15 天的计费备份使用量为 0 GB。您的费用如下：

```
(100 GB * 15 + 0 GB * 15) / 30 = 50 GB-month
```

- 当月前 10 天的计费备份使用量为 50 GB，接下来的 10 天为 100 GB，最后 10 天为 150 GB。您的费用如下：

```
(50 GB * 10 + 100 GB * 10 + 150 GB * 10) / 30 = 100 GB-month
```

我的数据库集群的回溯设置对备份存储使用量有何影响？

Aurora 数据库集群的回溯设置不会影响该集群的备份数据量。Amazon 单独对回溯数据存储收费。有关 Aurora 回溯的定价信息，请参阅 Amazon Aurora 定价页面。

共享快照的存储成本如何收费？

如果您与另一个用户共享快照，您仍然是该快照的拥有者。存储成本向快照拥有者收取。如果您删除了所拥有的共享快照，没有用户能够再访问它。

要继续访问其他人所有的共享快照，您可以复制该快照。这样会让您成为新快照的拥有者。所复制快照的所有存储成本都会计入您的账户。

有关共享快照的更多信息，请参阅共享数据库集群快照。有关复制快照的更多信息，请参阅数据库集群快照复制。

创建数据库集群快照

Amazon RDS 创建数据库集群的存储卷快照，并备份整个数据库集群而不仅仅是单个数据库。创建数据库集群快照时，您需要标识要备份的数据库集群，然后为数据库集群快照命名，以便稍后从此快照还原。创建数据库集群快照所用时间因数据库大小而异。由于快照包含整个存储卷，因此，文件（如临时文件）的大小也会影响创建快照所需的时间。

> **Note**
>
> 您的数据库集群必须处于 `available` 状态才能拍摄数据库集群快照。

与自动备份不同，手动快照不受备份保留期的限制。快照不会过期。

对于非常长期的备份，我们建议将快照数据导出到 Amazon S3。如果不再支持数据库引擎的主要版本，则无法从快照还原到该版本。有关更多信息，请参阅 将数据库集群快照数据导出到 Amazon S3。

您可以使用 AWS Management Console、AWS CLI 或 RDS API 创建数据库集群快照。

控制台

创建数据库集群快照

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择快照。

   将显示手动快照列表。
3. 选择拍摄快照。

   将显示 Take DB snapshot (拍摄数据库快照) 窗口。
4. 对于快照类型，请选择数据库集群。
5. 选择要为其拍摄快照的数据库集群。
6. 输入快照名称。
7. 选择拍摄快照。

   将显示手动快照列表，其中新的数据库集群快照的状态显示为 `Creating`。在其状态为 `Available` 后，您可以看到其创建时间。

AWS CLI

在使用 AWS CLI 创建数据库集群快照时，您需要指定要备份的数据库集群，然后指定数据库集群快照名称，以便以后从该快照还原。结合以下参数使用 AWS CLI `create-db-cluster-snapshot` 命令执行该操作：

- `--db-cluster-identifier`
- `--db-cluster-snapshot-identifier`

在该示例中，您为名为 *mydbcluster* 的数据库集群创建名为 *mydbclustersnapshot* 的数据库集群快照。

Example

对于 Linux、macOS 或 Unix：

```
aws rds create-db-cluster-snapshot \
    --db-cluster-identifier mydbcluster \
    --db-cluster-snapshot-identifier mydbclustersnapshot
```

对于 Windows：

```
aws rds create-db-cluster-snapshot ^
    --db-cluster-identifier mydbcluster ^
    --db-cluster-snapshot-identifier mydbclustersnapshot
```

RDS API

在使用 Amazon RDS API 创建数据库集群快照时，您需要指定要备份的数据库集群，然后指定数据库集群快照名称，以便以后从该快照还原。您可以使用具有以下参数的 Amazon RDS API `CreateDBClusterSnapshot` 命令执行该操作：

- DBClusterIdentifier
- DBClusterSnapshotIdentifier

确定数据库集群快照是否可用

您可通过下列方式确定数据库集群快照是否可用：在 AWS Management Console 的集群详情页面上查看 Maintenance & backups（维护和备份）选项卡下的 Snapshots（快照）；或者使用 `describe-db-cluster-snapshots` CLI 命令或 `DescribeDBClusterSnapshots` API 操作。

您也可以使用 `wait db-cluster-snapshot-available` CLI 命令每 30 秒轮询一次 API，直到快照可用。

从数据库集群快照还原

Amazon RDS 创建数据库集群的存储卷快照，并备份整个数据库集群而不仅仅是单个数据库。您可以通过从数据库快照还原来创建新的数据库集群。您可以提供用于还原的数据库集群快照的名称，然后提供还原后新建的数据库集群的名称。您无法从数据库集群快照还原到现有数据库集群；还原时将新建一个数据库集群。

> **Important**
>
> 如果您尝试将快照还原到已弃用的数据库引擎版本，则会立即升级到最新的引擎版本。从已弃用的引擎版本恢复的数据库集群在升级到更新的主要版本之前将无法访问。
> 此外，如果版本处于扩展支持状态或已到达标准支持终止日期，则可能会收取扩展支持费用。
> 有关更多信息，请参阅Amazon Aurora 的 Amazon RDS 扩展支持。

您可以使用已还原的数据库集群，只要其状态为 `available`。

您可以使用 AWS CloudFormation 从数据库集群快照中还原数据库集群。有关更多信息，请参阅《AWS CloudFormation 用户指南》中的 AWS::RDS::DBCluster。

> **Note**
>
> 共享手动数据库集群快照（无论是否加密）可允许经授权的 AWS 账户直接从快照还原数据库集群，无需复制数据库集群再从中进行还原。有关更多信息，请参阅共享数据库集群快照。

有关使用 RDS Extended Support 版本还原 Aurora 数据库集群或全局集群的信息，请参阅使用 Amazon RDS 扩展支持还原Aurora 数据集群或全局集群。

参数组注意事项

建议您保留所创建的任何数据库集群快照的数据库参数组和数据库集群参数组，以便还原的数据库集群可以与正确的参数组关联。

除非选择不同的实例，否则默认数据库参数组和数据库集群参数组将与还原的实例关联。默认参数组中没有可用的自定义参数设置。

您可以在还原数据库集群时指定参数组。

有关数据库参数组和数据库集群参数组的更多信息，请参阅Amazon Aurora 的参数组。

安全组注意事项

还原数据库时集群时，除非选择不同的值，否则默认的虚拟私有云（VPC）、数据库子网组和 VPC 安全组将与还原的实例关联。

- 如果您使用 Amazon RDS 控制台，则可以指定要与集群关联的自定义 VPC 安全组，或者创建新的 VPC 安全组。
- 如果您使用的是 AWS CLI，则可以在 `restore-db-cluster-from-snapshot` 命令中包括 `--vpc-security-group-ids` 选项，从而指定要与集群关联的自定义 VPC 安全组。
- 如果您使用的是 Amazon RDS API，则可以在 `VpcSecurityGroupIds.VpcSecurityGroupId.N` 操作中包括 `RestoreDBClusterFromSnapshot` 参数。

一旦完成还原，并且您的新数据库集群可用，您还可以通过修改数据库集群来更改 VPC 设置。有关更多信息，请参阅修改 Amazon Aurora 数据库集群。

Amazon Aurora 注意事项

利用 Aurora，可将数据库集群快照还原为数据库集群。

利用 Aurora MySQL 和 Aurora PostgreSQL，也可以将数据库集群快照还原为 Aurora Serverless 数据库集群。有关更多信息，请参阅还原 Aurora Serverless v1 数据库集群。

利用 Aurora MySQL，可以将数据库集群快照从没有并行查询的集群恢复到具有并行查询的集群。因为并行查询通常用于非常大型的表，所以快照机制是向支持 Aurora MySQL 并行查询的集群引入大量数据的最快方式。有关更多信息，请参阅Amazon Aurora MySQL 的并行查询。

从快照还原

您可以使用 AWS Management Console、AWS CLI 或 RDS API 从数据库集群快照还原数据库集群。

控制台

从数据库集群快照还原数据库集群

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。

2. 在导航窗格中，选择快照。
3. 选择要从其还原的数据库集群快照。
4. 对于操作，选择还原快照。

   将显示还原快照页面。
5. 选择您要将数据库集群还原到的数据库引擎版本。

   默认情况下，快照还原到与源数据库集群相同的数据库引擎版本（如果该版本可用）。
6. 对于 DB instance identifier（数据库实例标识符），请输入还原后的数据库实例的名称。请注意，Amazon RDS 从您指定的数据库实例标识符派生数据库集群标识符。
7. 指定其他设置，如数据库集群存储配置。

   有关每项设置的信息，请参阅 Aurora 数据库集群的设置。
8. 选择 Restore DB cluster（还原数据库集群）。

AWS CLI

要从数据库集群快照还原数据库集群实例，请使用 AWS CLI 命令 restore-db-cluster-from-snapshot。

在该示例中，您将从先前创建的名为 `mydbclustersnapshot` 的数据库集群快照中进行还原。还原为名为 `mynewdbcluster` 的新数据库集群。

您可以指定其它设置，例如数据库引擎版本。如果您未指定引擎版本，则数据库集群将还原到默认引擎版本。

有关每项设置的信息，请参阅 Aurora 数据库集群的设置。

Example

对于 Linux、macOS 或 Unix：

```
aws rds restore-db-cluster-from-snapshot \
    --db-cluster-identifier mynewdbcluster \
    --snapshot-identifier mydbclustersnapshot \
    --engine aurora-mysql|aurora-postgresql
```

对于 Windows：

```
aws rds restore-db-cluster-from-snapshot ^
    --db-cluster-identifier mynewdbcluster ^
    --snapshot-identifier mydbclustersnapshot ^
    --engine aurora-mysql|aurora-postgresql
```

在还原数据库集群后，如果您需要具有与以前的数据库集群相同的功能，则必须将数据库集群添加到用于创建数据库快照的数据库集群使用的安全组中。

> **Important**
>
> 如果您使用控制台还原数据库集群，则 Amazon RDS 为您的数据库集群自动创建主数据库实例（写入器）。如果您使用 AWS CLI 还原数据库集群，则必须明确为数据库集群创建主实例。主实例是在数据库集群中创建的第一个实例。如果您未创建主数据库实例，则数据库集群端点将保持 `creating` 状态。
>
> 调用 create-db-instance AWS CLI 命令，以便为数据库集群创建主实例。包括数据库集群名称以作为 `--db-cluster-identifier` 选项值。

RDS API

要从数据库集群快照还原数据库集群，请使用以下参数调用 RDS API 操作 RestoreDBClusterFromSnapshot：

- `DBClusterIdentifier`
- `SnapshotIdentifier`

> **Important**
>
> 如果您使用控制台还原数据库集群，则 Amazon RDS 为您的数据库集群自动创建主数据库实例（写入器）。如果您使用 RDS API 还原数据库集群，则必须明确为数据库集群创建主实例。主实例是在数据库集群中创建的第一个实例。如果您未创建主数据库实例，则数据库集群端点将保持 `creating` 状态。
>
> 调用 RDS API 操作 CreateDBInstance 以创建数据库集群的主实例。包括数据库集群的名称作为 `DBClusterIdentifier` 参数值。

数据库集群快照复制

使用 Amazon Aurora，您可以复制自动或手动数据库集群快照。在复制快照后，该副本为手动快照。可以创建自动备份或手动快照的多个副本，但每个副本必须具有唯一的标识符。

您可以复制位于同一 AWS 区域中的快照，可以跨 AWS 区域复制快照，也可以复制共享快照。

您不能通过一个步骤跨区域和账户复制数据库集群快照。每个这些复制操作都需要执行一个步骤。作为对复制的替代，您也可与其他 AWS 账户共享手动快照。有关更多信息，请参阅 共享数据库集群快照。

> **Note**
>
> Amazon 根据您保留的 Amazon Aurora 备份和快照数据量以及您保留的时间对您进行收费。有关与 Aurora 备份和快照关联的存储的信息，请参阅了解 Amazon Aurora 备份存储使用量。有关 Aurora 存储的定价信息，请参阅 Amazon RDS for Aurora 定价。

查看数据库集群快照复制的限制和注意事项。要复制数据库集群快照，请参阅以下主题之一。

- 使用 AWS Management Console复制数据库集群快照
- 使用 AWS CLI 或 Amazon RDS API 复制未加密的数据库集群快照
- 使用 AWS CLI 或 Amazon RDS API 复制加密的数据库集群快照
- 跨账户复制数据库集群快照

使用 AWS Management Console复制数据库集群快照

使用此主题中的过程复制数据库集群快照。如果源数据库引擎为 Aurora，则您的快照是数据库集群快照。

对于每个 AWS 账户，一次最多可以从一个 AWS 区域向另一个区域复制五个数据库集群快照。支持复制加密和未加密的数据库集群快照。如果您将数据库集群快照复制到另一个 AWS 区域，则可创建保留在该 AWS 区域中的手动数据库集群快照。从源 AWS 区域复制出数据库集群快照会产生 Amazon RDS 数据传输费用。

有关数据传输定价的更多信息，请参阅 Amazon RDS 定价。

在新 AWS 区域中创建数据库集群快照副本后，该数据库集群快照副本的行为与该 AWS 区域中所有其他数据库集群快照的行为相同。

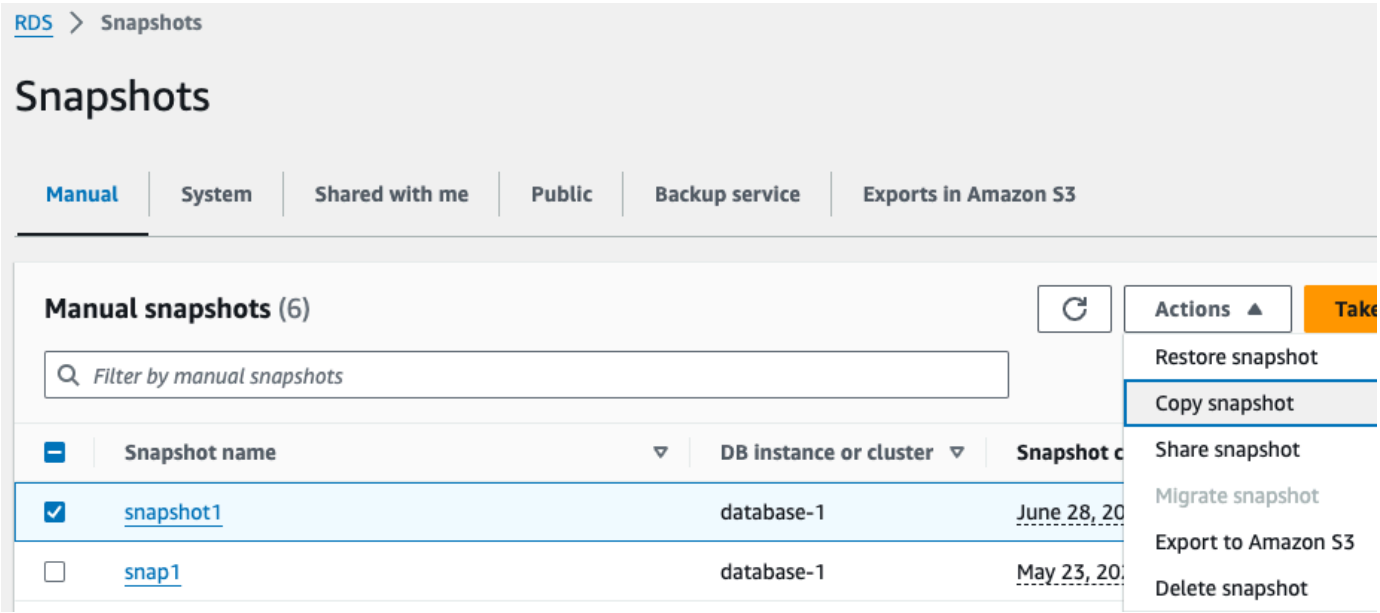
此程序可用于在同一 AWS 区域或跨区域复制加密和未加密的数据库集群快照。

要在正在进行复制时取消操作，请在数据库集群快照处于正在复制状态时删除目标数据库集群快照。

在复制数据库集群快照之前，请查看限制和快照复制注意事项。

复制数据库集群快照

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择快照。
3. 选择要复制的数据库集群快照。
4. 选择操作，然后选择复制快照。



5. （可选）要将数据库集群快照复制到其他 AWS 区域，请为 Destination Region（目标区域）选择该 AWS 区域。
6. 在 New DB Snapshot Identifier (新数据库实例标识符) 中输入数据库集群快照副本的名称。
7. 要将标签和值从快照复制到快照的副本，请选择 Copy Tags (复制标签)。
8. 选择 Copy Snapshot (复制快照)。

限制

复制快照时，存在以下一些限制：

- 您不能向或者从以下 AWS 区域复制快照：

  - 中国（北京）
  - 中国（宁夏）
- 您可以在 AWS GovCloud（美国东部）和 AWS GovCloud（美国西部）之间复制快照。但是，您不能在这些 AWS GovCloud (US) 区域和商业 AWS 区域之间复制快照。
- 如果您在目标快照可用之前删除了源快照，则快照复制会失败。在删除源快照之前，请确保目标快照的状态为 AVAILABLE。
- 每个账户最多可以同时进行到同一目标区域的五个快照复制请求。
- 当您为同一源数据库实例请求多个快照副本时，它们将在内部排队。在先前的快照副本完成之后，稍后请求的副本才会启动。有关更多信息，请参阅 AWS 知识中心里的为什么我的 EC2 AMI 或 EBS 快照创建速度很慢？。
- 根据所涉及的 AWS 区域和要复制的数据量，可能需要数小时才能完成跨区域快照复制。有时，某一给定的源区域可能会发出大量跨区域快照复制请求。在这种情况下，Amazon RDS 可能会将来自该源区域的新跨区域复制请求排入队列，直至某些区域正在进行的复制完成。当复制请求在队列中时，不显示有关这些复制请求的进度信息。复制开始后即显示进度信息。
- Aurora 不支持增量快照。Aurora 数据库集群快照副本始终存储为完整副本。完整快照副本包含还原数据库集群需要的所有数据和元数据。

快照复制注意事项

以下是复制快照时的注意事项。

主题

- 共享快照复制注意事项
- 加密的数据库集群快照复制注意事项
- 跨区域快照复制注意事项
- 参数组注意事项

共享快照复制注意事项

您可以复制其他 AWS 账户与您共享的快照。在某些情况下，您可以从另一个 AWS 账户复制共享的加密快照。在这些情况下，您必须有权访问用于加密快照的 AWS KMS key。

无论加密与否，都只能在相同 AWS 区域中复制共享的数据库集群快照。有关更多信息，请参阅 共享加密的快照。

加密的数据库集群快照复制注意事项

您可以复制已使用 KMS 密钥加密的快照。如果您复制加密的快照，则此快照的副本也必须加密。如果在同一 AWS 区域内复制加密的快照，可使用与原始快照相同的 KMS 密钥加密此副本。也可指定不同的 KMS 密钥。

如果跨区域复制加密快照，则必须指定在目标 AWS 区域中有效的 KMS 密钥。该密钥可以是某个区域的专用 KMS 密钥，也可以是多区域密钥。有关多区域 KMS 密钥的更多信息，请参阅在 AWS KMS 中使用多区域密钥。

有关 Amazon RDS 的 AWS KMS 密钥管理的更多信息，请参阅 AWS KMS key 管理。

源快照在复制过程中保持加密状态。有关更多信息，请参阅 Amazon Aurora 加密的数据库集群的限制。

> **Note**
>
> 对于 Amazon Aurora 数据库集群快照，在复制快照时，您无法对未加密的数据库集群快照进行加密。

要复制加密的数据库集群快照，请参阅以下主题。

- 使用 AWS CLI 或 Amazon RDS API 复制加密的数据库集群快照
- 跨账户复制数据库集群快照

跨区域快照复制注意事项

您可以跨 AWS 区域复制数据库集群快照。但是，跨区域快照复制具有某些限制和注意事项。

根据所涉及的 AWS 区域和要复制的数据量，可能需要数小时才能完成跨区域快照复制。

有时，某一给定的源 AWS 区域可能会发出大量跨区域快照复制请求。在这种情况下，Amazon RDS 可能会将来自该源 AWS 区域的新跨区域复制请求排入队列，直至某些正在进行的复制完成。当复制请求在队列中时，不显示有关这些复制请求的进度信息。复制开始后即显示进度信息。

跨区域快照复制会在目标数据中创建完整副本，但数据传输费用是递增的。增量数据包括自上次复制以来添加到客户数据库中的新数据，以及对现有数据所做的任何更改。有关更多信息，请参阅《AWS Backup 开发人员指南》中的 Creating backup copies across AWS 区域。

> **Note**
>
> Aurora 复制在目标区域中创建快照的完整副本所需的最低数据量。在区域之间复制快照时会收取数据传输费用。

参数组注意事项

跨区域复制快照时，复制不包括由原始数据库集群使用的参数组。当您还原快照来创建新数据库集群时，该数据库集群会获取创建它的 AWS 区域的默认参数组。要为新的数据库集群提供与源数据库集群相同的参数组，请执行以下操作：

1. 在目标 AWS 区域中，使用与原始数据库集群相同的设置来创建数据库集群参数组。如果新 AWS 区域中已存在选项组，也可以使用它。
2. 在目标 AWS 区域中还原快照之后，修改新数据库集群，并添加新参数组或上一步中的现有参数组。

使用 AWS CLI 或 Amazon RDS API 复制未加密的数据库集群快照

使用以下各节中的过程，通过 AWS Management Console、AWS CLI 或 Amazon RDS API 复制未加密的数据库集群快照。

要在正在进行复制时取消操作，请在数据库集群快照处于正在复制状态时删除由 `--target-db-cluster-snapshot-identifier` 或 `TargetDBClusterSnapshotIdentifier` 标识的目标数据库集群快照。

控制台

要使用 AWS Management Console复制数据库集群快照，请参阅使用 AWS Management Console复制数据库集群快照。

AWS CLI

要复制数据库快照，请使用 AWS CLI copy-db-cluster-snapshot 命令。如果您将快照复制到其他 AWS 区域，请在要将快照复制到其中的 AWS 区域运行命令。

以下选项用于复制未加密的数据库集群快照：

- `--source-db-cluster-snapshot-identifier` – 要复制的加密数据库集群快照的标识符。如果您正在将快照复制到另一个 AWS 区域，源 AWS 区域中的此标识符必须为 ARN 格式。

- `--target-db-cluster-snapshot-identifier` – 加密数据库集群快照的新副本的标识符。

以下代码在运行命令的 AWS 区域中创建名为 `myclustersnapshotcopy` 的数据库集群快照 `arn:aws:rds:us-east-1:123456789012:cluster-snapshot:aurora-cluster1-snapshot-20130805` 的副本。创建副本时，原始快照上的所有标签都将复制到快照副本。

Example

对于 Linux、macOS 或 Unix：

```
aws rds copy-db-cluster-snapshot \
  --source-db-cluster-snapshot-identifier arn:aws:rds:us-east-1:123456789012:cluster-snapshot:aurora-cluster1-snapshot-20130805 \
  --target-db-cluster-snapshot-identifier myclustersnapshotcopy \
  --copy-tags
```

对于 Windows：

```
aws rds copy-db-cluster-snapshot ^
  --source-db-cluster-snapshot-identifier arn:aws:rds:us-east-1:123456789012:cluster-snapshot:aurora-cluster1-snapshot-20130805 ^
  --target-db-cluster-snapshot-identifier myclustersnapshotcopy ^
  --copy-tags
```

RDS API

要复制数据库集群快照，请使用 Amazon RDS API CopyDBClusterSnapshot 操作。如果您将快照复制到其他 AWS 区域，请在要将快照复制到其中的 AWS 区域执行操作。

以下参数用于复制未加密的数据库集群快照：

- `SourceDBClusterSnapshotIdentifier` – 要复制的加密数据库集群快照的标识符。如果您正在将快照复制到另一个 AWS 区域，源 AWS 区域中的此标识符必须为 ARN 格式。
- `TargetDBClusterSnapshotIdentifier` – 加密数据库集群快照的新副本的标识符。

以下代码在 美国西部（加利福尼亚北部） 区域中创建名为 `arn:aws:rds:us-east-1:123456789012:cluster-snapshot:aurora-cluster1-snapshot-20130805` 的快照 `myclustersnapshotcopy` 的副本。创建副本时，原始快照上的所有标签都将复制到快照副本。

Example

```
https://rds.us-west-1.amazonaws.com/
   ?Action=CopyDBClusterSnapshot
   &CopyTags=true
   &SignatureMethod=HmacSHA256
   &SignatureVersion=4
   &SourceDBSnapshotIdentifier=arn%3Aaws%3Ards%3Aus-east-1%3A123456789012%3Acluster-
snapshot%3Aaurora-cluster1-snapshot-20130805
   &TargetDBSnapshotIdentifier=myclustersnapshotcopy
   &Version=2013-09-09
   &X-Amz-Algorithm=AWS4-HMAC-SHA256
   &X-Amz-Credential=AKIADQKE4SARGYLE/20140429/us-west-1/rds/aws4_request
   &X-Amz-Date=20140429T175351Z
   &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
   &X-Amz-Signature=9164337efa99caf850e874a1cb7ef62f3cea29d0b448b9e0e7c53b288ddffed2
```

使用 AWS CLI 或 Amazon RDS API 复制加密的数据库集群快照

使用以下各节中的过程，通过 AWS Management Console、AWS CLI 或 Amazon RDS API 复制加密的数据库集群快照。

要在正在进行复制时取消操作，请在数据库集群快照处于正在复制状态时删除由 `--target-db-cluster-snapshot-identifier` 或 `TargetDBClusterSnapshotIdentifier` 标识的目标数据库集群快照。

控制台

要使用 AWS Management Console复制数据库集群快照，请参阅使用 AWS Management Console复制数据库集群快照。

AWS CLI

要复制数据库快照，请使用 AWS CLI copy-db-cluster-snapshot 命令。如果您将快照复制到其他 AWS 区域，请在要将快照复制到其中的 AWS 区域运行命令。

以下选项用于复制加密的数据库集群快照：

- `--source-db-cluster-snapshot-identifier` – 要复制的加密数据库集群快照的标识符。如果您正在将快照复制到另一个 AWS 区域，源 AWS 区域中的此标识符必须为 ARN 格式。
- `--target-db-cluster-snapshot-identifier` – 加密数据库集群快照的新副本的标识符。

- `--kms-key-id` - 用于对数据库集群快照副本进行加密的密钥的 KMS 密钥标识符。

  如果数据库集群快照已加密，要在同一 AWS 区域中复制快照，并且希望指定新的 KMS 密钥用于加密副本，则可以选择使用该选项。否则，数据库集群快照副本将使用与源数据库集群快照相同的 KMS 密钥进行加密。

  如果数据库集群快照已加密，并且您要将该快照复制到另一 AWS 区域，则必须使用该选项。在这种情况下，必须为目标 AWS 区域指定 KMS 密钥。

以下代码示例将加密的数据库快照从美国西部（俄勒冈）区域复制到美国东部（弗吉尼亚北部）区域。在美国东部（弗吉尼亚北部）区域调用命令。

Example

对于 Linux、macOS 或 Unix：

```
aws rds copy-db-cluster-snapshot \
  --source-db-cluster-snapshot-identifier arn:aws:rds:us-west-2:123456789012:cluster-snapshot:aurora-cluster1-snapshot-20161115 \
  --target-db-cluster-snapshot-identifier myclustersnapshotcopy \
  --kms-key-id my-us-east-1-key
```

对于 Windows：

```
aws rds copy-db-cluster-snapshot ^
  --source-db-cluster-snapshot-identifier arn:aws:rds:us-west-2:123456789012:cluster-snapshot:aurora-cluster1-snapshot-20161115 ^
  --target-db-cluster-snapshot-identifier myclustersnapshotcopy ^
  --kms-key-id my-us-east-1-key
```

当您在 AWS GovCloud（美国东部）和 AWS GovCloud（美国西部）区域之间复制加密的数据库集群快照时，`--source-region` 参数是必需的。对于 `--source-region`，指定源数据库实例的 AWS 区域。`source-db-cluster-snapshot-identifier` 中指定的 AWS 区域必须匹配为 `--source-region` 指定的 AWS 区域。

如果未指定 `--source-region`，请指定 `--pre-signed-url` 值。预签名 URL 包含签名版本 4 签名的请求，该请求用于在源 AWS 区域中调用的 `copy-db-cluster-snapshot` 命令。要了解有关 `pre-signed-url` 选项的更多信息，请参阅《AWS CLI 命令参考》中的 copy-db-cluster-snapshot。

RDS API

要复制数据库集群快照，请使用 Amazon RDS API CopyDBClusterSnapshot 操作。如果您将快照复制到其他 AWS 区域，请在要将快照复制到其中的 AWS 区域执行操作。

以下参数用于复制加密的数据库集群快照：

- SourceDBClusterSnapshotIdentifier – 要复制的加密数据库集群快照的标识符。如果您正在将快照复制到另一个 AWS 区域，源 AWS 区域中的此标识符必须为 ARN 格式。
- TargetDBClusterSnapshotIdentifier – 加密数据库集群快照的新副本的标识符。
- KmsKeyId - 用于对数据库集群快照副本进行加密的密钥的 KMS 密钥标识符。

  如果数据库集群快照已加密，要在同一 AWS 区域中复制快照，并且指定新的 KMS 密钥用于加密副本，则可以选择使用该参数。否则，数据库集群快照副本将使用与源数据库集群快照相同的 KMS 密钥进行加密。

  如果数据库集群快照已加密，并且您要将该快照复制到另一 AWS 区域，则必须使用该参数。在这种情况下，必须为目标 AWS 区域指定 KMS 密钥。
- PreSignedUrl – 如果您将快照复制到其他 AWS 区域，必须指定 PreSignedUrl 参数。PreSignedUrl 值必须是一个 URL，包含要在源 AWS 区域（从中复制数据库集群快照）中调用的 CopyDBClusterSnapshot 操作的签名版本 4 签名的请求。要了解有关使用预签名 URL 的更多信息，请参阅 CopyDBClusterSnapshot。

以下代码示例将加密的数据库快照从 美国西部（俄勒冈） 区域复制到 US East (N. Virginia) 区域。在 US East (N. Virginia) 区域中调用操作。

Example

```
https://rds.us-east-1.amazonaws.com/
    ?Action=CopyDBClusterSnapshot
    &KmsKeyId=my-us-east-1-key
    &PreSignedUrl=https%253A%252F%252Frds.us-west-2.amazonaws.com%252F
         %253FAction%253DCopyDBClusterSnapshot
         %2526DestinationRegion%253Dus-east-1
         %2526KmsKeyId%253Dmy-us-east-1-key
         %2526SourceDBClusterSnapshotIdentifier%253Darn%25253Aaws%25253Ards
%25253Aus-west-2%25253A123456789012%25253Acluster-snapshot%25253Aaurora-cluster1-
snapshot-20161115
         %2526SignatureMethod%253DHmacSHA256
```

```
        %2526SignatureVersion%253D4
        %2526Version%253D2014-10-31
        %2526X-Amz-Algorithm%253DAWS4-HMAC-SHA256
        %2526X-Amz-Credential%253DAKIADQKE4SARGYLE%252F20161117%252Fus-west-2%252Frds
%252Faws4_request
        %2526X-Amz-Date%253D20161117T215409Z
        %2526X-Amz-Expires%253D3600
        %2526X-Amz-SignedHeaders%253Dcontent-type%253Bhost%253Buser-agent%253Bx-amz-
content-sha256%253Bx-amz-date
        %2526X-Amz-Signature
%253D255a0f17b4e717d3b67fad163c3ec26573b882c03a65523522cf890a67fca613
    &SignatureMethod=HmacSHA256
    &SignatureVersion=4
    &SourceDBClusterSnapshotIdentifier=arn%3Aaws%3Ards%3Aus-
west-2%3A123456789012%3Acluster-snapshot%3Aaurora-cluster1-snapshot-20161115
    &TargetDBClusterSnapshotIdentifier=myclustersnapshotcopy
    &Version=2014-10-31
    &X-Amz-Algorithm=AWS4-HMAC-SHA256
    &X-Amz-Credential=AKIADQKE4SARGYLE/20161117/us-east-1/rds/aws4_request
    &X-Amz-Date=20161117T221704Z
    &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
    &X-Amz-Signature=da4f2da66739d2e722c85fcfd225dc27bba7e2b8dbea8d8612434378e52adccf
```

当您在 PreSignedUrl GovCloud（美国东部）和 AWS GovCloud（美国西部）区域之间复制加密的数据库集群快照时，AWS 参数是必需的。PreSignedUrl 值必须是一个 URL，包含要在源 AWS 区域（从中复制数据库集群快照）中调用的 CopyDBClusterSnapshot 操作的签名版本 4 签名的请求。要了解有关使用预签名 URL 的更多信息，请参阅《Amazon RDS API 参考》中的 CopyDBClusterSnapshot。

要自动而非手动生成预签名的 URL，请改用具有 --source-region 选项的 AWS CLI copy-db-cluster-snapshot 命令。

跨账户复制数据库集群快照

您可以允许其他 AWS 账户通过使用 Amazon RDS API ModifyDBClusterSnapshotAttribute 和 CopyDBClusterSnapshot 操作来复制您指定的数据库集群快照。您只能在同一 AWS 区域中跨账户复制数据库集群快照。跨账户复制的流程如下，其中账户 A 提供可复制的快照，账户 B 复制该快照。

1. 使用账户 A，调用 ModifyDBClusterSnapshotAttribute，为 **restore** 参数指定 AttributeName，并为 ValuesToAdd 参数指定账户 B 的 ID。

2. (如果快照已加密) 使用账户 A，更新 KMS 密钥的密钥策略，首先将账户 B 的 ARN 添加为 `Principal`，然后允许 `kms:CreateGrant` 操作。
3. （如果快照已加密）使用账户 B，选择或创建用户，然后将允许它使用 KMS 密钥复制加密数据库集群快照的 IAM policy 附加到该用户。
4. 使用账户 B，调用 `CopyDBClusterSnapshot` 并使用 `SourceDBClusterSnapshotIdentifier` 参数指定要复制的数据库集群快照的 ARN，其中必须包括账户 A 的 ID。

要列出允许还原数据库集群快照的所有 AWS 账户，请使用 DescribeDBSnapshotAttributes 或 DescribeDBClusterSnapshotAttributes API 操作。

要取消 AWS 账户的共享权限，请使用 `ModifyDBSnapshotAttribute` 或 `ModifyDBClusterSnapshotAttribute` 操作并将 `AttributeName` 设置为 `restore`，然后在 `ValuesToRemove` 参数中删除账户的 ID。

将未加密的数据库集群快照复制到另一账户

使用以下步骤将未加密的数据库集群快照复制到同一 AWS 区域中的另一账户。

1. 在数据库集群快照的源账户中，调用 `ModifyDBClusterSnapshotAttribute`，为 **`restore`** 参数指定 `AttributeName`，并为 `ValuesToAdd` 参数指定目标账户的 ID。

   使用账户 987654321 运行以下示例将允许两个 AWS 账户标识符（123451234512 和 123456789012）恢复名为 `manual-snapshot1` 的数据库集群快照。

```
https://rds.us-west-2.amazonaws.com/
 ?Action=ModifyDBClusterSnapshotAttribute
 &AttributeName=restore
 &DBClusterSnapshotIdentifier=manual-snapshot1
 &SignatureMethod=HmacSHA256&SignatureVersion=4
 &ValuesToAdd.member.1=123451234512
 &ValuesToAdd.member.2=123456789012
 &Version=2014-10-31
 &X-Amz-Algorithm=AWS4-HMAC-SHA256
 &X-Amz-Credential=AKIADQKE4SARGYLE/20150922/us-west-2/rds/aws4_request
 &X-Amz-Date=20150922T220515Z
 &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
 &X-Amz-Signature=ef38f1ce3dab4e1dbf113d8d2a265c67d17ece1999ffd36be85714ed36dddbb3
```

2. 在目标账户中，调用 CopyDBClusterSnapshot 并使用 SourceDBClusterSnapshotIdentifier 参数指定要复制的数据库集群快照的 ARN，其中必须包括源账户的 ID。

   使用账户 123451234512 运行以下示例将从账户 aurora-cluster1-snapshot-20130805 复制数据库集群快照 987654321，然后创建名为 dbclustersnapshot1 的数据库集群快照。

```
https://rds.us-west-2.amazonaws.com/
   ?Action=CopyDBClusterSnapshot
   &CopyTags=true
   &SignatureMethod=HmacSHA256
   &SignatureVersion=4
   &SourceDBClusterSnapshotIdentifier=arn:aws:rds:us-west-2:987654321:cluster-
snapshot:aurora-cluster1-snapshot-20130805
   &TargetDBClusterSnapshotIdentifier=dbclustersnapshot1
   &Version=2013-09-09
   &X-Amz-Algorithm=AWS4-HMAC-SHA256
   &X-Amz-Credential=AKIADQKE4SARGYLE/20150922/us-west-2/rds/aws4_request
   &X-Amz-Date=20140429T175351Z
   &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-
date
   &X-Amz-
Signature=9164337efa99caf850e874a1cb7ef62f3cea29d0b448b9e0e7c53b288ddffed2
```

将加密的数据库集群快照复制到另一账户

使用以下步骤将加密的数据库集群快照复制到同一 AWS 区域中的另一账户。

1. 在数据库集群快照的源账户中，调用 ModifyDBClusterSnapshotAttribute，为 **restore** 参数指定 AttributeName，并为 ValuesToAdd 参数指定目标账户的 ID。

   使用账户 987654321 运行以下示例将允许两个 AWS 账户标识符（123451234512 和 123456789012）恢复名为 manual-snapshot1 的数据库集群快照。

```
https://rds.us-west-2.amazonaws.com/
 ?Action=ModifyDBClusterSnapshotAttribute
 &AttributeName=restore
 &DBClusterSnapshotIdentifier=manual-snapshot1
 &SignatureMethod=HmacSHA256&SignatureVersion=4
 &ValuesToAdd.member.1=123451234512
 &ValuesToAdd.member.2=123456789012
```

```
 &Version=2014-10-31
 &X-Amz-Algorithm=AWS4-HMAC-SHA256
 &X-Amz-Credential=AKIADQKE4SARGYLE/20150922/us-west-2/rds/aws4_request
 &X-Amz-Date=20150922T220515Z
 &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
 &X-Amz-Signature=ef38f1ce3dab4e1dbf113d8d2a265c67d17ece1999ffd36be85714ed36dddbb3
```

2. 在数据库集群快照的源账户中，在加密数据库集群快照所在的同一个 AWS 区域中创建自定义 KMS 密钥。在创建客户自主管理型密钥时，您可以为目标 AWS 账户提供对该密钥的访问权限。有关更多信息，请参阅 创建客户自主管理型密钥并授予对它的访问权限。
3. 将快照复制到目标 AWS 账户并与其共享。有关更多信息，请参阅 从源账户复制和共享快照。
4. 在目标账户中，调用 CopyDBClusterSnapshot 并使用 SourceDBClusterSnapshotIdentifier 参数指定要复制的数据库集群快照的 ARN，其中必须包括源账户的 ID。

   使用账户 123451234512 运行以下示例将从账户 aurora-cluster1-snapshot-20130805 复制数据库集群快照 987654321，然后创建名为 dbclustersnapshot1 的数据库集群快照。

```
https://rds.us-west-2.amazonaws.com/
   ?Action=CopyDBClusterSnapshot
   &CopyTags=true
   &SignatureMethod=HmacSHA256
   &SignatureVersion=4
   &SourceDBClusterSnapshotIdentifier=arn:aws:rds:us-west-2:987654321:cluster-
snapshot:aurora-cluster1-snapshot-20130805
   &TargetDBClusterSnapshotIdentifier=dbclustersnapshot1
   &Version=2013-09-09
   &X-Amz-Algorithm=AWS4-HMAC-SHA256
   &X-Amz-Credential=AKIADQKE4SARGYLE/20150922/us-west-2/rds/aws4_request
   &X-Amz-Date=20140429T175351Z
   &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-
date
   &X-Amz-
Signature=9164337efa99caf850e874a1cb7ef62f3cea29d0b448b9e0e7c53b288ddffed2
```

共享数据库集群快照

使用 Amazon RDS，您可以按以下方式共享手动数据库集群快照：

- 共享手动数据库集群快照（无论是否加密）可允许经授权的 AWS 账户复制快照。
- 共享手动数据库集群快照（无论是否加密）可允许经授权的 AWS 账户直接从快照还原数据库集群，无需复制数据库集群再从中进行还原。

> **Note**
>
> 要共享自动数据库集群快照，请通过复制自动快照来创建手动数据库集群快照，然后共享该副本。此过程也适用于 AWS Backup 生成的资源。

有关复制快照的更多信息，请参阅数据库集群快照复制。有关从数据库集群快照还原数据库实例的更多信息，请参阅从数据库集群快照还原。

有关从数据库群集快照还原数据库群集的更多信息，请参阅 备份和还原 Aurora 数据库集群的概述。

您可以与最多 20 个其他 AWS 账户共享手动快照。

与其它 AWS 账户共享手动快照时存在以下限制：

- 当使用 AWS Command Line Interface (AWS CLI) 或 Amazon RDS API 从共享的快照还原数据库集群时，您必须指定共享快照的 Amazon 资源名称（ARN）作为快照标识符。

在以下各节中学习共享快照、公有快照和加密快照。还可以学习如何停止共享快照。

主题

- 共享快照
- 共享公有快照
- 共享加密的快照
- 停止快照共享

共享快照

您可以使用 AWS Management Console、AWS CLI 或 RDS API 共享数据库集群快照。

控制台

使用 Amazon RDS 控制台，可以与多达 20 个 AWS 账户共享手动数据库集群快照。您还可以使用该控制台停止与一个或多个账户共享手动快照。

使用 Amazon RDS 控制台共享手动数据库集群快照

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择快照。
3. 选择要共享的手动快照。
4. 对于 Actions（操作），请选择 Share snapshot（共享快照）。
5. 为 DB snapshot visibility (数据库快照可见性) 选择以下一个选项。

   - 如果源未加密，则选择公有以允许所有 AWS 账户从您的手动数据库集群快照还原数据库集群，或选择私有以仅允许您指定的 AWS 账户从手动数据库集群快照还原数据库集群。

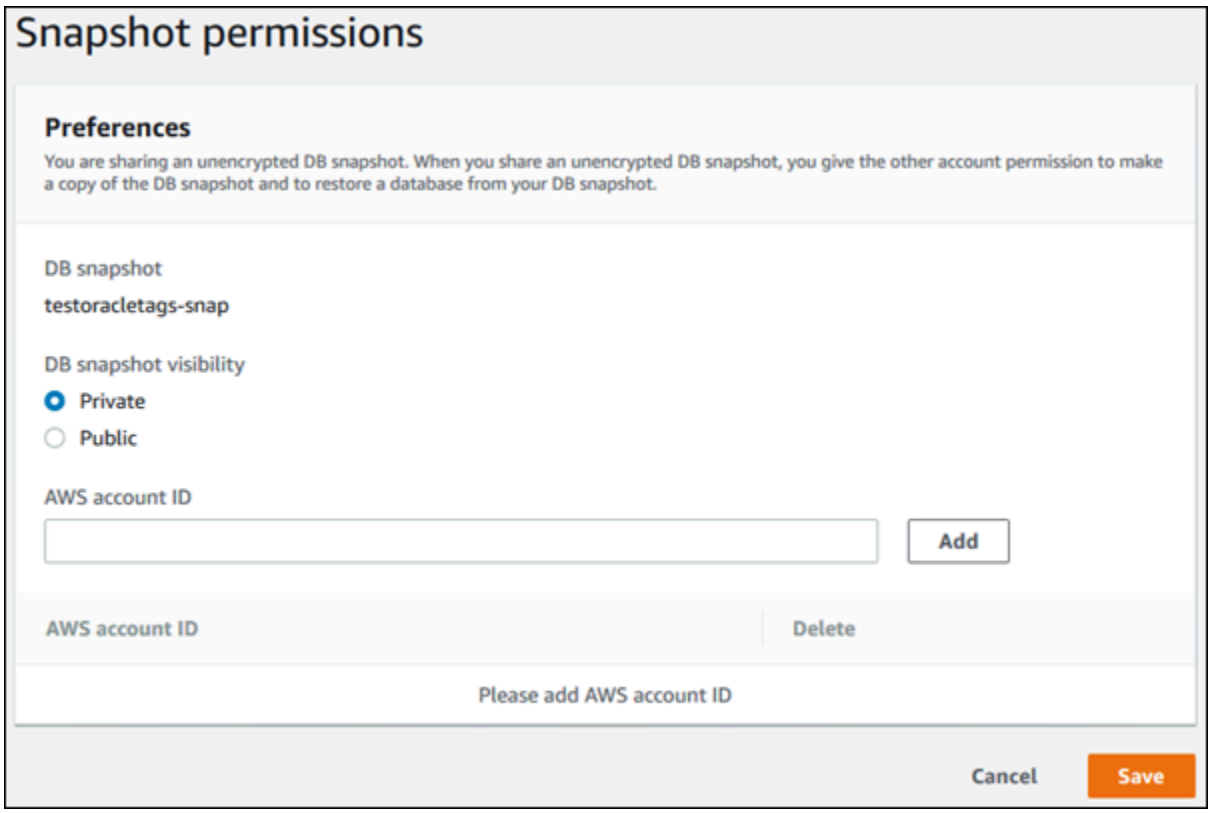
     > **Warning**
     >
     > 如果将数据库快照可见性设置为公有，则所有 AWS 账户均可从您的手动数据库集群快照还原数据库集群，并且可访问您的数据。请勿将包含私密信息的任何手动数据库集群快照以公开形式共享。
     > 有关更多信息，请参阅 共享公有快照。

   - 如果源已加密，由于已加密的快照无法公开共享，DB snapshot visibility (数据库快照可见性) 将设为 Private (私密)。

     > **Note**
     >
     > 无法共享已使用默认 AWS KMS key 加密的快照。有关如何解决此问题的信息，请参阅共享加密的快照。

6. 对于 AWS 账户 ID，输入您想要允许从您的手动快照还原数据库集群的账户的 AWS 账户标识符，然后选择添加。重复操作以加入其它 AWS 账户标识符，最多可包含 20 个 AWS 账户。

   如果您在许可账户列表中错加了某个 AWS 账户标识符，可以选择错误 AWS 账户标识符右侧的删除将其从列表中删除。



7. 为您想要允许还原手动快照的所有 AWS 账户添加标识符以后，选择保存以保存您的更改。

AWS CLI

要共享数据库集群快照，请使用 `aws rds modify-db-cluster-snapshot-attribute` 命令。使用 `--values-to-add` 参数添加有权还原手动快照的 AWS 账户的 ID 列表。

Example 与单个账户共享快照

以下示例使 AWS 账户标识符 `123456789012` 能够还原名为 `cluster-3-snapshot` 的数据库集群快照。

对于 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster-snapshot-attribute \
--db-cluster-snapshot-identifier cluster-3-snapshot \
--attribute-name restore \
--values-to-add 123456789012
```

对于 Windows：

```
aws rds modify-db-cluster-snapshot-attribute ^
```

```
--db-cluster-snapshot-identifier cluster-3-snapshot ^
--attribute-name restore ^
--values-to-add 123456789012
```

Example 与多个账户共享快照

以下示例使两个 AWS 账户标识符（111122223333 和 444455556666）能够还原名为 manual-cluster-snapshot1 的数据库集群快照。

对于 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster-snapshot-attribute \
--db-cluster-snapshot-identifier manual-cluster-snapshot1 \
--attribute-name restore \
--values-to-add {"111122223333","444455556666"}
```

对于 Windows：

```
aws rds modify-db-cluster-snapshot-attribute ^
--db-cluster-snapshot-identifier manual-cluster-snapshot1 ^
--attribute-name restore ^
--values-to-add "[\"111122223333\",\"444455556666\"]"
```

> **Note**
>
> 使用 Windows 命令提示符时，必须在 JSON 代码中转义双引号 (")，方法是使用反斜杠 (\) 作为其前缀。

要列出能够用于还原快照的 AWS 账户，请使用 describe-db-cluster-snapshot-attributes AWS CLI 命令。

RDS API

您还可以使用 Amazon RDS API 与其它 AWS 账户共享手动数据库集群快照。为此，请调用 ModifyDBClusterSnapshotAttribute 操作。为 AttributeName 指定 restore，并使用 ValuesToAdd 参数添加有权还原手动快照的 AWS 账户的 ID 列表。

要将手动快照设为公有并允许所有 AWS 账户进行还原，请使用值 all。但请注意，若任何手动快照包含您不想向所有 AWS 账户公开的私有信息，则不要添加 all 值。此外，由于此类快照不支持公开共享，请不要为已加密的快照指定 all。

要列出有权还原快照的所有 AWS 账户，请使用 DescribeDBClusterSnapshotAttributes API 操作。

共享公有快照

您可以将未加密的手动快照作为公有快照进行共享，这样所有 AWS 账户均可使用此快照。当以公有快照形式共享快照时，确保不要将您的私有信息包含在公有快照之中。

公开共享快照时，它会授予所有 AWS 账户复制快照并从中创建数据库集群的权限。

您不需要为其他账户拥有的公有快照的备份存储付费。您只需为您拥有的快照付费。

如果您复制公有快照，则您拥有该副本。您需要为快照副本的备份存储付费。如果您从公有快照创建数据库集群，则需要为该数据库集群付费。有关 Amazon Aurora 定价信息，请参阅 Aurora 定价页面。

您只能删除您拥有的公有快照。要删除共享或公有快照，务必登录到拥有快照的 AWS 账户。

查看其它 AWS 账户拥有的公有快照

您可以在 Amazon RDS 控制台的快照页面的公有选项卡上查看特定 AWS 区域中其它账户拥有的公有快照。您的快照（由您账户拥有的快照）不会显示在此选项卡上。

要查看公有快照

1. 通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择快照。
3. 选择 Public (公有) 选项卡。

   此时将显示公有快照。您可以在 Owner (拥有者) 列中查看哪个账户拥有公有快照。

> **Note**
>
> 您可能需要通过选择 Public snapshots (公有快照) 列表右上角的齿轮图标来修改页面首选项，才能看到此列。

查看您拥有的公有快照

您可以使用以下 AWS CLI 命令（仅限 Unix）查看您的 AWS 账户在特定 AWS 区域中拥有的公有快照。

```
aws rds describe-db-cluster-snapshots --snapshot-type public --include-public |
 grep account_number
```

如果您有公共快照，则返回的输出与以下示例类似。

```
"DBClusterSnapshotArn": "arn:aws:rds:us-west-2:123456789012:cluster-
snapshot:myclustersnapshot1",
"DBClusterSnapshotArn": "arn:aws:rds:us-west-2:123456789012:cluster-
snapshot:myclustersnapshot2",
```

共享来自已弃用数据库引擎版本的公有快照

不支持从已弃用的数据库引擎版本还原或复制公有快照。要使现有的不受支持的公有快照可供还原或复制，请执行以下步骤：

1. 将快照标记为私有。
2. 还原快照。
3. 将还原的数据库集群升级到受支持的引擎版本。
4. 创建新的快照。
5. 公开重新共享快照。

共享加密的快照

您可共享使用 AES-256 加密算法“静态”加密的数据库集群快照，如加密 Amazon Aurora 资源中所述。

以下限制适用于共享加密快照：

- 您无法公开共享加密的快照。
- 如果某个快照已使用共享该快照的 AWS 账户的默认 KMS 密钥进行加密，则您无法共享该快照。

  有关 Amazon RDS 的 AWS KMS 密钥管理的更多信息，请参阅 AWS KMS key 管理。

要解决默认 KMS 密钥问题，请执行以下任务：

1. 创建客户自主管理型密钥并授予对它的访问权限.
2. 从源账户复制和共享快照.
3. 复制目标账户中的共享快照.

创建客户自主管理型密钥并授予对它的访问权限

首先，在加密数据库集群快照所在的同一个 AWS 区域中创建一个自定义 KMS 密钥。在创建客户自主管理型密钥时，您可以为另一个 AWS 账户提供对它的访问权限。

创建客户自主管理型密钥并授予对它的访问权限

1. 从源 AWS 账户登录 AWS Management Console。
2. 从 https://console.aws.amazon.com/kms 打开 AWS KMS 控制台。
3. 要更改 AWS 区域，请使用页面右上角的区域选择器。
4. 在导航窗格中，选择客户托管密钥。
5. 选择 Create key。
6. 在配置密钥页面上：
   a. 对于密钥类型，选择对称。
   b. 对于密钥用途，选择加密和解密。
   c. 展开 Advanced options (高级选项)。
   d. 对于密钥材料来源，选择 KMS。
   e. 对于区域性，请选择单区域密钥。
   f. 选择下一步。
7. 在添加标签页面上：
   a. 对于别名，输入您的 KMS 密钥的显示名称，例如 **`share-snapshot`**。
   b. （可选）为 KMS 密钥输入描述。
   c. （可选）向 KMS 密钥添加标签。
   d. 选择下一步。
8. 在定义密钥管理权限页面上，选择下一步。
9. 在定义密钥使用权限页面上：
   a. 对于其他 AWS 账户，选择添加另一个 AWS 账户。
   b. 输入您要向其授予访问权限的 AWS 账户的 ID。

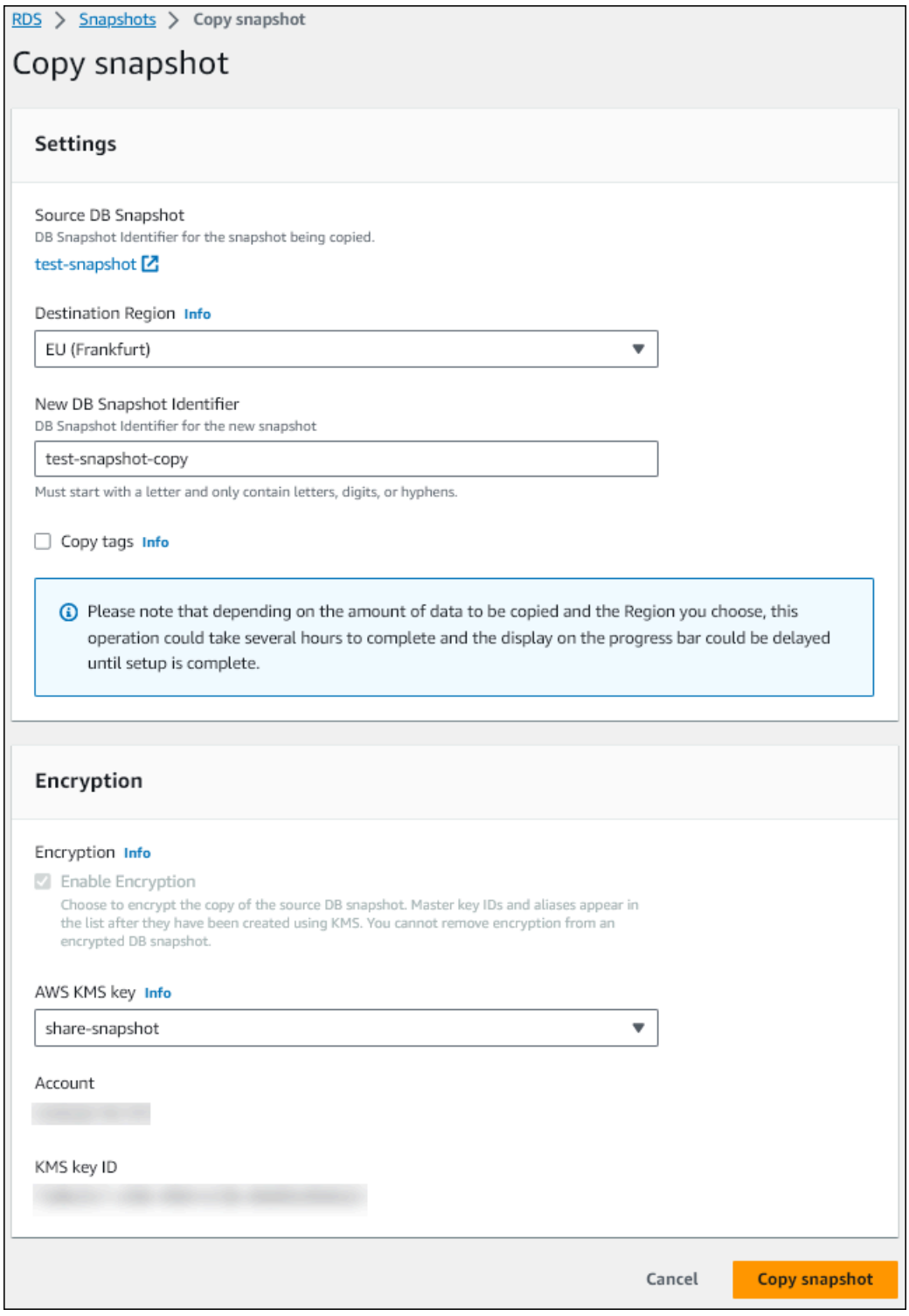
      您可以向多个 AWS 账户授予访问权限。
   c. 选择下一步。
10. 查看您的 KMS 密钥，然后选择完成。

从源账户复制和共享快照

接下来，使用客户自主管理型密钥将源数据库集群快照复制到新快照。然后，您将与目标 AWS 账户共享它。

复制和共享快照

1. 从源 AWS 账户登录 AWS Management Console。
2. 通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/
3. 在导航窗格中，选择快照。
4. 选择要复制的数据库集群快照。
5. 对于 Actions (操作)，请选择 Copy snapshot (复制快照)。
6. 在复制快照页面上：
    a. 对于目标区域，选择您在上一个过程中创建客户自主管理型密钥的 AWS 区域。
    b. 在 New DB Snapshot Identifier (新数据库实例标识符) 中输入数据库集群快照副本的名称。
    c. 对于 AWS KMS key，选择您创建的客户自主管理型密钥。



d. 选择复制快照。

7. 当快照副本可用时，将其选中。
8. 对于 Actions（操作），请选择 Share snapshot（共享快照）。
9. 在快照权限页面上：

a. 输入您要与之共享快照副本的 AWS 账户 ID，然后选择添加。
b. 选择保存。

共享此快照。

复制目标账户中的共享快照

现在，您可以复制目标 AWS 账户中的共享快照。

复制共享快照

1. 从目标 AWS 账户登录 AWS Management Console。
2. 通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/
3. 在导航窗格中，选择快照。
4. 选择与我共享选项卡。
5. 选择共享快照。
6. 对于 Actions (操作)，请选择 Copy snapshot (复制快照)。
7. 按照前面的过程选择用于复制快照的设置，但要使用属于目标账户的 AWS KMS key。

选择复制快照。

停止快照共享

要停止共享数据库集群快照，请删除目标 AWS 账户的权限。

控制台

停止与 AWS 账户共享手动数据库集群快照

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择快照。
3. 选择要停止共享的手动快照。
4. 选择 Actions（操作），然后选择 Share snapshot（共享快照）。
5. 要移除 AWS 账户的权限，请从授权账户列表中选择该账户的 AWS 账户标识符所对应的删除。

6. 选择 保存 以保存您的更改。

CLI

要从列表中删除 AWS 账户标识符，请使用 `--values-to-remove` 参数。

Example 停止共享快照

以下示例禁止 AWS 账户 ID 444455556666 还原快照。

对于 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster-snapshot-attribute \
--db-cluster-snapshot-identifier manual-cluster-snapshot1 \
--attribute-name restore \
--values-to-remove 444455556666
```

对于 Windows：

```
aws rds modify-db-cluster-snapshot-attribute ^
--db-cluster-snapshot-identifier manual-cluster-snapshot1 ^
--attribute-name restore ^
--values-to-remove 444455556666
```

RDS API

要移除 AWS 账户的共享权限，请使用 ModifyDBClusterSnapshotAttribute 操作（`AttributeName` 设置为 `restore`）和 `ValuesToRemove` 参数。要将手动快照标记为私有，请将值 `all` 从 `restore` 属性的值列表中删除。

将数据库集群数据导出到 Amazon S3

您可以将数据从实时 Amazon Aurora 数据库集群导出到 Amazon S3 桶。导出过程在后台运行，不会影响活动数据库集群的性能。

原定设置情况下，将导出数据库集群中的所有数据。但是，您可以选择导出特定的一组数据库、方案或表。

Amazon Aurora 克隆数据库集群，从克隆中提取数据，并将数据存储在 Amazon S3 桶中。数据以压缩和一致的 Apache Parquet 格式存储。各个 Parquet 文件的大小通常约为 1-10MB。

导出 Aurora MySQL 版本 2 和版本 3 的快照数据所能获得的更快的性能不适用于导出数据库集群数据。有关更多信息，请参阅 将数据库集群快照数据导出到 Amazon S3。

无论是导出全部数据还是部分数据，您都需要为导出整个数据库集群付费。有关更多信息，请参阅 Amazon Aurora 定价页面。

导出数据后，您可以通过 Amazon Athena 或 Amazon Redshift Spectrum 等工具直接分析导出的数据。有关使用 Athena 读取 Parque 数据的更多信息，请参阅 Amazon Athena 用户指南中的 Parquet SerDe。有关使用 Redshift Spectrum 读取 Parquet 数据的更多信息，请参阅《Amazon Redshift 数据库开发人员指南》中的从列式数据格式执行 COPY 操作。

功能可用性和支持因每个数据库引擎的特定版本以及 AWS 区域而异。有关将数据库集群数据导出到 S3 的版本和区域可用性的更多信息，请参阅 支持将集群数据导出到 Amazon S3 的区域和 Aurora 数据库引擎。

您可以使用以下过程将数据库集群数据导出到 Amazon S3 桶。有关更多详细信息，请参阅以下部分。

导出数据库集群数据概述

1. 标识要导出其数据的数据库集群。
2. 设置对 Amazon S3 存储桶的访问权限。

   存储桶是 Amazon S3 对象或文件的容器。要提供访问存储桶的信息，请执行以下步骤：

   a. 标识要将数据库集群数据导出到的 S3 桶。S3 桶和数据库集群必须位于同一 AWS 区域中。有关更多信息，请参阅 标识要导出到的 Amazon S3 存储桶。
   b. 创建一个 AWS Identity and Access Management（IAM）角色，用于授予数据库集群导出任务对 S3 桶的访问权限。有关更多信息，请参阅 使用 IAM 角色提供对 Amazon S3 存储桶的访问权限。

3. 创建对称加密 AWS KMS key 以进行服务器端加密。集群导出任务使用 KMS 密钥在将导出数据写入 S3 时设置 AWS KMS 服务器端加密。

   KMS 密钥策略必须同时包含 `kms:CreateGrant` 和 `kms:DescribeKey` 权限。有关在 Amazon Aurora 中使用 KMS 密钥的更多信息，请参阅 AWS KMS key 管理。

   如果 KMS 密钥策略中有拒绝语句，则确保显式排除 AWS 服务主体 `export.rds.amazonaws.com`。

   您可以在您的 AWS 账户内使用 KMS 密钥，或者您可以使用跨账户 KMS 密钥。有关更多信息，请参阅 使用跨账户 AWS KMS key。
4. 使用控制台或 `start-export-task` CLI 命令将数据库集群导出到 Amazon S3。有关更多信息，请参阅 创建数据库集群导出任务。
5. 要访问 Amazon S3 存储桶中导出的数据，请参阅 Amazon Simple Storage Service 用户指南中的上传、下载和管理对象。

在以下各节中，学习设置、导出、监控、取消数据库集群导出任务并对其进行故障排除。

主题

- 数据库集群导出注意事项
- 设置 Amazon S3 存储桶的访问权限
- 创建数据库集群导出任务
- 监控数据库集群导出任务
- 取消数据库集群导出任务
- 数据库集群导出故障排除

数据库集群导出注意事项

使用以下各节来了解将数据库集群数据导出到 Amazon S3 时的限制、文件命名约定以及数据转换和存储。

主题

- 限制
- 文件命名约定
- 数据转换和存储格式

限制

将数据库集群数据导出到 Amazon S3 有以下限制：

- 您不能同时为同一个数据库集群运行多个导出任务。这同时适用于完全导出和部分导出。
- 每个 AWS 账户 最多可以执行五个并发数据库快照导出任务。
- Aurora Serverless v1 数据库集群不支持导出到 S3。
- Aurora MySQL 和 Aurora PostgreSQL 仅对于预调配的引擎模式支持导出到 S3。
- 导出到 S3 不支持包含冒号 (:) 的 S3 前缀。
- 在导出过程中，S3 文件路径中的以下字符将转换为下划线 (_)：

```
\ ` " (space)
```

- 如果数据库、架构或表的名称中包含以下字符以外的字符，则不支持部分导出。但是，您可以导出整个数据库集群。
  - 拉丁字母 (A–Z)
  - 数字 (0–9)
  - 美元符号 ($)
  - 下划线 (_)
- 数据库表列名不支持空格 ( ) 和某些字符。在导出过程中会跳过列名中包含以下字符的表：

```
, ; { } ( ) \n \t = (space)
```

- 在导出过程中会跳过其名称中包含斜杠 (/) 的表。
- 在导出期间，将跳过 Aurora PostgreSQL 临时表和未记录的表。
- 如果数据包含接近或大于 500MB 的大型对象（例如 BLOB 或 CLOB），则导出失败。
- 如果表中某个大行的大小接近或大于 2GB，则会在导出过程中略过该表。
- 对于部分导出，`ExportOnly` 列表的最大大小为 200 KB。
- 强烈建议您为每个导出任务使用唯一的名称。如果您没有使用唯一的任务名称，可能会收到以下错误消息：

  ExportTaskAlreadyExistsFault：调用 StartExportTask 操作时发生错误 (ExportTaskAlreadyExists)：ID 为 *xxxxx* 的导出任务已存在。
- 由于某些表可能会被跳过，因此建议您在导出后验证数据中的行数和表数。

文件命名约定

特定表的导出数据以 *base_prefix*/*files* 格式存储，基本前缀如下：

```
export_identifier/database_name/schema_name.table_name/
```

例如：

```
export-1234567890123-459/rdststcluster/mycluster.DataInsert_7ADB5D19965123A2/
```

输出文件使用以下命名约定，其中 *partition_index* 是字母数字：

```
partition_index/part-00000-random_uuid.format-based_extension
```

例如：

```
1/part-00000-c5a881bb-58ff-4ee6-1111-b41ecff340a3-c000.gz.parquet
    a/part-00000-d7a881cc-88cc-5ab7-2222-c41ecab340a4-c000.gz.parquet
```

文件命名约定可能会更改。因此，在读取目标表时，我们建议您读取表的基本前缀内的所有内容。

数据转换和存储格式

将数据库集群导出到 Amazon S3 存储桶时，Amazon Aurora 以 Parquet 格式转换、导出并存储数据。有关更多信息，请参阅 导出到 Amazon S3 存储桶时的数据转换。

设置 Amazon S3 存储桶的访问权限

您标识 AmazonS3 桶，然后授予数据库集群导出任务访问它的权限。

主题

- 标识要导出到的 Amazon S3 存储桶
- 使用 IAM 角色提供对 Amazon S3 存储桶的访问权限
- 使用跨账户 Amazon S3 存储桶

标识要导出到的 Amazon S3 存储桶

标识要将数据库集群数据导出到的 Amazon S3 桶。使用现有 S3 存储桶或创建新的 S3 存储桶。

> **Note**
>
> S3 桶和数据库集群必须位于同一 AWS 区域中。

有关使用 Amazon S3 存储桶的详细信息，请参阅 Amazon Simple Storage Service 用户指南中的以下主题：

- 如何查看 S3 存储桶的属性？
- 如何为 Amazon S3 存储桶启用默认加密？
- 如何创建 S3 存储桶？

使用 IAM 角色提供对 Amazon S3 存储桶的访问权限

将数据库集群数据导出到 Amazon S3 之前，请授予导出任务对 Amazon S3 桶的写入访问权限。

要授予此权限，请创建 IAM 策略以提供对桶的访问权限，然后创建一个 IAM 角色并将该策略附加到该角色。稍后，您可以将此 IAM 角色分配给数据库集群导出任务。

> **Important**
>
> 如果计划使用 AWS Management Console导出数据库集群，则可以选择在导出数据库集群时自动创建 IAM 策略和角色。有关说明，请参阅 创建数据库集群导出任务。

授予任务访问 Amazon S3 的权限

1. 创建一个 IAM 策略。此策略提供允许数据库集群导出任务访问 Amazon S3 的桶和对象权限。

   在策略中，包含以下必需操作，以允许将文件从 Amazon Aurora 桶传输到 S3 桶：

   - `s3:PutObject*`
   - `s3:GetObject*`
   - `s3:ListBucket`
   - `s3:DeleteObject*`

- s3:GetBucketLocation

在策略中，包含以下资源以标识 S3 桶以及该桶中的对象。以下资源列表显示用于访问 Amazon S3 的 Amazon Resource Name (ARN) 格式。

- arn:aws:s3:::*amzn-s3-demo-bucket*
- arn:aws:s3:::*amzn-s3-demo-bucket*/*

有关为 Amazon Aurora 创建 IAM 策略的更多信息，请参阅 创建和使用适用于 IAM 数据库访问的 IAM 策略。另请参阅 IAM 用户指南中的教程：创建和附加您的第一个客户托管式策略。

以下 AWS CLI 命令使用这些选项创建一个名为 ExportPolicy 的 IAM 策略。该策略授予对名为 *amzn-s3-demo-bucket* 的存储桶的访问权限。

> **Note**
>
> 创建策略后，请记下策略的 ARN。在将策略附加到 IAM 角色时，您在后面的步骤中需要使用 ARN。

```
aws iam create-policy  --policy-name ExportPolicy --policy-document '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ExportPolicy",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject*",
                "s3:ListBucket",
                "s3:GetObject*",
                "s3:DeleteObject*",
                "s3:GetBucketLocation"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket",
                "arn:aws:s3:::amzn-s3-demo-bucket/*"
            ]
        }
    ]
```

```
}'
```

2. 创建一个 IAM 角色，以便 Aurora 可以代入该 IAM 角色，代表您访问 Amazon S3 桶。有关更多信息，请参阅 IAM 用户指南中的创建向 IAM 用户委派权限的角色。

   以下示例说明了如何使用 AWS CLI 命令创建一个名为 `rds-s3-export-role` 的角色。

```
aws iam create-role  --role-name rds-s3-export-role  --assume-role-policy-document
 '{
    "Version": "2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Principal": {
           "Service": "export.rds.amazonaws.com"
         },
        "Action": "sts:AssumeRole"
      }
    ]
  }'
```

3. 将您创建的 IAM 策略附加到您创建的 IAM 角色。

   以下 AWS CLI 命令将之前创建的策略附加到名为 `rds-s3-export-role` 的角色。将 *`your-policy-arn`* 替换为您在先前步骤中记下的策略 ARN。

```
aws iam attach-role-policy  --policy-arn your-policy-arn  --role-name rds-s3-
export-role
```

使用跨账户 Amazon S3 存储桶

您可以跨 AWS 账户使用 S3 桶。有关更多信息，请参阅 使用跨账户 Amazon S3 存储桶。

创建数据库集群导出任务

创建导出任务，来将数据从 Aurora 数据库集群导出到 Amazon S3 存储桶。每个 AWS 账户 最多可以执行五个并发数据库集群导出任务。

> **Note**
>
> 导出数据库集群数据可能需要一段时间，具体取决于您的数据库类型和大小。导出任务首先克隆并扩展整个数据库，然后再将数据提取到 Amazon S3。此阶段的任务进度显示为正在启动。当任务切换到将数据导出到 S3 时，进度显示为正在进行。
> 完成导出所需的时间取决于数据库中存储的数据。例如，具有分布良好的数字主键或索引列的表导出速度最快。不包含适用于分区的列的表，以及只有基于字符串的列上的一个索引的表需要更长时间，因为导出使用较慢的单线程进程。

您可以使用 AWS Management Console、AWS CLI 或 RDS API 将数据库集群数据导出到 Amazon S3。

如果您使用 Lambda 函数导出数据库集群数据，请将 `kms:DescribeKey` 操作添加到 Lambda 函数策略中。有关更多信息，请参阅 AWS Lambda 权限。

控制台

仅为可导出到 Amazon S3 的数据库集群显示 Export to Amazon S3（导出到 Amazon S3）控制台选项。由于以下原因，数据库集群可能无法导出：

- 此数据库引擎不支持 S3 导出。
- 数据库集群版本不支持 S3 导出。
- 创建了数据库集群的 AWS 区域不支持 S3 导出。

导出数据库集群数据

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Databases（数据库）。
3. 选择要导出其数据的数据库集群。
4. 对于 Actions (操作)，选择 Export to Amazon S3 (导出到 Amazon S3)。

   此时将显示 Export to Amazon S3 (导出到 Amazon S3) 窗口。
5. 对于 Export identifier (导出标识符)，输入用于标识导出任务的名称。此值也用于在 S3 存储桶中创建的文件的名称。
6. 选择要导出的数据：

- 选择 All（全部）可导出数据库集群中的所有数据。
- 选择 Partial（部分）可导出数据库集群的特定部分。如需标识要导出集群的哪些部分，请为 Identifiers（标识符）输入一个或多个数据库、模式或表，以空格分隔。

  使用以下格式：

  ```
  database[.schema][.table] database2[.schema2][.table2] ... databasen[.scheman]
  [.tablen]
  ```

  例如：

  ```
  mydatabase mydatabase2.myschema1 mydatabase2.myschema2.mytable1
   mydatabase2.myschema2.mytable2
  ```

7. 对于 S3 bucket (S3 存储桶)，选择要导出到的存储桶。

   要将导出的数据分配给 S3 存储桶中的文件夹路径，请为 S3 prefix (S3 前缀) 输入可选路径。

8. 对于 IAM role (IAM 角色)，请选择一个角色以授予您对所选 S3 存储桶的写入访问权限，或创建新角色。

   - 如果您按照 使用 IAM 角色提供对 Amazon S3 存储桶的访问权限中的步骤创建了角色，请选择该角色。
   - 如果您没有创建授予您对所选 S3 桶的写入访问权限的角色，则选择 Create a new role（创建新角色）来自动创建该角色。接下来，在 IAM role name (IAM 角色名称) 中输入角色的名称。

9. 对于 KMS key（KMS 密钥），输入要用于加密导出数据的密钥的 ARN。
10. 选择 Export to Amazon S3 (导出到 Amazon S3)。

AWS CLI

要使用 AWS CLI 将数据库集群数据导出到 Amazon S3，请使用包含以下所需选项的 start-export-task 命令：

- `--export-task-identifier`
- `--source-arn` – 数据库集群的 Amazon 资源名称（ARN）
- `--s3-bucket-name`
- `--iam-role-arn`

- `--kms-key-id`

在以下示例中，导出任务名为 *my-cluster-export*，该任务将数据导出到名为 *amzn-s3-demo-destination-bucket* 的 S3 存储桶。

Example

对于 Linux、macOS 或 Unix：

```
aws rds start-export-task \
    --export-task-identifier my-cluster-export \
    --source-arn arn:aws:rds:us-west-2:123456789012:cluster:my-cluster \
    --s3-bucket-name amzn-s3-demo-destination-bucket \
    --iam-role-arn iam-role \
    --kms-key-id my-key
```

对于 Windows：

```
aws rds start-export-task ^
    --export-task-identifier my-DB-cluster-export ^
    --source-arn arn:aws:rds:us-west-2:123456789012:cluster:my-cluster ^
    --s3-bucket-name amzn-s3-demo-destination-bucket ^
    --iam-role-arn iam-role ^
    --kms-key-id my-key
```

示例输出如下。

```
{
    "ExportTaskIdentifier": "my-cluster-export",
    "SourceArn": "arn:aws:rds:us-west-2:123456789012:cluster:my-cluster",
    "S3Bucket": "amzn-s3-demo-destination-bucket",
    "IamRoleArn": "arn:aws:iam:123456789012:role/ExportTest",
    "KmsKeyId": "my-key",
    "Status": "STARTING",
    "PercentProgress": 0,
    "TotalExtractedDataInGB": 0,
}
```

要在 S3 桶中为数据库集群导出提供文件夹路径，请在 start-export-task 命令中包含 `--s3-prefix` 选项。

RDS API

要使用 Amazon RDS API 将数据库集群数据导出到 Amazon S3，请使用包含以下所需参数的 StartExportTask 操作：

- `ExportTaskIdentifier`
- `SourceArn` – 数据库集群的 ARN
- `S3BucketName`
- `IamRoleArn`
- `KmsKeyId`

监控数据库集群导出任务

您可以使用 AWS Management Console、AWS CLI 或 RDS API 监控数据库集群导出。

控制台

监控数据库集群导出

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Exports in Amazon S3（Amazon S3 中的导出）。

   数据库集群导出在 Source type（源类型）列中指示。导出状态显示在 Status（状态）列中。
3. 要查看有关特定数据库集群导出的详细信息，请选择导出任务。

AWS CLI

要使用 AWS CLI 监控数据库集群导出任务，请使用 describe-export-tasks 命令。

以下示例说明如何显示有关所有数据库集群导出的当前信息。

Example

```
aws rds describe-export-tasks

{
    "ExportTasks": [
        {
```

```
            "Status": "CANCELED",
            "TaskEndTime": "2022-11-01T17:36:46.961Z",
            "S3Prefix": "something",
            "S3Bucket": "amzn-s3-demo-bucket",
            "PercentProgress": 0,
            "KmsKeyId": "arn:aws:kms:us-west-2:123456789012:key/K7MDENG/
bPxRfiCYEXAMPLEKEY",
            "ExportTaskIdentifier": "anewtest",
            "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
            "TotalExtractedDataInGB": 0,
            "SourceArn": "arn:aws:rds:us-west-2:123456789012:cluster:parameter-groups-
test"
        },
{
            "Status": "COMPLETE",
            "TaskStartTime": "2022-10-31T20:58:06.998Z",
            "TaskEndTime": "2022-10-31T21:37:28.312Z",
            "WarningMessage": "{\"skippedTables\":[],\"skippedObjectives\":[],\"general
\":[{\"reason\":\"FAILED_TO_EXTRACT_TABLES_LIST_FOR_DATABASE\"}]}",
            "S3Prefix": "",
            "S3Bucket": "amzn-s3-demo-bucket1",
            "PercentProgress": 100,
            "KmsKeyId": "arn:aws:kms:us-west-2:123456789012:key/2Zp9Utk/
h3yCo8nvbEXAMPLEKEY",
            "ExportTaskIdentifier": "thursday-events-test",
            "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
            "TotalExtractedDataInGB": 263,
            "SourceArn": "arn:aws:rds:us-
west-2:123456789012:cluster:example-1-2019-10-31-06-44"
        },
        {
            "Status": "FAILED",
            "TaskEndTime": "2022-10-31T02:12:36.409Z",
            "FailureCause": "The S3 bucket amzn-s3-demo-bucket2 isn't located in the
 current AWS Region. Please, review your S3 bucket name and retry the export.",
            "S3Prefix": "",
            "S3Bucket": "amzn-s3-demo-bucket2",
            "PercentProgress": 0,
            "KmsKeyId": "arn:aws:kms:us-west-2:123456789012:key/2Zp9Utk/
h3yCo8nvbEXAMPLEKEY",
            "ExportTaskIdentifier": "wednesday-afternoon-test",
            "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
            "TotalExtractedDataInGB": 0,
```

```
            "SourceArn": "arn:aws:rds:us-
west-2:123456789012:cluster:example-1-2019-10-30-06-45"
        }
    ]
}
```

要显示有关特定导出任务的信息，请在 `describe-export-tasks` 命令中包含 `--export-task-identifier` 选项。要筛选输出，请包括 `--Filters` 选项。有关更多选项，请参阅 describe-export-tasks 命令。

RDS API

要使用 Amazon RDS API 显示有关数据库集群导出的信息，请使用 DescribeExportTasks 操作。

要跟踪导出工作流的完成情况或启动其他工作流，您可以订阅 Amazon Simple Notification Service 主题。有关 Amazon SNS 的更多信息，请参阅 使用 Amazon RDS 事件通知。

取消数据库集群导出任务

您可以使用 AWS Management Console、AWS CLI 或 RDS API 取消数据库集群导出任务。

> **Note**
>
> 取消导出任务不会删除导出到 Amazon S3 的任何数据。有关如何使用控制台删除数据的信息，请参阅如何从 S3 存储桶删除对象？ 要使用 CLI 删除数据，请使用 delete-object 命令。

控制台

取消数据库集群导出任务

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Exports in Amazon S3（Amazon S3 中的导出）。

   数据库集群导出在 Source type（源类型）列中指示。导出状态显示在 Status（状态）列中。
3. 选择要取消的导出任务。
4. 选择 Cancel (取消)。
5. 在确认页面上选择 Cancel export task (取消导出任务)。

AWS CLI

要使用 AWS CLI 取消导出任务，请使用 cancel-export-task 命令。该命令需要 `--export-task-identifier` 选项。

Example

```
aws rds cancel-export-task --export-task-identifier my-export
{
    "Status": "CANCELING",
    "S3Prefix": "",
    "S3Bucket": "amzn-s3-demo-bucket",
    "PercentProgress": 0,
    "KmsKeyId": "arn:aws:kms:us-west-2:123456789012:key/K7MDENG/bPxRfiCYEXAMPLEKEY",
    "ExportTaskIdentifier": "my-export",
    "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
    "TotalExtractedDataInGB": 0,
    "SourceArn": "arn:aws:rds:us-west-2:123456789012:cluster:export-example-1"
}
```

RDS API

要使用 Amazon RDS API 取消导出任务，请使用带 `ExportTaskIdentifier` 参数的 CancelExportTask 操作。

数据库集群导出故障排除

使用以下各节来协助排查将数据库集群导出到 Amazon S3 的任务的失败消息和 PostgreSQL 权限错误。

Amazon S3 导出任务的失败消息

下表描述了 Amazon S3 导出任务失败时返回的消息。

失败消息	描述
无法找到或访问源数据库集群：[集群名称]	无法克隆源数据库集群。
出现未知的内部错误。	由于未知错误、异常或故障导致任务失败。

失败消息	描述
将导出任务的元数据写入 S3 存储桶 [存储桶名称] 时出现未知的内部错误。	由于未知错误、异常或故障导致任务失败。
RDS 导出无法编写导出任务的元数据，因为它无法担任 IAM 角色 [角色 ARN]。	导出任务将担任您的 IAM 角色来验证是否允许向 S3 存储桶写入元数据。如果任务无法担任您的 IAM 角色，它将失败。
RDS 导出未能使用带有 KMS 密钥 [密钥 ID] 的 IAM 角色 [角色 ARN] 将导出任务的元数据写入 S3 存储桶 [存储桶名称]。错误代码：[错误代码]	缺少一个或多个权限，因此导出任务无法访问 S3 存储桶。收到以下错误代码之一时，会引发此失败消息： • 带有错误代码 `AccessDenied` 的 `AWSSecurityTokenServiceException` • 带有错误代码 `NoSuchBucket` 、`AccessDenied` 、`KMS.KMSInvalidStateException` 、`403 Forbidden` 或 `KMS.DisabledException` 的 `AmazonS3Exception` 这些错误代码表示 IAM 角色、S3 桶或 KMS 密钥的设置出现配置错误。
IAM 角色 [角色 ARN] 无权在 S3 存储桶 [存储桶名称] 上调用 [S3 操作]。查看您的权限并重试导出。	IAM 策略配置错误。缺少对 S3 桶执行特定 S3 操作的权限，这会导致导出任务失败。
KMS 密钥检查失败。检查 KMS 密钥上的凭证然后重试。	KMS 密钥凭证检查失败。
S3 凭证检查失败。检查 S3 存储桶和 IAM 策略的权限。	S3 凭证检查失败。
S3 存储桶 [存储桶名称] 无效。它不在当前 AWS 区域 中，或者它不存在。检查 S3 存储桶名称，然后重试导出。	S3 存储桶无效。
S3 桶 [桶名称] 不在当前 AWS 区域 中。检查 S3 存储桶名称，然后重试导出。	S3 桶处于错误的 AWS 区域 中。

排查 PostgreSQL 权限错误

将 PostgreSQL 数据库导出到 Amazon S3 时，您可能会看到 PERMISSIONS_DO_NOT_EXIST 错误，指出已跳过某些表。当您在创建数据库集群时指定的超级用户无权访问这些表时，通常会发生此错误。

要修复此错误，请运行以下命令：

```
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA schema_name TO superuser_name
```

有关超级用户权限的更多信息，请参阅 主用户账户权限。

将数据库集群快照数据导出到 Amazon S3

您可以将数据库集群快照数据导出到 Amazon S3 存储桶。导出过程在后台运行，不会影响活动数据库集群的性能。

导出数据库集群快照时，Amazon Aurora 从快照中提取数据并将其存储在 Amazon S3 存储桶中。您可以导出手动快照和自动系统快照。默认情况下，将导出快照中的所有数据。但是，您可以选择导出特定的一组数据库、方案或表。

> **Note**
>
> 从数据库集群快照导出数据要求还原快照。还原时间受多种因素的影响，例如 AWS 区域相对于其可用带宽接收的网络流量。当流量突然增加时，完成时间可能比预期更长。
> 缩短 Aurora 数据库的 S3 导出时间的另一种方法是将数据库集群实时导出到 S3。数据库集群导出的开始时间比数据库快照导出更短，因为无需还原快照。有关更多信息，请参阅 将数据库集群数据导出到 Amazon S3。

数据以压缩和一致的 Apache Parquet 格式存储。各个 Parquet 文件的大小通常约为 1-10MB。

导出数据后，您可以通过 Amazon Athena 或 Amazon Redshift Spectrum 等工具直接分析导出的数据。有关使用 Athena 读取 Parque 数据的更多信息，请参阅 Amazon Athena 用户指南中的 Parquet SerDe。有关使用 Redshift Spectrum 读取 Parquet 数据的更多信息，请参阅《Amazon Redshift 数据库开发人员指南》中的从列式数据格式执行 COPY 操作。

功能可用性和支持因每个数据库引擎的特定版本以及 AWS 区域而异。有关将数据库集群快照数据导出到 S3 的版本和区域可用性的更多信息，请参阅 支持将快照数据导出到 Amazon S3 的区域和 Aurora 数据库引擎。

您可以使用以下过程将数据库快照数据导出到 Amazon S3 存储桶。有关更多详细信息，请参阅以下部分。

导出快照数据概述

1. 确定要导出的快照。

   使用现有的自动快照或手动快照，或创建数据库实例的手动快照。
2. 设置对 Amazon S3 存储桶的访问权限。

   存储桶是 Amazon S3 对象或文件的容器。要提供访问存储桶的信息，请执行以下步骤：

a. 标识要将快照导出到的 S3 存储桶。S3 存储桶必须与快照位于同一 AWS 区域。有关更多信息，请参阅 标识要导出到的 Amazon S3 存储桶。

b. 创建一个 AWS Identity and Access Management (IAM) 角色，用于授予快照导出任务对 S3 存储桶的访问权限。有关更多信息，请参阅 使用 IAM 角色提供对 Amazon S3 存储桶的访问权限。

3. 创建对称加密 AWS KMS key 以进行服务器端加密。快照导出任务使用 KMS 密钥在将导出数据写入 S3 时设置 AWS KMS 服务器端加密。

   KMS 密钥策略必须同时包含 `kms:CreateGrant` 和 `kms:DescribeKey` 权限。有关在 Amazon Aurora 中使用 KMS 密钥的更多信息，请参阅 AWS KMS key 管理。

   如果 KMS 密钥策略中有拒绝语句，则确保显式排除 AWS 服务主体 `export.rds.amazonaws.com`。

   您可以在您的 AWS 账户内使用 KMS 密钥，或者您可以使用跨账户 KMS 密钥。有关更多信息，请参阅 使用跨账户 AWS KMS key。

4. 使用控制台或 `start-export-task` CLI 命令将快照导出到 Amazon S3。有关更多信息，请参阅 创建快照导出任务。

5. 要访问 Amazon S3 存储桶中导出的数据，请参阅 Amazon Simple Storage Service 用户指南中的上传、下载和管理对象。

在以下各节中，学习设置、导出、监控、取消数据库集群快照导出任务并对其进行故障排除。

主题

- 数据库集群快照导出注意事项
- 设置 Amazon S3 存储桶的访问权限
- 创建快照导出任务
- 监控快照导出
- 取消快照导出任务
- Aurora MySQL 中的导出性能
- 快照导出故障排除

数据库集群快照导出注意事项

限制

将数据库快照数据导出到 Amazon S3 有以下限制：

- 您不能为同一个数据库集群快照同时运行多个导出任务。这同时适用于完全导出和部分导出。
- 每个 AWS 账户 最多可以执行五个并发数据库快照导出任务。
- 您不能将快照数据从 Aurora Serverless v1 数据库集群导出到 S3。
- 导出到 S3 不支持包含冒号 (:) 的 S3 前缀。
- 在导出过程中，S3 文件路径中的以下字符将转换为下划线 (_)：

```
\ ` " (space)
```

- 如果数据库、架构或表的名称中包含以下字符以外的字符，则不支持部分导出。但是，您可以导出整个数据库快照。
  - 拉丁字母 (A–Z)
  - 数字 (0–9)
  - 美元符号 ($)
  - 下划线 (_)
- 数据库表列名不支持空格 ( ) 和某些字符。在导出过程中会跳过列名中包含以下字符的表：

```
, ; { } ( ) \n \t = (space)
```

- 在导出过程中会跳过其名称中包含斜杠 (/) 的表。
- 在导出期间，将跳过 Aurora PostgreSQL 临时表和未记录的表。
- 如果数据包含接近或大于 500MB 的大型对象（例如 BLOB 或 CLOB），则导出失败。
- 如果表中某个大行的大小接近或大于 2GB，则会在导出过程中略过该表。
- 对于部分导出，`ExportOnly` 列表的最大大小为 200 KB。
- 强烈建议您为每个导出任务使用唯一的名称。如果您没有使用唯一的任务名称，可能会收到以下错误消息：

  ExportTaskAlreadyExistsFault：调用 StartExportTask 操作时发生错误 (ExportTaskAlreadyExists)：ID 为 *xxxxx* 的导出任务已存在。
- 您可以在将快照数据导出到 S3 时删除快照，但是在导出任务完成之前，仍需支付该快照的存储成本。

- 您无法将从 S3 导出的快照数据恢复到新的数据库集群。

文件命名约定

特定表的导出数据以 *base_prefix*/*files* 格式存储，基本前缀如下：

```
export_identifier/database_name/schema_name.table_name/
```

例如：

```
export-1234567890123-459/rdststdb/rdststdb.DataInsert_7ADB5D19965123A2/
```

文件的命名方式有两种约定。

- 当前约定：

  ```
  batch_index/part-partition_index-random_uuid.format-based_extension
  ```

  批量索引是一个序列号，表示从表中读取的一批数据。如果我们无法将您的表分区成小块以并行导出，则会有多个批量索引。如果表分区成多个表，也会发生同样的情况。这会出现多个批量索引，主表的每个表分区对应一个。

  如果我们可以将表分区成小块以并行读取，那么就只有批量索引 1 文件夹。

  在批量索引文件夹中，有一个或多个包含表数据的 Parquet 文件。Parquet 文件名的前缀是 part-*partition_index*。如果您的表已分区，则会有多个以分区索引 00000 开头的文件。

  分区索引序列中可能存在间隙。之所以发生这种情况，是因为每个分区都是从表中的范围查询中获得的。如果该分区的范围内没有数据，则跳过该序列号。

  例如，假设 id 列是表的主键，其最小值和最大值为 100 和 1000。当我们尝试导出这个带有九个分区的表时，我们会使用并行查询读取它，如下所示：

  ```
  SELECT * FROM table WHERE id <= 100 AND id < 200
   SELECT * FROM table WHERE id <= 200 AND id < 300
  ```

  这应该生成九个文件，从 part-00000-*random_uuid*.gz.parquet 到 part-00008-*random_uuid*.gz.parquet。但是，如果没有 ID 介于 200 和 350 之间的

行，则其中一个已完成的分区为空，并且不会为其创建任何文件。在前面的示例中，未创建 part-00001-*random_uuid*.gz.parquet。

- 较早的约定：

```
part-partition_index-random_uuid.format-based_extension
```

这与当前约定相同，但没有 *batch_index* 前缀，例如：

```
part-00000-c5a881bb-58ff-4ee6-1111-b41ecff340a3-c000.gz.parquet
 part-00001-d7a881cc-88cc-5ab7-2222-c41ecab340a4-c000.gz.parquet
 part-00002-f5a991ab-59aa-7fa6-3333-d41eccd340a7-c000.gz.parquet
```

文件命名约定可能会更改。因此，在读取目标表时，我们建议您读取表的基本前缀内的所有内容。

导出到 Amazon S3 存储桶时的数据转换

将数据库快照导出到 Amazon S3 存储桶时，Amazon Aurora 以 Parquet 格式转换数据、导出数据并存储数据。有关 Parquet 的更多信息，请参阅 Apache Parquet 网站。

Parquet 将所有数据存储为以下原始类型之一：

- BOOLEAN
- INT32
- INT64
- INT96
- FLOAT
- DOUBLE
- BYTE_ARRAY – 一个可变长度的字节数组，也称为二进制
- FIXED_LEN_BYTE_ARRAY – 当值具有恒定大小时使用的固定长度字节数组

Parquet 数据类型很少能减少读取和写入格式的复杂性。Parquet 提供了用于扩展原始类型的逻辑类型。逻辑类型实现为具有 `LogicalType` 元数据字段中数据的注释。逻辑类型注释说明如何解释原始类型。

当 STRING 逻辑类型注释 BYTE_ARRAY 类型时，它表示字节数组应被解释为 UTF-8 编码的字符串。导出任务完成后，如果发生了任何字符串转换，则 Amazon Aurora 会通知您。导出的基础数据始终与源中的数据相同。但是，由于 UTF-8 中的编码差异，在工具（如 Athena）中读取时，某些字符可能会显示与源不同。

有关更多信息，请参阅 Parquet 文档中的 Parquet 逻辑类型定义。

主题

- MySQL 数据类型到 Parquet 的映射
- PostgreSQL 数据类型到 Parquet 的映射

MySQL 数据类型到 Parquet 的映射

下表显示在将数据转换并导出到 Amazon S3 时从 MySQL 数据类型到 Parquet 数据类型的映射。

源数据类型	Parquet 原始类型	逻辑类型注释	转换说明
数字数据类型			
BIGINT	INT64		
BIGINT UNSIGNED	FIXED_LEN_BYTE_ARRAY(9)	DECIMAL(20,0)	Parquet 仅支持签名类型，因此映射需要额外的字节（8 加 1）来存储 BIGINT_UNSIGNED 类型。
BIT	BYTE_ARRAY		
DECIMAL	INT32	DECIMAL (p,s)	如果源值小于 2^{31}，它其存储为 INT32。
	INT64	DECIMAL (p,s)	如果源值等于或大于 2^{31}，但小于 2^{63}，它将存储为 INT64。
	FIXED_LEN_BYTE_ARRAY(N)	DECIMAL (p,s)	如果源值等于或大于 2^{63}，则将它存

源数据类型	Parquet 原始类型	逻辑类型注释	转换说明
			储为 FIXED_LEN _BYTE_ARRAY(N)。
	BYTE_ARRAY	STRING	Parquet 不支持大于 38 的小数精度。十进制值转换为 BYTE_ARRAY 类型的字符串，并编码为 UTF8。
DOUBLE	DOUBLE		
FLOAT	DOUBLE		
INT	INT32		
INT UNSIGNED	INT64		
MEDIUMINT	INT32		
MEDIUMINT UNSIGNED	INT64		
NUMERIC	INT32	DECIMAL (p,s)	如果源值小于 2^{31}，它其存储为 INT32。
	INT64	DECIMAL (p,s)	如果源值等于或大于 2^{31}，但小于 2^{63}，它将存储为 INT64。
	FIXED_LEN _ARRAY(N)	DECIMAL (p,s)	如果源值等于或大于 2^{63}，则将它存储为 FIXED_LEN _BYTE_ARRAY(N)。

源数据类型	Parquet 原始类型	逻辑类型注释	转换说明
	BYTE_ARRAY	STRING	Parquet 不支持大于 38 的数值精度。此数值类型的值转换为 BYTE_ARRAY 类型的字符串，并编码为 UTF8。
SMALLINT	INT32		
SMALLINT UNSIGNED	INT32		
TINYINT	INT32		
TINYINT UNSIGNED	INT32	INT(16, true)	
字符串数据类型			
BINARY	BYTE_ARRAY		
BLOB	BYTE_ARRAY		
CHAR	BYTE_ARRAY		
ENUM	BYTE_ARRAY	STRING	
LINESTRING	BYTE_ARRAY		
LONGBLOB	BYTE_ARRAY		
LONGTEXT	BYTE_ARRAY	STRING	
MEDIUMBLOB	BYTE_ARRAY		
MEDIUMTEXT	BYTE_ARRAY	STRING	
MULTILINESTRING	BYTE_ARRAY		
SET	BYTE_ARRAY	STRING	

源数据类型	Parquet 原始类型	逻辑类型注释	转换说明
TEXT	BYTE_ARRAY	STRING	
TINYBLOB	BYTE_ARRAY		
TINYTEXT	BYTE_ARRAY	STRING	
VARBINARY	BYTE_ARRAY		
VARCHAR	BYTE_ARRAY	STRING	
日期和时间数据类型			
DATE	BYTE_ARRAY	STRING	日期将转换为 BYTE_ARRAY 类型的字符串，并编码为 UTF8。
DATETIME	INT64	TIMESTAMP_MICROS	
TIME	BYTE_ARRAY	STRING	TIME 类型转换为 BYTE_ARRAY 类型的字符串，并编码为 UTF8。
TIMESTAMP	INT64	TIMESTAMP_MICROS	
YEAR	INT32		
几何数据类型			
GEOMETRY	BYTE_ARRAY		
GEOMETRYCOLLECTION	BYTE_ARRAY		
MULTIPOINT	BYTE_ARRAY		

源数据类型	Parquet 原始类型	逻辑类型注释	转换说明
MULTIPOLYGON	BYTE_ARRAY		
POINT	BYTE_ARRAY		
POLYGON	BYTE_ARRAY		
JSON 数据类型			
JSON	BYTE_ARRAY	STRING	

PostgreSQL 数据类型到 Parquet 的映射

下表显示在将数据转换并导出到 Amazon S3 时从 PostgreSQL 数据类型到 Parquet 数据类型的映射。

PostgreSQL 数据类型	Parquet 原始类型	逻辑类型注释	映射注释
数字数据类型			
BIGINT	INT64		
BIGSERIAL	INT64		
DECIMAL	BYTE_ARRAY	STRING	DECIMAL 类型转换为 BYTE_ARRAY 类型的字符串，并编码为 UTF8。 此转换是为了避免由于数据精度和非数字 (NaN) 的数据值而引起的复杂性。
DOUBLE PRECISION	DOUBLE		
INTEGER	INT32		
MONEY	BYTE_ARRAY	STRING	

PostgreSQL 数据类型	Parquet 原始类型	逻辑类型注释	映射注释
REAL	FLOAT		
SERIAL	INT32		
SMALLINT	INT32	INT(16, true)	
SMALLSERIAL	INT32	INT(16, true)	
字符串和相关数据类型			
ARRAY	BYTE_ARRAY	STRING	数组转换为字符串并编码为 BINARY (UTF8)。 此转换是为了避免因数据精度、非数字 (NaN) 的数据值和时间数据值而产生的复杂性。
BIT	BYTE_ARRAY	STRING	
BIT VARYING	BYTE_ARRAY	STRING	
BYTEA	BINARY		
CHAR	BYTE_ARRAY	STRING	
CHAR(N)	BYTE_ARRAY	STRING	
ENUM	BYTE_ARRAY	STRING	
NAME	BYTE_ARRAY	STRING	
TEXT	BYTE_ARRAY	STRING	
文本搜索	BYTE_ARRAY	STRING	

PostgreSQL 数据类型	Parquet 原始类型	逻辑类型注释	映射注释
VARCHAR(N)	BYTE_ARRAY	STRING	
XML	BYTE_ARRAY	STRING	
日期和时间数据类型			
DATE	BYTE_ARRAY	STRING	
INTERVAL	BYTE_ARRAY	STRING	
TIME	BYTE_ARRAY	STRING	
带时区的时间	BYTE_ARRAY	STRING	
TIMESTAMP	BYTE_ARRAY	STRING	
TIMESTAMP（有时区）	BYTE_ARRAY	STRING	
几何数据类型			
BOX	BYTE_ARRAY	STRING	
CIRCLE	BYTE_ARRAY	STRING	
LINE	BYTE_ARRAY	STRING	
LINESEGMENT	BYTE_ARRAY	STRING	
路径	BYTE_ARRAY	STRING	
POINT	BYTE_ARRAY	STRING	
POLYGON	BYTE_ARRAY	STRING	
JSON 数据类型			
JSON	BYTE_ARRAY	STRING	
JSONB	BYTE_ARRAY	STRING	

PostgreSQL 数据类型	Parquet 原始类型	逻辑类型注释	映射注释
其他数据类型			
BOOLEAN	BOOLEAN		
CIDR	BYTE_ARRAY	STRING	网络数据类型
COMPOSITE	BYTE_ARRAY	STRING	
DOMAIN	BYTE_ARRAY	STRING	
INET	BYTE_ARRAY	STRING	网络数据类型
MACADDR	BYTE_ARRAY	STRING	
对象标识符	不适用		
PG_LSN	BYTE_ARRAY	STRING	
RANGE	BYTE_ARRAY	STRING	
UUID	BYTE_ARRAY	STRING	

设置 Amazon S3 存储桶的访问权限

您识别 AmazonS3 桶，然后授予快照访问它的权限。

主题

- 标识要导出到的 Amazon S3 存储桶
- 使用 IAM 角色提供对 Amazon S3 存储桶的访问权限
- 使用跨账户 Amazon S3 存储桶
- 使用跨账户 AWS KMS key

标识要导出到的 Amazon S3 存储桶

标识要将数据库快照导出到的 Amazon S3 存储桶。使用现有 S3 存储桶或创建新的 S3 存储桶。

> **Note**
>
> 要导出到的 S3 存储桶必须与快照位于同一 AWS 区域中。

有关使用 Amazon S3 存储桶的详细信息，请参阅 Amazon Simple Storage Service 用户指南中的以下主题：

- 如何查看 S3 存储桶的属性？
- 如何为 Amazon S3 存储桶启用默认加密？
- 如何创建 S3 存储桶？

使用 IAM 角色提供对 Amazon S3 存储桶的访问权限

将数据库快照数据导出到 Amazon S3 之前，请授予快照导出任务对 Amazon S3 存储桶的写入访问权限。

要授予此权限，请创建一个 IAM 策略以提供对桶的访问权限，然后创建一个 IAM 角色并将该策略附加到该角色。稍后，您可以将此 IAM 角色分配给快照导出任务。

> **Important**
>
> 如果计划使用AWS Management Console导出快照，则可以选择在导出快照时自动创建 IAM 策略和角色。有关说明，请参阅创建快照导出任务。

授予数据库快照任务访问 Amazon S3 的权限

1. 创建一个 IAM 策略。此策略提供允许快照导出任务访问 Amazon S3 的存储桶和对象权限。

   在策略中，包含以下必需操作，以允许将文件从 Amazon Aurora 桶传输到 S3 桶：

   - `s3:PutObject*`
   - `s3:GetObject*`
   - `s3:ListBucket`
   - `s3:DeleteObject*`
   - `s3:GetBucketLocation`

在策略中，包含以下资源以标识 S3 桶以及该桶中的对象。以下资源列表显示用于访问 Amazon S3 的 Amazon Resource Name (ARN) 格式。

- `arn:aws:s3:::amzn-s3-demo-bucket`
- `arn:aws:s3:::amzn-s3-demo-bucket/*`

有关为 Amazon Aurora 创建 IAM 策略的更多信息，请参阅创建和使用适用于 IAM 数据库访问的 IAM 策略。另请参阅 IAM 用户指南中的教程：创建和附加您的第一个客户托管式策略。

以下 AWS CLI 命令使用这些选项创建一个名为 `ExportPolicy` 的 IAM 策略。该策略授予对名为 *amzn-s3-demo-bucket* 的存储桶的访问权限。

> **Note**
>
> 创建策略后，请记下策略的 ARN。在将策略附加到 IAM 角色时，您在后面的步骤中需要使用 ARN。

```
aws iam create-policy  --policy-name ExportPolicy --policy-document '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ExportPolicy",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject*",
                "s3:ListBucket",
                "s3:GetObject*",
                "s3:DeleteObject*",
                "s3:GetBucketLocation"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket",
                "arn:aws:s3:::amzn-s3-demo-bucket/*"
            ]
        }
    ]
}'
```

2. 创建一个 IAM 角色，以便 Aurora 可以代入该 IAM 角色，代表您访问 Amazon S3 桶。有关更多信息，请参阅 IAM 用户指南中的创建向 IAM 用户委派权限的角色。

   以下示例说明了如何使用 AWS CLI 命令创建一个名为 `rds-s3-export-role` 的角色。

```
aws iam create-role  --role-name rds-s3-export-role  --assume-role-policy-document
 '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Principal": {
            "Service": "export.rds.amazonaws.com"
          },
         "Action": "sts:AssumeRole"
       }
     ]
   }'
```

3. 将您创建的 IAM 策略附加到您创建的 IAM 角色。

   以下 AWS CLI 命令将之前创建的策略附加到名为 `rds-s3-export-role` 的角色。将 *`your-policy-arn`* 替换为您在先前步骤中记下的策略 ARN。

```
aws iam attach-role-policy  --policy-arn your-policy-arn  --role-name rds-s3-
export-role
```

使用跨账户 Amazon S3 存储桶

您可以跨 AWS 账户使用 Amazon S3 存储桶。要使用跨账户存储桶，请添加存储桶策略以允许访问您用于 S3 导出的 IAM 角色。有关更多信息，请参阅示例 2：存储桶拥有者授予跨账户存储桶权限。

- 将存储桶策略附加到存储桶，如下面的示例所示。

  JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
```

```
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:role/Admin"
            },
            "Action": [
                "s3:PutObject*",
                "s3:ListBucket",
                "s3:GetObject*",
                "s3:DeleteObject*",
                "s3:GetBucketLocation"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-destination-bucket",
                "arn:aws:s3:::amzn-s3-demo-destination-bucket/*"
            ]
        }
    ]
}
```

使用跨账户 AWS KMS key

您可以使用跨账户 AWS KMS key 以加密 Amazon S3 导出的内容。首先，向本地账户添加密钥策略，然后在外部账户中添加 IAM 策略。有关更多信息，请参阅允许其他账户中的用户使用 KMS 密钥。

要使用跨账户 KMS 密钥

1. 向本地账户添加密钥策略。

   以下示例为外部账户 444455556666 中的 `ExampleRole` 和 `ExampleUser` 提供了内部账户 123456789012 中的权限。

```
{
    "Sid": "Allow an external account to use this KMS key",
    "Effect": "Allow",
    "Principal": {
        "AWS": [
            "arn:aws:iam::444455556666:role/ExampleRole",
            "arn:aws:iam::444455556666:user/ExampleUser"
        ]
    },
    "Action": [
```

```
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:CreateGrant",
        "kms:DescribeKey",
        "kms:RetireGrant"
    ],
    "Resource": "*"
}
```

2. 在外部账户中添加 IAM 策略。

   以下示例 IAM 策略允许主体使用账户 123456789012 中的 KMS 密钥执行加密操作。要向账户 444455556666 中的 `ExampleRole` 和 `ExampleUser` 授予此权限，请将策略附加到该账户中的用户或角色。

```
{
    "Sid": "Allow use of KMS key in account 123456789012",
    "Effect": "Allow",
    "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:CreateGrant",
        "kms:DescribeKey",
        "kms:RetireGrant"
    ],
    "Resource": "arn:aws:kms:us-
west-2:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"
}
```

创建快照导出任务

创建快照导出任务，来将数据从快照导出到 Amazon S3 存储桶。每个 AWS 账户 最多可以执行五个并发数据库快照导出任务。

> **Note**
>
> 导出 RDS 快照可能需要一段时间，具体取决于您的数据库类型和大小。导出任务首先还原并扩展整个数据库，然后再将数据提取到 Amazon S3。此阶段的任务进度显示为正在启动。当任务切换到将数据导出到 S3 时，进度显示为正在进行。
> 完成导出所需的时间取决于数据库中存储的数据。例如，具有分布良好的数字主键或索引列的表导出速度最快。不包含适用于分区的列的表，以及只有基于字符串的列上的一个索引的表将需要更长时间。导出时间之所以更长，是因为导出使用较慢的单线程进程。

您可以使用AWS Management Console、AWS CLI 或 RDS API 将数据库快照导出到 Amazon S3。

如果您使用 Lambda 函数导出快照，请将 `kms:DescribeKey` 操作添加到 Lambda 函数策略中。有关更多信息，请参阅 AWS Lambda 权限。

控制台

仅为可导出到 Amazon S3 的快照显示导出到 Amazon S3 控制台选项。由于以下原因，快照可能无法导出：

- 不支持数据库引擎的 S3 导出。
- 不支持数据库实例版本的 S3 导出。
- 创建了快照的 AWS 区域不支持 S3 导出。

导出数据库快照

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择快照。
3. 从选项卡中，选择要导出的快照类型。
4. 在快照列表中，选择要导出的快照。
5. 对于 Actions (操作)，选择 Export to Amazon S3 (导出到 Amazon S3)。

   此时将显示 Export to Amazon S3 (导出到 Amazon S3) 窗口。
6. 对于 Export identifier (导出标识符)，输入用于标识导出任务的名称。此值也用于在 S3 存储桶中创建的文件的名称。
7. 选择要导出的数据：

- 选择 All (全部) 可导出快照中的所有数据。
- 选择 Partial (部分) 可导出快照的特定部分。如需标识要导出快照的哪些部分，请为 Identifiers (标识符) (以空格分隔) 输入一个或多个数据库、架构或表。

  使用以下格式：

  ```
  database[.schema][.table] database2[.schema2][.table2] ... databasen[.scheman]
  [.tablen]
  ```

  例如：

  ```
  mydatabase mydatabase2.myschema1 mydatabase2.myschema2.mytable1
   mydatabase2.myschema2.mytable2
  ```

8. 对于 S3 bucket (S3 存储桶)，选择要导出到的存储桶。

   要将导出的数据分配给 S3 存储桶中的文件夹路径，请为 S3 prefix (S3 前缀) 输入可选路径。

9. 对于 IAM role (IAM 角色)，请选择一个角色以授予您对所选 S3 存储桶的写入访问权限，或创建新角色。

   - 如果您按照 使用 IAM 角色提供对 Amazon S3 存储桶的访问权限中的步骤创建了角色，请选择该角色。
   - 如果您没有创建授予您对所选 S3 桶的写入访问权限的角色，则选择 Create a new role (创建新角色) 来自动创建该角色。接下来，在 IAM role name (IAM 角色名称) 中输入角色的名称。

10. 对于 AWS KMS key，输入要用于加密导出数据的密钥的 ARN。
11. 选择 Export to Amazon S3 (导出到 Amazon S3)。

AWS CLI

要使用 AWS CLI 将数据库快照导出到 Amazon S3，请使用包含以下所需选项的 start-export-task 命令：

- `--export-task-identifier`
- `--source-arn`
- `--s3-bucket-name`
- `--iam-role-arn`

- `--kms-key-id`

在以下示例中，快照导出任务名为 *my-snapshot-export*，该任务将快照导出到名为 *amzn-s3-demo-destination-bucket* 的 S3 存储桶。

Example

对于 Linux、macOS 或 Unix：

```
aws rds start-export-task \
    --export-task-identifier my-snapshot-export \
    --source-arn arn:aws:rds:AWS_Region:123456789012:snapshot:snapshot-name \
    --s3-bucket-name amzn-s3-demo-destination-bucket \
    --iam-role-arn iam-role \
    --kms-key-id my-key
```

对于 Windows：

```
aws rds start-export-task ^
    --export-task-identifier my-snapshot-export ^
    --source-arn arn:aws:rds:AWS_Region:123456789012:snapshot:snapshot-name ^
    --s3-bucket-name amzn-s3-demo-destination-bucket ^
    --iam-role-arn iam-role ^
    --kms-key-id my-key
```

示例输出如下。

```
{
    "Status": "STARTING",
    "IamRoleArn": "iam-role",
    "ExportTime": "2019-08-12T01:23:53.109Z",
    "S3Bucket": "amzn-s3-demo-destination-bucket",
    "PercentProgress": 0,
    "KmsKeyId": "my-key",
    "ExportTaskIdentifier": "my-snapshot-export",
    "TotalExtractedDataInGB": 0,
    "TaskStartTime": "2019-11-13T19:46:00.173Z",
    "SourceArn": "arn:aws:rds:AWS_Region:123456789012:snapshot:snapshot-name"
}
```

要在 S3 存储桶中为快照导出提供文件夹路径，请在 start-export-task 命令中包含 `--s3-prefix` 选项。

RDS API

要使用 Amazon RDS API 将数据库快照导出到 Amazon S3，请使用包含以下所需参数的 StartExportTask 操作：

- `ExportTaskIdentifier`
- `SourceArn`
- `S3BucketName`
- `IamRoleArn`
- `KmsKeyId`

监控快照导出

您可以使用 AWS Management Console、AWS CLI 或 RDS API 监控数据库快照导出。

控制台

监视数据库快照导出

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Exports in Amazon S3（Amazon S3 中的导出）。

   数据库快照导出在 Source type（源类型）列中指示。导出状态显示在 Status（状态）列中。
3. 要查看有关特定快照导出的详细信息，请选择导出任务。

AWS CLI

要使用 AWS CLI 监控数据库快照导出，请使用 describe-export-tasks 命令。

以下示例说明如何显示有关所有快照导出的当前信息。

Example

```
aws rds describe-export-tasks
```

```
{
    "ExportTasks": [
        {
            "Status": "CANCELED",
            "TaskEndTime": "2019-11-01T17:36:46.961Z",
            "S3Prefix": "something",
            "ExportTime": "2019-10-24T20:23:48.364Z",
            "S3Bucket": "amzn-s3-demo-bucket",
            "PercentProgress": 0,
            "KmsKeyId": "arn:aws:kms:AWS_Region:123456789012:key/K7MDENG/
bPxRfiCYEXAMPLEKEY",
            "ExportTaskIdentifier": "anewtest",
            "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
            "TotalExtractedDataInGB": 0,
            "TaskStartTime": "2019-10-25T19:10:58.885Z",
            "SourceArn": "arn:aws:rds:AWS_Region:123456789012:snapshot:parameter-
groups-test"
        },
{
            "Status": "COMPLETE",
            "TaskEndTime": "2019-10-31T21:37:28.312Z",
            "WarningMessage": "{\"skippedTables\":[],\"skippedObjectives\":[],\"general
\":[{\"reason\":\"FAILED_TO_EXTRACT_TABLES_LIST_FOR_DATABASE\"}]}",
            "S3Prefix": "",
            "ExportTime": "2019-10-31T06:44:53.452Z",
            "S3Bucket": "amzn-s3-demo-bucket1",
            "PercentProgress": 100,
            "KmsKeyId": "arn:aws:kms:AWS_Region:123456789012:key/2Zp9Utk/
h3yCo8nvbEXAMPLEKEY",
            "ExportTaskIdentifier": "thursday-events-test",
            "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
            "TotalExtractedDataInGB": 263,
            "TaskStartTime": "2019-10-31T20:58:06.998Z",
            "SourceArn":
 "arn:aws:rds:AWS_Region:123456789012:snapshot:rds:example-1-2019-10-31-06-44"
        },
        {
            "Status": "FAILED",
            "TaskEndTime": "2019-10-31T02:12:36.409Z",
            "FailureCause": "The S3 bucket my-exports isn't located in the current AWS
 Region. Please, review your S3 bucket name and retry the export.",
            "S3Prefix": "",
            "ExportTime": "2019-10-30T06:45:04.526Z",
            "S3Bucket": "amzn-s3-demo-bucket2",
```

```
            "PercentProgress": 0,
            "KmsKeyId": "arn:aws:kms:AWS_Region:123456789012:key/2Zp9Utk/
h3yCo8nvbEXAMPLEKEY",
            "ExportTaskIdentifier": "wednesday-afternoon-test",
            "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
            "TotalExtractedDataInGB": 0,
            "TaskStartTime": "2019-10-30T22:43:40.034Z",
            "SourceArn":
 "arn:aws:rds:AWS_Region:123456789012:snapshot:rds:example-1-2019-10-30-06-45"
        }
    ]
}
```

要显示有关特定快照导出的信息，请在 `--export-task-identifier` 命令中包含 `describe-export-tasks` 选项。要筛选输出，请包括 `--Filters` 选项。有关更多选项，请参阅 describe-export-tasks 命令。

RDS API

要使用 Amazon RDS API 显示有关数据库快照导出的信息，请使用 DescribeExportTasks 操作。

要跟踪导出工作流的完成情况或启动其他工作流，您可以订阅 Amazon Simple Notification Service 主题。有关 Amazon SNS 的更多信息，请参阅 使用 Amazon RDS 事件通知。

取消快照导出任务

您可以使用 AWS Management Console、AWS CLI 或 RDS API 取消数据库快照导出任务。

> **Note**
>
> 取消快照导出任务不会删除导出到 Amazon S3 的任何数据。有关如何使用控制台删除数据的信息，请参阅如何从 S3 存储桶删除对象？ 要使用 CLI 删除数据，请使用 delete-object 命令。

控制台

取消快照导出任务

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Exports in Amazon S3（Amazon S3 中的导出）。

数据库快照导出在 Source type（源类型）列中指示。导出状态显示在 Status（状态）列中。

3. 选择要取消的快照导出任务。
4. 选择 Cancel (取消)。
5. 在确认页面上选择 Cancel export task (取消导出任务)。

AWS CLI

要使用 AWS CLI 取消快照导出任务，请使用 cancel-export-task 命令。该命令需要 `--export-task-identifier` 选项。

Example

```
aws rds cancel-export-task --export-task-identifier my_export
{
    "Status": "CANCELING",
    "S3Prefix": "",
    "ExportTime": "2019-08-12T01:23:53.109Z",
    "S3Bucket": "amzn-s3-demo-bucket",
    "PercentProgress": 0,
    "KmsKeyId": "arn:aws:kms:AWS_Region:123456789012:key/K7MDENG/bPxRfiCYEXAMPLEKEY",
    "ExportTaskIdentifier": "my_export",
    "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
    "TotalExtractedDataInGB": 0,
    "TaskStartTime": "2019-11-13T19:46:00.173Z",
    "SourceArn": "arn:aws:rds:AWS_Region:123456789012:snapshot:export-example-1"
}
```

RDS API

要使用 Amazon RDS API 取消快照导出任务，请使用带 `ExportTaskIdentifier` 参数的 CancelExportTask 操作。

Aurora MySQL 中的导出性能

Aurora MySQL 版本 2 和版本 3 数据库集群快照使用高级导出机制来提高性能并缩短导出时间。该机制包括诸如多个导出线程和 Aurora MySQL 并行查询等优化措施，以利用 Aurora 共享存储架构。优化措施以自适应方式应用，具体取决于数据集的大小和结构。

您不需要开启并行查询来使用更快的导出过程，但该过程确实具有与并行查询相同的限制。此外，不支持某些数据值，例如月中的某天为 `0` 或年份为 `0000` 的日期。有关更多信息，请参阅 Amazon Aurora MySQL 的并行查询。

应用性能优化后，您可能还会看到用于 Aurora MySQL 版本 2 和 3 导出的 Parquet 文件要大得多（大约 200GB）。

如果无法使用更快的导出流程，例如由于数据类型或值不兼容，Aurora 会自动切换到单线程导出模式而无需并行查询。根据使用的流程和要导出的数据量，导出性能可能会有所不同。

快照导出故障排除

使用以下各节来协助排查将数据库集群导出到 Amazon S3 的任务的失败消息和 PostgreSQL 权限错误。

Amazon S3 导出任务的失败消息

下表描述了 Amazon S3 导出任务失败时返回的消息。

失败消息	描述
出现未知的内部错误。	由于未知错误、异常或故障导致任务失败。
将导出任务的元数据写入 S3 存储桶 [存储桶名称] 时出现未知的内部错误。	由于未知错误、异常或故障导致任务失败。
RDS 导出无法编写导出任务的元数据，因为它无法担任 IAM 角色 [角色 ARN]。	导出任务将担任您的 IAM 角色来验证是否允许向 S3 存储桶写入元数据。如果任务无法担任您的 IAM 角色，它将失败。
RDS 导出未能使用带有 KMS 密钥 [密钥 ID] 的 IAM 角色 [角色 ARN] 将导出任务的元数据写入 S3 存储桶 [存储桶名称]。错误代码：[错误代码]	缺少一个或多个权限，因此导出任务无法访问 S3 存储桶。收到以下错误代码之一时，会引发此失败消息： • 带有错误代码 `AccessDenied` 的 `AWSSecurityTokenServiceException` • 带有错误代码 `NoSuchBucket`、`AccessDenied`、`KMS.KMSInvalidStateException`、`403 Forbidden` 或 `KMS.DisabledException` 的 `AmazonS3Exception`

失败消息	描述
	这些错误代码表示 IAM 角色、S3 桶或 KMS 密钥的设置出现配置错误。
IAM 角色 [角色 ARN] 无权在 S3 存储桶 [存储桶名称] 上调用 [S3 操作]。查看您的权限并重试导出。	IAM 策略配置错误。缺少对 S3 桶执行特定 S3 操作的权限，这会导致导出任务失败。
KMS 密钥检查失败。检查 KMS 密钥上的凭证然后重试。	KMS 密钥凭证检查失败。
S3 凭证检查失败。检查 S3 存储桶和 IAM 策略的权限。	S3 凭证检查失败。
S3 存储桶 [存储桶名称] 无效。它不在当前 AWS 区域 中，或者它不存在。检查 S3 存储桶名称，然后重试导出。	S3 存储桶无效。
S3 桶 [桶名称] 不在当前 AWS 区域 中。检查 S3 存储桶名称，然后重试导出。	S3 桶处于错误的 AWS 区域 中。

排查 PostgreSQL 权限错误

将 PostgreSQL 数据库导出到 Amazon S3 时，您可能会看到 PERMISSIONS_DO_NOT_EXIST 错误，指出已跳过某些表。当您在创建数据库实例时指定的超级用户无权访问这些表时，通常会发生此错误。

要修复此错误，请运行以下命令：

```
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA schema_name TO superuser_name
```

有关超级用户权限的更多信息，请参阅 主用户账户权限。

将数据库集群还原到指定时间

您可以将数据库集群还原到特定时间点，从而创建新的数据库集群。

将数据库集群还原到某个时间点时，您可以选择默认的 Virtual Private Cloud (VPC) 安全组，也可以将自定义 VPC 安全组应用于数据库集群。

还原的数据库集群自动与默认数据库集群和数据库参数组关联。但是，您可以通过在还原期间指定自定义参数组来应用它们。

Amazon Aurora 不断将数据库集群的日志记录上传到 Amazon S3。要查看某个数据库集群的最晚可还原时间，请使用 AWS CLI describe-db-clusters 命令，并查看该数据库集群的 `LatestRestorableTime` 字段中返回的值。

您可以还原至备份保留期内的任何时间点。要查看某个数据库集群的最早可还原时间，请使用 AWS CLI describe-db-clusters 命令，并查看该数据库集群的 `EarliestRestorableTime` 字段中返回的值。

还原的数据库集群的备份保留期与源数据库集群的备份保留期相同。

> **Note**
>
> 本主题中的信息适用于 Amazon Aurora。有关还原 Amazon RDS 数据库实例的信息，请参阅将数据库实例还原到指定时间。
> 有关备份和还原 Aurora 数据库集群的更多信息，请参阅备份和还原 Aurora 数据库集群的概述。
> 对于 Aurora MySQL，可以将预置的数据库集群还原为 Aurora Serverless 数据库集群。有关更多信息，请参阅 还原 Aurora Serverless v1 数据库集群。
> 您也可以使用 AWS Backup 来管理 Amazon Aurora 数据库集群的备份。如果您的数据库集群与 AWS Backup 中的备份计划关联，则该备份计划用于时间点恢复。有关信息，请参阅使用 AWS Backup 将数据库集群还原到指定时间。

有关使用 RDS 扩展支持版本还原 Aurora 数据库集群或全局集群的信息，请参阅使用 Amazon RDS 扩展支持还原Aurora 数据集群或全局集群。

通过自动备份、保留的自动备份或使用 AWS Backup 将数据库集群还原到指定时间。

主题

- 将数据库集群还原到某个时间点
- 将数据库集群从保留的自动备份还原到指定时间
- 使用 AWS Backup 将数据库集群还原到指定时间

将数据库集群还原到某个时间点

您可以使用 AWS Management Console、AWS CLI 或 RDS API 将数据库集群还原到某个时间点。

控制台

将数据库集群还原到指定时间

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Automated backups（自动备份）。

   自动备份便会显示在 Current Region（当前区域）选项卡上。



3. 选择要还原的 数据库集群。
4. 对于 Actions (操作)，选择 Restore to point in time (还原到时间点)。

   此时会显示还原到时间点窗口。
5. 选择最近可还原时间以还原到可能的最近时间，或选择自定义来选择时间。

   如果选择 Custom（自定义），请输入要还原集群的指定日期和时间。

> **Note**
>
> 时间以您的本地时区显示，表示为协调世界时 (UTC) 的偏移量。例如，UTC-5 是东部标准时间/中部夏令时。

6. 对于数据库集群标识符，请输入还原后的目标数据库集群的名称。名称必须唯一。

7. 根据需要选择其他选项，例如数据库实例类和数据库集群存储配置。

   有关每项设置的信息，请参阅 Aurora 数据库集群的设置。

8. 选择还原到时间点。

AWS CLI

要将数据库集群还原到指定时间，请使用 AWS CLI 命令 restore-db-cluster-to-point-in-time 创建新的数据库集群。

您可以指定其他设置。有关每项设置的信息，请参阅 Aurora 数据库集群的设置。

此操作支持资源标记。使用 `--tags` 选项时，将忽略源数据库集群标签，并使用提供的标签。否则，将使用源集群中的最新标签。

Example

对于 Linux、macOS 或 Unix：

```
aws rds restore-db-cluster-to-point-in-time \
       --source-db-cluster-identifier mysourcedbcluster \
       --db-cluster-identifier mytargetdbcluster \
       --restore-to-time 2017-10-14T23:45:00.000Z
```

对于 Windows：

```
aws rds restore-db-cluster-to-point-in-time ^
       --source-db-cluster-identifier mysourcedbcluster ^
       --db-cluster-identifier mytargetdbcluster ^
       --restore-to-time 2017-10-14T23:45:00.000Z
```

Important

如果您使用控制台将数据库集群还原到指定时间，则 Amazon RDS 会为您的数据库集群自动创建主实例（写入器）。如果您使用 AWS CLI 将数据库集群还原到指定时间，则必须明确为数据库集群创建主实例。主实例是在数据库集群中创建的第一个实例。
要为数据库集群创建主实例，请调用 create-db-instance AWS CLI 命令。包括数据库集群名称以作为 `--db-cluster-identifier` 选项值。

RDS API

要将数据库集群还原到指定时间，请结合以下参数调用 Amazon RDS API `RestoreDBClusterToPointInTime` 操作：

- `SourceDBClusterIdentifier`
- `DBClusterIdentifier`
- `RestoreToTime`

> **Important**
>
> 如果您使用控制台将数据库集群还原到指定时间，则 Amazon RDS 会为您的数据库集群自动创建主实例（写入器）。如果使用 RDS API 将数据库集群还原到指定时间，请确保为数据库集群明确创建主实例。主实例是在数据库集群中创建的第一个实例。
> 要为数据库集群创建主实例，请调用 RDS API 操作 CreateDBInstance。包括数据库集群的名称作为 `DBClusterIdentifier` 参数值。

将数据库集群从保留的自动备份还原到指定时间

如果备份在源集群的保留期内，则可以在删除源数据库集群后，从保留的自动备份中还原数据库集群。该过程类似于从自动备份中还原数据库集群。

> **Note**
>
> 您无法使用此过程还原 Aurora Serverless v1 数据库集群，因为不会保留 Aurora Serverless v1 集群的自动备份。

控制台

将数据库集群还原到指定时间

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Automated backups（自动备份）。
3. 选择已保留选项卡。



4. 选择要还原的 数据库集群。
5. 对于 Actions (操作)，选择 Restore to point in time (还原到时间点)。

   此时会显示还原到时间点窗口。
6. 选择最近可还原时间以还原到可能的最近时间，或选择自定义来选择时间。

   如果选择 Custom（ 自定义 ），请输入要还原集群的指定日期和时间。

   > **Note**
   >
   > 时间以您的本地时区显示，表示为协调世界时 (UTC) 的偏移量。例如，UTC-5 是东部标准时间/中部夏令时。
7. 对于数据库集群标识符，请输入还原后的目标数据库集群的名称。名称必须唯一。
8. 根据需要选择其他选项，例如数据库实例类。

   有关每项设置的信息，请参阅 Aurora 数据库集群的设置。
9. 选择还原到时间点。

AWS CLI

要将数据库集群还原到指定时间，请使用 AWS CLI 命令 restore-db-cluster-to-point-in-time 创建新的数据库集群。

您可以指定其他设置。有关每项设置的信息，请参阅 Aurora 数据库集群的设置。

此操作支持资源标记。使用 `--tags` 选项时，将忽略源数据库集群标签，并使用提供的标签。否则，将使用源集群中的最新标签。

Example

对于 Linux、macOS 或 Unix：

```
aws rds restore-db-cluster-to-point-in-time \
    --source-db-cluster-resource-id cluster-123ABCEXAMPLE \
    --db-cluster-identifier mytargetdbcluster \
    --restore-to-time 2017-10-14T23:45:00.000Z
```

对于 Windows：

```
aws rds restore-db-cluster-to-point-in-time ^
    --source-db-cluster-resource-id cluster-123ABCEXAMPLE ^
    --db-cluster-identifier mytargetdbcluster ^
    --restore-to-time 2017-10-14T23:45:00.000Z
```

Important

如果您使用控制台将数据库集群还原到指定时间，则 Amazon RDS 会为您的数据库集群自动创建主实例（写入器）。如果您使用 AWS CLI 将数据库集群还原到指定时间，则必须明确为数据库集群创建主实例。主实例是在数据库集群中创建的第一个实例。
要为数据库集群创建主实例，请调用 create-db-instance AWS CLI 命令。包括数据库集群名称以作为 `--db-cluster-identifier` 选项值。

RDS API

要将数据库集群还原到指定时间，请结合以下参数调用 Amazon RDS API `RestoreDBClusterToPointInTime` 操作：

- `SourceDbClusterResourceId`
- `DBClusterIdentifier`
- `RestoreToTime`

Important

如果您使用控制台将数据库集群还原到指定时间，则 Amazon RDS 会为您的数据库集群自动创建主实例（写入器）。如果使用 RDS API 将数据库集群还原到指定时间，请确保为数据库集群明确创建主实例。主实例是在数据库集群中创建的第一个实例。
要为数据库集群创建主实例，请调用 RDS API 操作 CreateDBInstance。包括数据库群集的名称作为 `DBClusterIdentifier` 参数值。

使用 AWS Backup 将数据库集群还原到指定时间

您可以使用 AWS Backup 管理您的自动备份，然后将其还原到指定时间。为此，您需要在 AWS Backup 中创建备份计划，并将您的数据库集群分配为资源。然后，您可以在备份规则中为 PITR 启用连续备份。有关备份计划和备份规则的更多信息，请参阅 AWS Backup 开发人员指南。

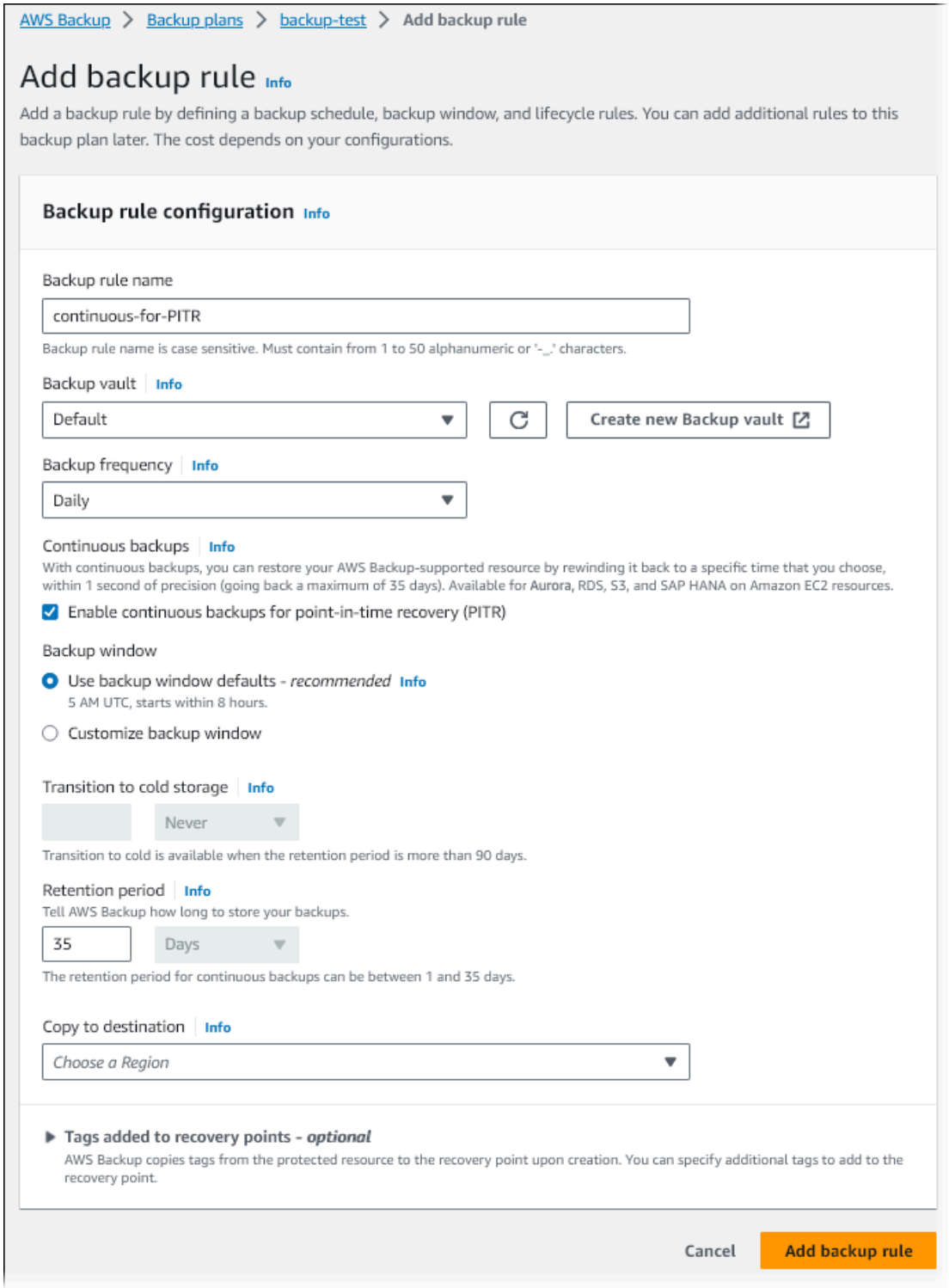
在 AWS Backup 中启用连续备份

您可以在备份规则中启用连续备份。

为 PITR 启用连续备份

1. 登录到 AWS Management Console，然后通过以下网址打开 AWS Backup 控制台：https://console.aws.amazon.com/backup。
2. 在导航窗格中，选择备份计划。
3. 在备份计划名称下，选择用于备份数据库集群的备份计划。
4. 在备份规则部分下，选择添加备份规则。

   将显示添加备份规则页面。
5. 选中启用连续备份以实现时间点故障恢复（PITR）复选框。



6. 根据需要选择其他设置，然后选择添加备份规则。

从 AWS Backup 中的连续备份还原

您从备份保管库还原到指定时间。

控制台

可以使用 AWS Management Console 将数据库集群还原到指定时间。

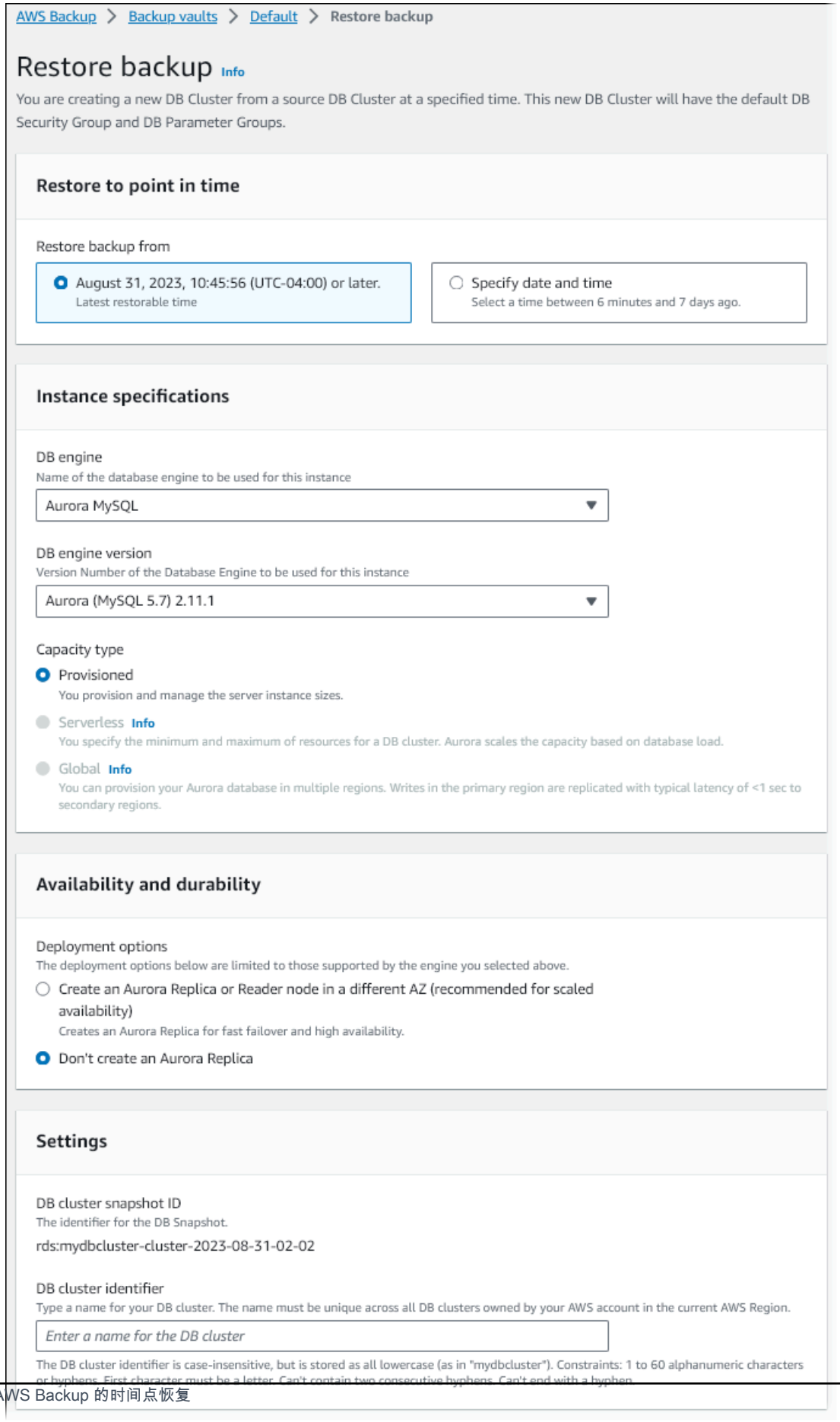
从 AWS Backup 中的连续备份还原

1. 登录到 AWS Management Console，然后通过以下网址打开 AWS Backup 控制台：https://console.aws.amazon.com/backup。
2. 在导航窗格中，选择备份保管库。
3. 例如，选择包含连续备份的备份保管库，例如默认。

   将显示备份保管库详细信息页面。
4. 在恢复点下，选择自动备份的恢复点。

   它的备份类型为连续，且名称带有 `continuous:cluster-`*`AWS-Backup-job-number`*。
5. 对于操作，选择还原。

   将显示还原备份页面。


AWS Backup > Backup vaults > Default > Restore backup
Restore backup Info
You are creating a new DB Cluster from a source DB Cluster at a specified time. This new DB Cluster will have the default DB Security Group and DB Parameter Groups.
Restore to point in time
Restore backup from
August 31, 2023, 10:45:56 (UTC-04:00) or later.
Latest restorable time
Specify date and time
Select a time between 6 minutes and 7 days ago.
Instance specifications
DB engine
Name of the database engine to be used for this instance
Aurora MySQL
DB engine version
Version Number of the Database Engine to be used for this instance
Aurora (MySQL 5.7) 2.11.1
Capacity type
Provisioned
You provision and manage the server instance sizes.
Serverless Info
You specify the minimum and maximum of resources for a DB cluster. Aurora scales the capacity based on database load.
Global Info
You can provision your Aurora database in multiple regions. Writes in the primary region are replicated with typical latency of <1 sec to secondary regions.
Availability and durability
Deployment options
The deployment options below are limited to those supported by the engine you selected above.
Create an Aurora Replica or Reader node in a different AZ (recommended for scaled availability)
Creates an Aurora Replica for fast failover and high availability.
Don't create an Aurora Replica
Settings
DB cluster snapshot ID
The identifier for the DB Snapshot.
rds:mydbcluster-cluster-2023-08-31-02-02
DB cluster identifier
Type a name for your DB cluster. The name must be unique across all DB clusters owned by your AWS account in the current AWS Region.
Enter a name for the DB cluster
The DB cluster identifier is case-insensitive, but is stored as all lowercase (as in "mydbcluster"). Constraints: 1 to 60 alphanumeric characters or hyphens. First character must be a letter. Can't contain two consecutive hyphens. Can't end with a hyphen.

6. 对于还原到时间点，请选择指定日期和时间以还原到特定的时间点。
7. 根据需要选择其他设置以还原数据库集群，然后选择还原备份。

   将显示任务页面，其中显示还原任务窗格。页面顶部的消息提供了有关还原作业的信息。

还原数据库集群后，必须向其添加主（写入器）数据库实例。要为数据库集群创建主实例，请调用 create-db-instance AWS CLI 命令。包括数据库群集的名称作为 `--db-cluster-identifier` 参数值。

CLI

请使用 start-restore-job AWS CLI 命令将数据库集群还原到指定时间。以下参数为必需参数：

- `--recovery-point-arn` – 要从中还原的恢复点的 Amazon 资源名称（ARN）。
- `--resource-type` – 使用 `Aurora`。
- `--iam-role-arn` – 您用于 AWS Backup 操作的 IAM 角色的 ARN。
- `--metadata` – 用于还原数据库集群的元数据。以下参数为必需参数：
  - `DBClusterIdentifier`
  - `Engine`
  - `RestoreToTime` 或 `UseLatestRestorableTime`

以下示例说明如何将数据库集群还原到指定时间。

```
aws backup start-restore-job \
--recovery-point-arn arn:aws:backup:eu-central-1:123456789012:recovery-
point:continuous:cluster-itsreallyjustanexample1234567890-487278c2 \
--resource-type Aurora \
--iam-role-arn arn:aws:iam::123456789012:role/service-role/AWSBackupDefaultServiceRole
 \
--metadata '{"DBClusterIdentifier":"backup-pitr-test","Engine":"aurora-
mysql","RestoreToTime":"2023-09-01T17:00:00.000Z"}'
```

以下示例说明如何将数据库集群还原到最新的可还原时间。

```
aws backup start-restore-job \
--recovery-point-arn arn:aws:backup:eu-central-1:123456789012:recovery-
point:continuous:cluster-itsreallyjustanexample1234567890-487278c2 \
--resource-type Aurora \
```

```
--iam-role-arn arn:aws:iam::123456789012:role/service-role/AWSBackupDefaultServiceRole
 \
--metadata '{"DBClusterIdentifier":"backup-pitr-latest","Engine":"aurora-
mysql","UseLatestRestorableTime":"true"}'
```

还原数据库集群后，必须向其添加主（写入器）数据库实例。要为数据库集群创建主实例，请调用 create-db-instance AWS CLI 命令。包括数据库群集的名称作为 `--db-cluster-identifier` 参数值。

删除数据库集群快照

如果不再需要，您可以删除 Amazon RDS 管理的数据库集群快照。

> **Note**
>
> 要删除 AWS Backup 管理的备份，请使用 AWS Backup 控制台。有关 AWS Backup 的更多信息，请参阅 AWS Backup 开发人员指南。

删除数据库集群快照

您可以使用控制台、AWS CLI 或 RDS API 删除数据库集群快照。

要删除共享或公共快照，您必须登录到拥有快照的 AWS 账户。

控制台

删除数据库集群快照

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择快照。
3. 选择要删除的数据库集群快照。
4. 对于 Actions（操作），选择 Delete snapshot（删除快照）。
5. 在确认页面上选择 Delete (删除)。

AWS CLI

您可以使用 AWS CLI 命令 delete-db-cluster-snapshot 删除数据库集群快照。

以下选项用于删除数据库集群快照。

- `--db-cluster-snapshot-identifier` – 数据库集群快照的标识符。

Example

以下代码删除 `mydbclustersnapshot` 数据库集群快照。

对于 Linux、macOS 或 Unix：

```
aws rds delete-db-cluster-snapshot \
    --db-cluster-snapshot-identifier mydbclustersnapshot
```

对于 Windows：

```
aws rds delete-db-cluster-snapshot ^
    --db-cluster-snapshot-identifier mydbclustersnapshot
```

RDS API

您可以使用 Amazon RDS API 操作 DeleteDBClusterSnapshot 删除数据库集群快照。

以下参数用于删除数据库集群快照。

- `DBClusterSnapshotIdentifier` – 数据库集群快照的标识符。

教程：从数据库集群快照中还原 Amazon Aurora 数据库集群

使用 Amazon Aurora 的一个常见情况就是您有一个偶尔使用而不是一直使用的数据库实例。例如，您可以使用数据库集群来保存仅每季度运行一次的报告的数据。在这种情况下节省资金的一种方法是在完成报告后创建数据库集群的数据库集群快照。然后，您删除数据库集群，并在需要上载新数据以及在下一季度运行报告时将其还原。

在还原数据库集群时，您需要提供用来还原的数据库集群快照的名称。然后，为通过还原操作创建的新数据库集群提供名称。有关从快照还原数据库集群的更多详细信息，请参阅从数据库集群快照还原。

在本教程中，我们还将还原的数据库集群从 Aurora MySQL 版本 2（与 MySQL 5.7 兼容）升级到 Aurora MySQL 版本 3（与 MySQL 8.0 兼容）。

使用 Amazon RDS 控制台或 AWS CLI，通过数据库集群快照将数据库集群还原到指定时间。

有关 Amazon RDS 的 AWS KMS 密钥管理的信息，请参阅 AWS KMS key 管理。

主题

- 教程：使用 Amazon RDS 控制台从数据库集群快照中还原数据库集群
- 教程：使用 AWS CLI 从数据库集群快照中还原数据库集群

教程：使用 Amazon RDS 控制台从数据库集群快照中还原数据库集群

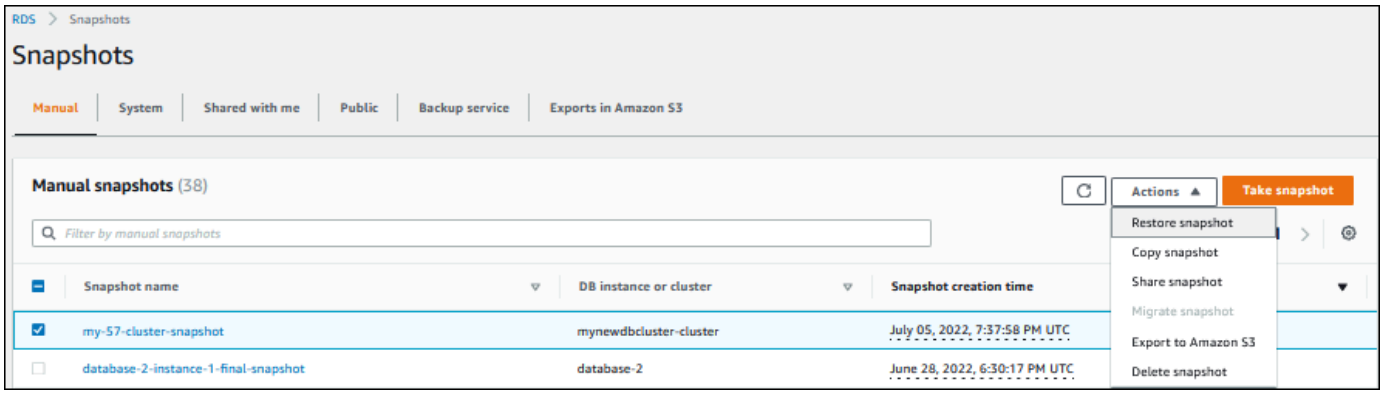
在本教程中，您使用 Amazon RDS 控制台从数据库集群快照中还原数据库集群。使用 AWS Management Console从快照还原数据库集群时，还将创建主（写入器）数据库实例。

> **Note**
>
> 在创建主数据库实例时，它显示为读取器实例，但创建后，它是写入器实例。

从数据库集群快照还原数据库集群

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择快照。
3. 选择要从其还原的数据库集群快照。

4. 对于操作，选择还原快照。

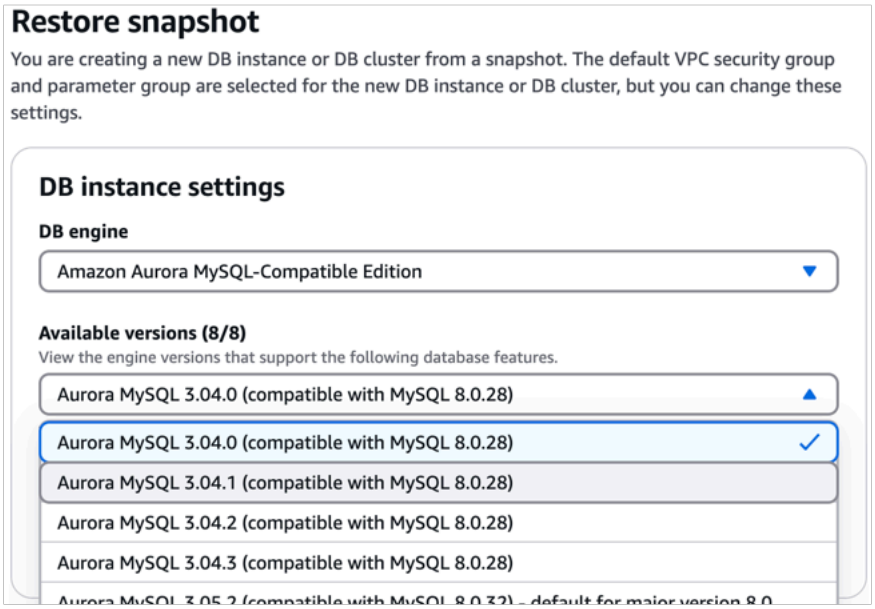


此时会显示还原快照页面。

5. 在 DB instance settings（数据库实例设置）下，执行下列操作：

   a. 对 DB engine（数据库引擎）使用默认设置。

   b. 对于可用版本，选择与 MySQL 8.0 兼容的版本，例如 Aurora MySQL 3.04.0（与 MySQL 8.0.28 兼容）。



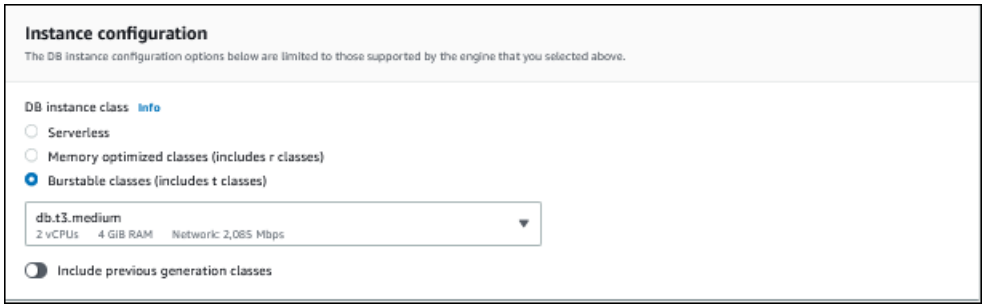
6. 在 Settings（设置）项下的 DB instance identifier（数据库实例标识符）中，输入要用于还原的数据库实例的唯一名称，如 **my-80**。

> **Note**
>
> 为创建数据库集群标识符，Amazon RDS 将 `-cluster` 附加到您指定的数据库实例标识符之后。

7. 在 Connectivity（连接）项下，使用以下各项的默认设置：
   - Virtual Private Cloud (VPC) [虚拟私有云（VPC）]
   - DB subnet group（数据库子网组）
   - 公有访问权限
   - VPC security group (firewall) [VPC 安全组（防火墙）]
8. 选择 DB instance class（数据库实例类）。

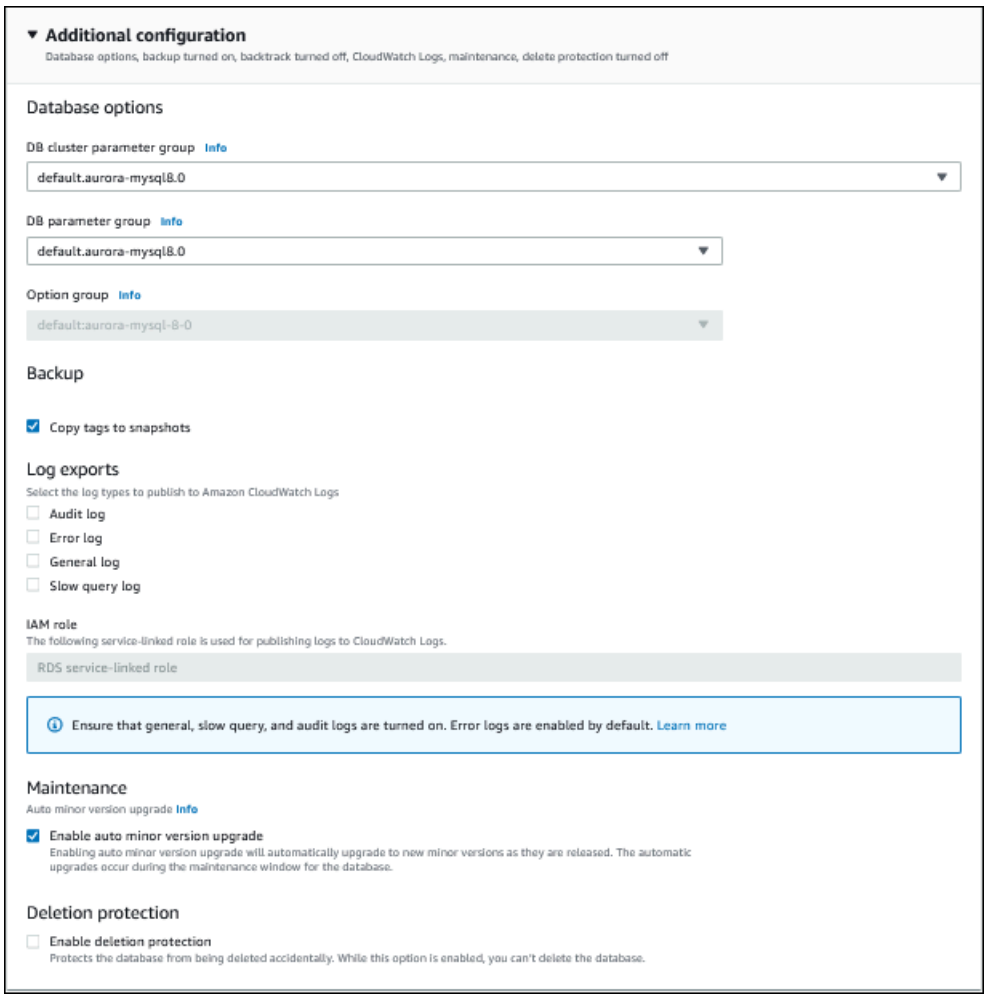
   在本教程中，请选择 Burstable classes (includes t classes) [突增型类（包括 t 类）]，然后选择 db.t3.medium。

> **Note**
>
> 建议仅将 T 数据库实例类用于开发和测试服务器，或其他非生产服务器。有关 T 实例类的更多详细信息，请参阅数据库实例类类型。



9. 对于 Database authentication（数据库身份验证），请使用默认设置。
10. 使用 Encryption（加密）项的默认设置。

    如果快照的源数据库集群已加密，则还原的数据库集群也会加密。您无法将其设置为未加密。
11. 展开页面底部的 Additional configuration（其他配置）。



12. 进行以下选择：

    a. 对于本教程，为 DB cluster parameter group（数据库集群参数组）使用默认值。

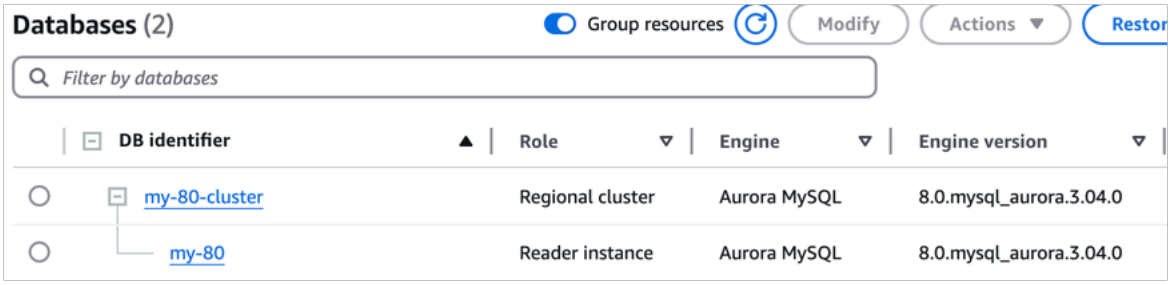
    b. 对于本教程，为 DB parameter group（数据库参数组）使用默认值。

    c. 对于 Log exports（日志导出），选中所有复选框。

    d. 对于 Deletion protection（删除保护），选中 Enable deletion protection（启用删除保护）复选框。

13. 选择还原数据库实例。

Databases（数据库）页面会显示状态为 `Creating` 的已还原数据库集群。



在创建主数据库实例时，它显示为读取器实例，但创建后，它是写入器实例。

教程：使用 AWS CLI 从数据库集群快照中还原数据库集群

在本教程中，您使用 AWS CLI 从数据库集群快照中还原数据库集群。使用 AWS CLI 从快照还原恢复数据库集群有两个步骤：

1. 还原数据库集群 使用 restore-db-cluster-from-snapshot 命令
2. 创建主（写入器）数据库实例 使用 create-db-instance 命令

还原数据库集群

您使用 `restore-db-cluster-from-snapshot` 命令。以下选项为必填：

- `--db-cluster-identifier` – 已还原的数据库集群的名称。
- `--snapshot-identifier` – 要从中进行还原的数据库快照的名称。
- `--engine` – 已还原的数据库集群的数据库引擎。它必须与源数据库集群的数据库引擎兼容。

  选项如下：

  - `aurora-mysql` – 与 Aurora MySQL 5.7 和 8.0 兼容。
  - `aurora-postgresql` – 与 Aurora PostgreSQL 兼容。

  在此示例中，我们使用的是 `aurora-mysql`。
- `--engine-version` – 已还原的数据库集群的版本。在此示例中，我们使用与 MySQL 8.0 兼容的版本。

以下示例从名为 `my-57-cluster-snapshot` 的数据库集群快照中还原与 Aurora MySQL 8.0 兼容的名为 `my-new-80-cluster` 的数据库集群。

还原数据库集群

- 使用以下命令之一。

  对于 Linux、macOS 或 Unix：

```
aws rds restore-db-cluster-from-snapshot \
    --db-cluster-identifier my-new-80-cluster \
    --snapshot-identifier my-57-cluster-snapshot \
    --engine aurora-mysql \
    --engine-version 8.0.mysql_aurora.3.02.0
```

  对于 Windows：

```
aws rds restore-db-cluster-from-snapshot ^
    --db-cluster-identifier my-new-80-cluster ^
    --snapshot-identifier my-57-cluster-snapshot ^
    --engine aurora-mysql ^
    --engine-version 8.0.mysql_aurora.3.02.0
```

输出与以下内容类似。

```
{
    "DBCluster": {
        "AllocatedStorage": 1,
        "AvailabilityZones": [
            "eu-central-1b",
            "eu-central-1c",
            "eu-central-1a"
        ],
        "BackupRetentionPeriod": 14,
        "DatabaseName": "",
        "DBClusterIdentifier": "my-new-80-cluster",
        "DBClusterParameterGroup": "default.aurora-mysql8.0",
        "DBSubnetGroup": "default",
        "Status": "creating",
        "Endpoint": "my-new-80-cluster.cluster-############.eu-
central-1.rds.amazonaws.com",
        "ReaderEndpoint": "my-new-80-cluster.cluster-ro-############.eu-
central-1.rds.amazonaws.com",
        "MultiAZ": false,
```

```
        "Engine": "aurora-mysql",
        "EngineVersion": "8.0.mysql_aurora.3.02.0",
        "Port": 3306,
        "MasterUsername": "admin",
        "PreferredBackupWindow": "01:55-02:25",
        "PreferredMaintenanceWindow": "thu:21:14-thu:21:44",
        "ReadReplicaIdentifiers": [],
        "DBClusterMembers": [],
        "VpcSecurityGroups": [
            {
                "VpcSecurityGroupId": "sg-########",
                "Status": "active"
            }
        ],
        "HostedZoneId": "Z1RLNU0EXAMPLE",
        "StorageEncrypted": true,
        "KmsKeyId": "arn:aws:kms:eu-central-1:123456789012:key/#######-5ccc-49cc-8aaa-
############",
        "DbClusterResourceId": "cluster-ZZ12345678ITSJUSTANEXAMPLE",
        "DBClusterArn": "arn:aws:rds:eu-central-1:123456789012:cluster:my-new-80-
cluster",
        "AssociatedRoles": [],
        "IAMDatabaseAuthenticationEnabled": false,
        "ClusterCreateTime": "2022-07-05T20:45:42.171000+00:00",
        "EngineMode": "provisioned",
        "DeletionProtection": false,
        "HttpEndpointEnabled": false,
        "CopyTagsToSnapshot": false,
        "CrossAccountClone": false,
        "DomainMemberships": [],
        "TagList": []
    }
}
```

创建主（写入器）数据库实例

要创建主（写入器）数据库实例，请使用 `create-db-instance` 命令。以下选项为必填：

- `--db-cluster-identifier` – 已还原的数据库集群的名称。
- `--db-instance-identifier` – 主数据库实例的名称。
- `--db-instance-class` – 主数据库实例类的实例名称。在此示例中，我们使用的是 `db.t3.medium`。

> **Note**
>
> 建议仅将 T 数据库实例类用于开发和测试服务器，或其他非生产服务器。有关 T 实例类的更多详细信息，请参阅数据库实例类类型。

- `--engine` –主数据库实例的数据库引擎。它必须与已还原的数据库集群使用的数据库引擎相同。

  选项如下：

  - `aurora-mysql` – 与 Aurora MySQL 5.7 和 8.0 兼容。
  - `aurora-postgresql` – 与 Aurora PostgreSQL 兼容。

  在此示例中，我们使用的是 `aurora-mysql`。

以下示例在已还原的与 Aurora MySQL 8.0 兼容且名为 `my-new-80-cluster` 的数据库集群中创建一个名为 `my-new-80-cluster-instance` 的主（写入器）数据库实例。

创建主数据库实例

- 使用以下命令之一。

  对于 Linux、macOS 或 Unix：

```
aws rds create-db-instance \
    --db-cluster-identifier my-new-80-cluster \
    --db-instance-identifier my-new-80-cluster-instance \
    --db-instance-class db.t3.medium \
    --engine aurora-mysql
```

  对于 Windows：

```
aws rds create-db-instance ^
    --db-cluster-identifier my-new-80-cluster ^
    --db-instance-identifier my-new-80-cluster-instance ^
    --db-instance-class db.t3.medium ^
    --engine aurora-mysql
```

输出与以下内容类似。

```
{
    "DBInstance": {
        "DBInstanceIdentifier": "my-new-80-cluster-instance",
        "DBInstanceClass": "db.t3.medium",
        "Engine": "aurora-mysql",
        "DBInstanceStatus": "creating",
        "MasterUsername": "admin",
        "AllocatedStorage": 1,
        "PreferredBackupWindow": "01:55-02:25",
        "BackupRetentionPeriod": 14,
        "DBSecurityGroups": [],
        "VpcSecurityGroups": [
            {
                "VpcSecurityGroupId": "sg-########",
                "Status": "active"
            }
        ],
        "DBParameterGroups": [
            {
                "DBParameterGroupName": "default.aurora-mysql8.0",
                "ParameterApplyStatus": "in-sync"
            }
        ],
        "DBSubnetGroup": {
            "DBSubnetGroupName": "default",
            "DBSubnetGroupDescription": "default",
            "VpcId": "vpc-2305ca49",
            "SubnetGroupStatus": "Complete",
            "Subnets": [
                {
                    "SubnetIdentifier": "subnet-########",
                    "SubnetAvailabilityZone": {
                        "Name": "eu-central-1a"
                    },
                    "SubnetOutpost": {},
                    "SubnetStatus": "Active"
                },
                {
                    "SubnetIdentifier": "subnet-########",
                    "SubnetAvailabilityZone": {
                        "Name": "eu-central-1b"
                    },
                    "SubnetOutpost": {},
```

```
                    "SubnetStatus": "Active"
                },
                {
                    "SubnetIdentifier": "subnet-########",
                    "SubnetAvailabilityZone": {
                        "Name": "eu-central-1c"
                    },
                    "SubnetOutpost": {},
                    "SubnetStatus": "Active"
                }
            ]
        },
        "PreferredMaintenanceWindow": "sat:02:41-sat:03:11",
        "PendingModifiedValues": {},
        "MultiAZ": false,
        "EngineVersion": "8.0.mysql_aurora.3.02.0",
        "AutoMinorVersionUpgrade": true,
        "ReadReplicaDBInstanceIdentifiers": [],
        "LicenseModel": "general-public-license",
        "OptionGroupMemberships": [
            {
                "OptionGroupName": "default:aurora-mysql-8-0",
                "Status": "in-sync"
            }
        ],
        "PubliclyAccessible": false,
        "StorageType": "aurora",
        "DbInstancePort": 0,
        "DBClusterIdentifier": "my-new-80-cluster",
        "StorageEncrypted": true,
        "KmsKeyId": "arn:aws:kms:eu-central-1:534026745191:key/#######-5ccc-49cc-8aaa-
############",
        "DbiResourceId": "db-5C6UT5PU0YETANOTHEREXAMPLE",
        "CACertificateIdentifier": "rds-ca-2019",
        "DomainMemberships": [],
        "CopyTagsToSnapshot": false,
        "MonitoringInterval": 0,
        "PromotionTier": 1,
        "DBInstanceArn": "arn:aws:rds:eu-central-1:123456789012:db:my-new-80-cluster-
instance",
        "IAMDatabaseAuthenticationEnabled": false,
        "PerformanceInsightsEnabled": false,
        "DeletionProtection": false,
        "AssociatedRoles": [],
```

```
        "TagList": []
    }
}
```

监控 Amazon Aurora 集群中的指标

Amazon Aurora 使用重复的数据库服务器集群。监控 Aurora 集群通常需要检查多个数据库实例的运行状况。实例可能具有专门的作用，主要处理写入操作和/或读取操作。您还可以通过衡量复制滞后来监控集群的整体运行状况。这是指一个数据库实例所做的更改可供其他实例使用之前的时间。

主题

- 监控计划
- 性能基准
- 性能准则
- Amazon Aurora 的监控工具
- 查看集群状态
- 来自 Amazon Aurora 的建议
- 在 Amazon RDS 控制台中查看指标
- 使用性能详情控制面板查看组合指标
- 使用 Amazon CloudWatch 监控 Amazon Aurora 指标
- 使用 CloudWatch 数据库洞察监控 Amazon Aurora 数据库
- 在 Amazon Aurora 上使用性能详情监控数据库负载
- 使用 Amazon DevOps Guru for Amazon RDS 分析 Aurora 性能异常
- 使用增强监控来监控操作系统指标
- Amazon Aurora 的指标参考

监控计划

在开始监控 Amazon Aurora 之前，请创建监控计划。该计划应回答以下问题：

- 您的监控目标是什么？
- 您将监控哪些资源？
- 监控这些资源的频率如何？
- 您将使用哪些监控工具？
- 谁负责执行监控任务？
- 出现问题时应通知谁？

性能基准

为了实现您的监控目标，您需要建立一个基准。为此，请在 Amazon Aurora 环境的不同时间、不同负载条件下测量性能。您可以监控如下指标：

- 网络吞吐量
- 客户端连接
- 用于读取、写入或元数据操作的 I/O
- 数据库实例的突增信用余额

我们建议您存储 Amazon Aurora 的历史性能数据。使用存储的数据，您可以将当前性能与过去的趋势进行比较。您还可以区分正常性能模式与异常情况，并设计解决问题的方法。

性能准则

通常，性能指标的可接受值取决于应用程序相对于基准正在执行的操作。应调查相对于基准性能的一致或趋势性变化。以下指标通常是性能问题的根源：

- 高 CPU 或 RAM 消耗 – CPU 或 RAM 消耗值高可能是正常情况，前提是它们符合您的应用程序目标（如吞吐量或并发度）并且符合预期。
- 磁盘空间消耗 – 如果使用的空间始终不低于总磁盘空间的 85%，则应调查磁盘空间消耗。应查看是否可以从实例中删除数据或是将数据存档到其他系统以释放空间。
- 网络流量 – 对于网络流量，应与系统管理员进行讨论，以了解域网络和 Internet 连接的预期吞吐量。如果吞吐量始终低于预期，则应调查网络流量。
- 数据库连接 – 如果发现用户连接数较高，实例性能下降且响应时间延长，请考虑约束数据库连接。数据库实例的最佳用户连接数因您的实例类所执行操作的复杂性而异。要确定数据库连接的数量，请将数据库实例与参数组关联，其中 `User Connections` 参数设置为 0 以外的值（无限制）。您可以使用现有参数组或新建一个。有关更多信息，请参阅“Amazon Aurora 的参数组”。
- IOPS 指标 – IOPS 指标的预期值取决于磁盘规格和服务器配置，因此，请使用您的基准来了解典型状况。调查值是否始终与您的基准不同。为获得最佳 IOPS 性能，请确保典型工作集与内存大小相适，以最大限度地减少读取和写入操作。

当性能超出已建立的基准时，您可能需要进行更改以优化数据库可用性，进而适应工作负载。例如，您可能需要更改数据库实例的实例类。或者，您可能需要更改可供客户端使用的数据库实例和只读副本的数量。

Amazon Aurora 的监控工具

监控是保持 Amazon Aurora 和您的其他 AWS 解决方案的可靠性、可用性和性能的重要方面。AWS 提供各种监控工具来监控 Amazon Aurora、在出现错误时进行报告并适时自动采取措施。

主题

- 自动监控工具
- 手动监控工具

自动监控工具

建议您尽可能实现监控任务自动化。

主题

- Amazon Aurora 集群状态和建议
- Amazon Aurora 的 Amazon CloudWatch 指标
- Amazon RDS 性能详情和操作系统监控
- 集成服务

Amazon Aurora 集群状态和建议

您可以使用以下自动化工具来监控 Amazon Aurora，并在出现错误时进行报告：

- Amazon Aurora 集群状态 – 通过使用 Amazon RDS 控制台、AWS CLI 或 RDS API，查看有关您的集群当前状态的详细信息。
- Amazon Aurora 建议 — 回应自动提供数据库资源（例如数据库实例、数据库集群、和数据库集群参数组）的建议。有关更多信息，请参阅 来自 Amazon Aurora 的建议。

Amazon Aurora 的 Amazon CloudWatch 指标

Amazon Aurora 与 Amazon CloudWatch 集成以提供其他监控功能。

- Amazon CloudWatch – 此服务可实时监控您的AWS资源以及您在AWS上运行的应用程序。可以将以下 Amazon CloudWatch 功能用于 Amazon Aurora：

- Amazon CloudWatch 指标 – Amazon Aurora 每分钟自动向 CloudWatch 发送一次每个活动数据库的指标。对于 CloudWatch 中的 Amazon RDS 指标，您不会获得额外费用。有关更多信息，请参阅Amazon Aurora 的 Amazon CloudWatch 指标。
- Amazon CloudWatch 警报 – 您可以在特定时间段内监控单个 Amazon Aurora 指标。然后，您可以根据相对于您设置的阈值的指标值来执行一个或多个操作。

Amazon RDS 性能详情和操作系统监控

您可以使用以下自动化工具来监控 Amazon Aurora 性能：

- Amazon RDS Performance Insights - 可评估数据库上的负载，并确定执行操作的时间和位置。有关更多信息，请参阅 在 Amazon Aurora 上使用性能详情监控数据库负载。
- Amazon RDS 增强监控 - 实时查看操作系统的指标。有关更多信息，请参阅 使用增强监控来监控操作系统指标。

集成服务

以下 AWS 服务与 Amazon Aurora 集成：

- Amazon EventBridge 是一种无服务器事件总线服务，可以轻松地将应用程序与来自各种来源的数据相连接。有关更多信息，请参阅 监控 Amazon Aurora 事件。
- Amazon CloudWatch Logs 可让您监控、存储和访问来自 Amazon Aurora 实例、CloudTrail 和其他来源的日志文件。有关更多信息，请参阅 监控 Amazon Aurora 日志文件。
- AWS CloudTrail 捕获由您的 AWS 账户 或代表该账户发出的 API 调用和相关事件，并将日志文件传输到您指定的 Amazon S3 桶。有关更多信息，请参阅 监控 AWS CloudTrail 中的 Amazon Aurora API 调用。
- 数据库活动流是一项 Amazon Aurora 功能，它提供 数据库集群中近乎实时的活动流。有关更多信息，请参阅 使用数据库活动流监控 Amazon Aurora。
- DevOps Guru for RDS 是一项 Amazon DevOps Guru 功能，它将机器学习应用于 Amazon Aurora 数据库的 Performance Insights 指标。有关更多信息，请参阅 使用 Amazon DevOps Guru for Amazon RDS 分析 Aurora 性能异常。

手动监控工具

您需要手动监控那些 CloudWatch 警报未覆盖的项目。Amazon RDS、CloudWatch、AWS Trusted Advisor 和其他AWS控制台控制面板提供您的AWS环境状态的概览视图。建议您还要查看数据库实例上的日志文件。

- 您可以从 Amazon RDS 控制台监控资源的以下项目：
  - 与数据库实例的连接数
  - 针对数据库实例的读写操作数量
  - 数据库实例当前使用的存储量
  - 为数据库实例使用的内存和 CPU 量
  - 流入和流出数据库实例的网络流量
- 您可以从 Trusted Advisor 控制面板查看以下成本优化、安全性、容错能力和性能改进检查：
  - Amazon RDS 闲置数据库实例
  - Amazon RDS 安全组访问风险
  - Amazon RDS 备份
  - Amazon RDS 多可用区
  - Aurora 数据库实例可访问性

  有关这些检查的更多信息，请参阅 Trusted Advisor 最佳实践（检查）。
- CloudWatch 主页显示：
  - 当前告警和状态
  - 告警和资源图表
  - 服务运行状况

  此外，您还可以使用 CloudWatch 执行以下操作：
  - 创建自定义控制面板以监控您关注的服务。
  - 绘制指标数据图，以排除问题并弄清楚趋势。
  - 搜索并浏览您所有的 AWS 资源指标。
  - 创建和编辑警报以接收有关问题的通知。

查看集群状态

使用 Amazon RDS 控制台，您可以快速访问数据库集群的状态。

主题

- 查看 Amazon Aurora 数据库集群
- 查看数据库集群状态
- 查看 Aurora 集群中的数据库实例状态

查看 Amazon Aurora 数据库集群

您有若干选项可用于查看有关您的 Amazon Aurora 数据库集群以及数据库集群中的数据库实例的信息。

- 您可以通过从导航窗格选择 Databases (数据库) 在 Amazon RDS 控制台中查看数据库集群和数据库实例。
- 您可以使用 AWS Command Line Interface (AWS CLI) 获取数据库集群和数据库实例信息。
- 您可以使用 Amazon RDS API 获取数据库集群和数据库实例信息。

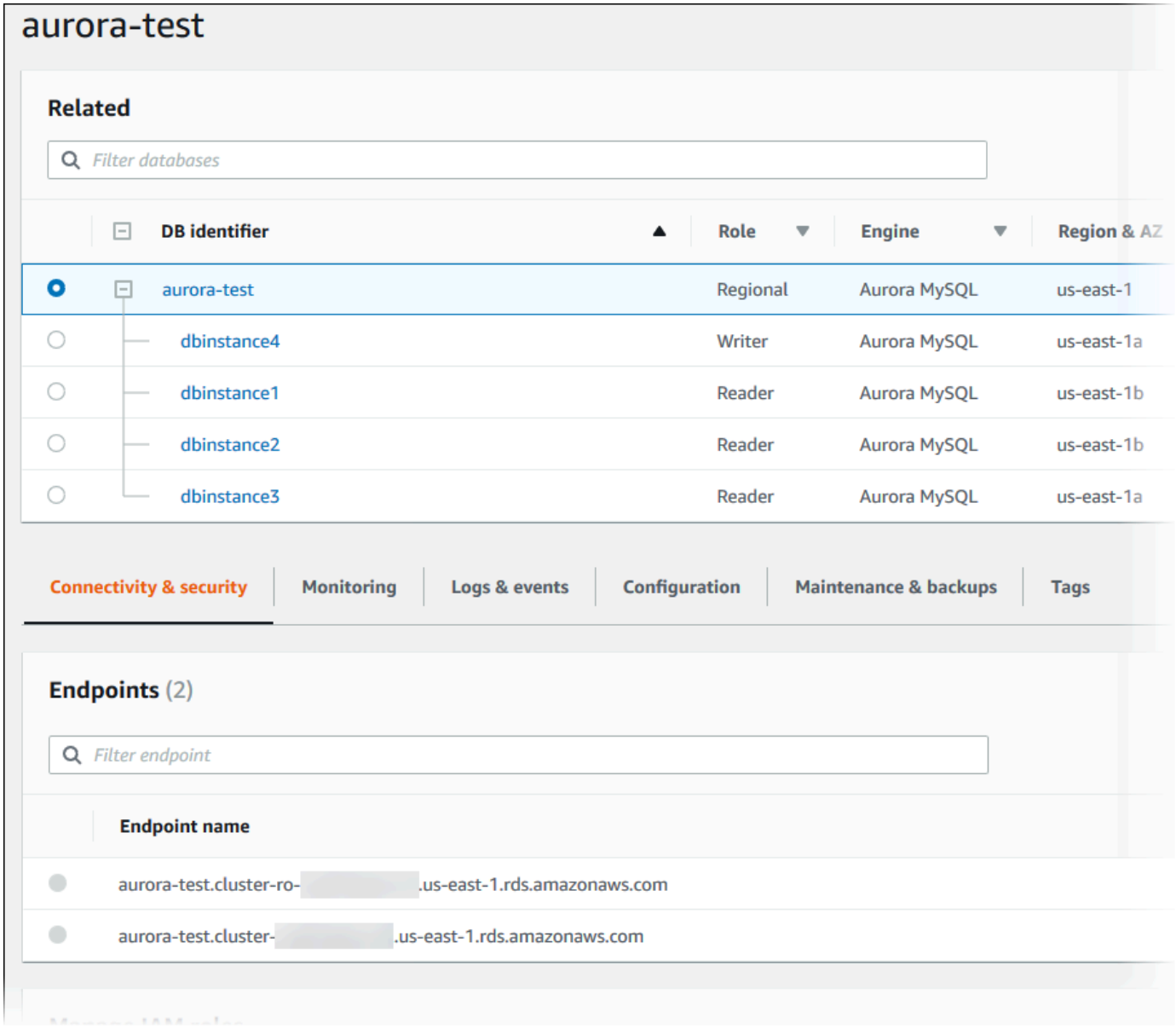
控制台

在 Amazon RDS 控制台中，您可以通过从控制台的导航窗格选择 Databases (数据库) 来查看有关数据库集群的详细信息。您还可以查看属于 Amazon Aurora 数据库集群成员的数据库实例的详细信息。

在 Amazon RDS 控制台中查看或修改数据库集群

1. 登录AWS Management Console并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Databases (数据库)。
3. 从列表中选择您要查看的 Aurora 数据库集群的名称。

   例如，下图显示名为 `aurora-test` 的数据库集群的详细信息页。数据库集群在数据库标识符列表中显示了四个数据库实例。写入器数据库实例 `dbinstance4` 是该数据库集群的主数据库实例。

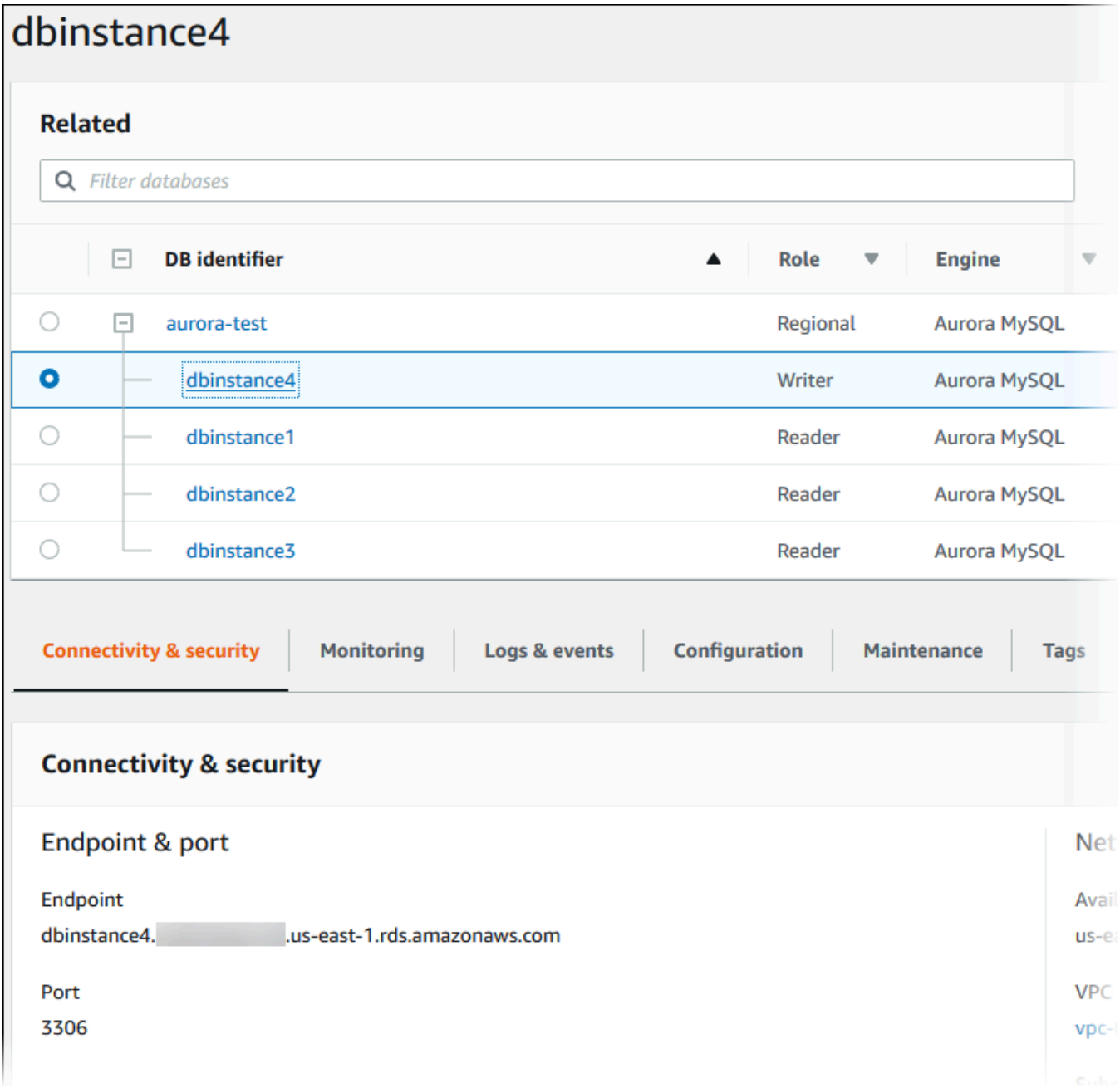


4. 要修改数据库集群，请从列表中选择数据库集群，并选择 Modify（修改）。

在 Amazon RDS 控制台中查看或修改数据库集群的数据库实例

1. 登录AWS Management Console并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Databases (数据库)。
3. 请执行以下操作之一：
   - 要查看数据库实例，请从列表中选择一个属于 Aurora 数据库集群成员的实例。

例如，如果您选择 dbinstance4 数据库实例标识符，控制台将显示 dbinstance4 数据库实例的详细信息页面，如下图所示。



- 要修改数据库实例，请从列表中选择数据库实例，然后选择 Modify（修改）。有关修改数据库集群的更多信息，请参阅 修改 Amazon Aurora 数据库集群。

AWS CLI

要使用 AWS CLI 查看数据库集群信息，请使用 describe-db-clusters 命令。例如，以下 AWS CLI 命令为配置的 us-east-1 账户列出修改 AWS 区域中的所有数据库集群的数据库集群信息。

```
aws rds describe-db-clusters --region us-east-1
```

如果已为 AWS CLI 配置了 JSON 输出，该命令将返回以下输出。

```
{
    "DBClusters": [
        {
            "Status": "available",
            "Engine": "aurora-mysql",
            "Endpoint": "sample-cluster1.cluster-123456789012.us-
east-1.rds.amazonaws.com"
            "AllocatedStorage": 1,
            "DBClusterIdentifier": "sample-cluster1",
            "MasterUsername": "mymasteruser",
            "EarliestRestorableTime": "2023-03-30T03:35:42.563Z",
            "DBClusterMembers": [
                {
                    "IsClusterWriter": false,
                    "DBClusterParameterGroupStatus": "in-sync",
                    "DBInstanceIdentifier": "sample-replica"
                },
                {
                    "IsClusterWriter": true,
                    "DBClusterParameterGroupStatus": "in-sync",
                    "DBInstanceIdentifier": "sample-primary"
                }
            ],
            "Port": 3306,
            "PreferredBackupWindow": "03:34-04:04",
            "VpcSecurityGroups": [
                {
                    "Status": "active",
                    "VpcSecurityGroupId": "sg-ddb65fec"
                }
            ],
            "DBSubnetGroup": "default",
            "StorageEncrypted": false,
            "DatabaseName": "sample",
```

```
            "EngineVersion": "5.7.mysql_aurora.2.11.0",
            "DBClusterParameterGroup": "default.aurora-mysql5.7",
            "BackupRetentionPeriod": 1,
            "AvailabilityZones": [
                "us-east-1b",
                "us-east-1c",
                "us-east-1d"
            ],
            "LatestRestorableTime": "2023-03-31T20:06:08.903Z",
            "PreferredMaintenanceWindow": "wed:08:15-wed:08:45"
        },
        {
            "Status": "available",
            "Engine": "aurora-mysql",
            "Endpoint": "aurora-sample.cluster-123456789012.us-
east-1.rds.amazonaws.com",
            "AllocatedStorage": 1,
            "DBClusterIdentifier": "aurora-sample-cluster",
            "MasterUsername": "mymasteruser",
            "EarliestRestorableTime": "2023-03-30T10:21:34.826Z",
            "DBClusterMembers": [
                {
                    "IsClusterWriter": false,
                    "DBClusterParameterGroupStatus": "in-sync",
                    "DBInstanceIdentifier": "aurora-replica-sample"
                },
                {
                    "IsClusterWriter": true,
                    "DBClusterParameterGroupStatus": "in-sync",
                    "DBInstanceIdentifier": "aurora-sample"
                }
            ],
            "Port": 3306,
            "PreferredBackupWindow": "10:20-10:50",
            "VpcSecurityGroups": [
                {
                    "Status": "active",
                    "VpcSecurityGroupId": "sg-55da224b"
                }
            ],
            "DBSubnetGroup": "default",
            "StorageEncrypted": false,
            "DatabaseName": "sample",
            "EngineVersion": "5.7.mysql_aurora.2.11.0",
```

```
            "DBClusterParameterGroup": "default.aurora-mysql5.7",
            "BackupRetentionPeriod": 1,
            "AvailabilityZones": [
                "us-east-1b",
                "us-east-1c",
                "us-east-1d"
            ],
            "LatestRestorableTime": "2023-03-31T20:00:11.491Z",
            "PreferredMaintenanceWindow": "sun:03:53-sun:04:23"
        }
    ]
}
```

RDS API

要使用 Amazon RDS API 查看数据库集群信息，请使用 DescribeDBClusters 操作。

查看数据库集群状态

数据库集群状态表示其运行状况。您可以使用 Amazon RDS 控制台、AWS CLI 或 API 查看数据库集群和集群实例的状态。

> **Note**
>
> Aurora 还使用名为维护状态的另一种状态，此状态显示在 Amazon RDS 控制台的维护栏中。该值指示需要应用于数据库集群的任何维护修补程序的状态。维护状态独立于数据库集群状态。有关维护状态的更多信息，请参阅对数据库集群应用更新。

在下表中可找到数据库集群的可能状态值。

数据库集群状态	已计费	描述
Available	已计费	数据库集群正常且可用。当 Aurora Serverless 集群可用并暂停时，您只需对存储计费。
Backing-up	已计费	当前正在备份数据库集群。
Backtracking	已计费	当前正在回溯数据库集群。此状态仅适用于 Aurora MySQL。
Cloning-failed	不计费	克隆数据库集群失败。
Creating	不计费	正在创建数据库集群。无法访问正在创建的数据库集群。
Deleting	不计费	正在删除数据库集群。
Failing-over	已计费	正在执行从主实例到 Aurora 副本的失效转移。
Inaccessible-encryption-credentials	不计费	无法访问或恢复用于加密或解密数据库集群的 AWS KMS key。
Inaccessible-encryption-credentials-recoverable	对存储计费	无法访问用于加密或解密数据库集群的 KMS 密钥。但是，如果 KMS 密钥处于活动状态，则重新启动数据库集群可以将其恢复。

数据库集群状态	已计费	描述
		有关更多信息，请参阅加密 Amazon Aurora 数据库集群。
Maintenance	已计费	Amazon RDS 正在对数据库集群应用维护更新。此状态用于 RDS 预先计划的数据库集群级别的维护。
Migrating	已计费	正将数据库集群快照还原到数据库集群。
Migration-failed	不计费	迁移失败。
Modifying	已计费	因为客户请求修改数据库集群，所以正在修改该数据库集群。
Promoting	已计费	正将只读副本提升为独立的数据库集群。
Preparing-data-migration	计费	Amazon RDS 正准备将数据迁移到 Aurora。
Renaming	已计费	按照客户请求正在重命名数据库集群。
Resetting-master-credentials	已计费	按照客户请求正在重置数据库集群的主凭证。
Starting	对存储计费	数据库集群正在启动。
Stopped	对存储计费	已停止数据库集群。
Stopping	对存储计费	正在停止数据库集群。
Storage-optimization	已计费	正在修改数据库实例以更改存储大小或类型。数据库实例完全正常运行。不过，在数据库实例状态为 storage-optimization (存储优化) 时，您无法请求对数据库实例的存储进行任何更改。存储优化过程通常很短，但有时可能会达到甚至超过 24 小时。
Update-iam-db-auth	已计费	正在更新数据库集群的 IAM 授权。

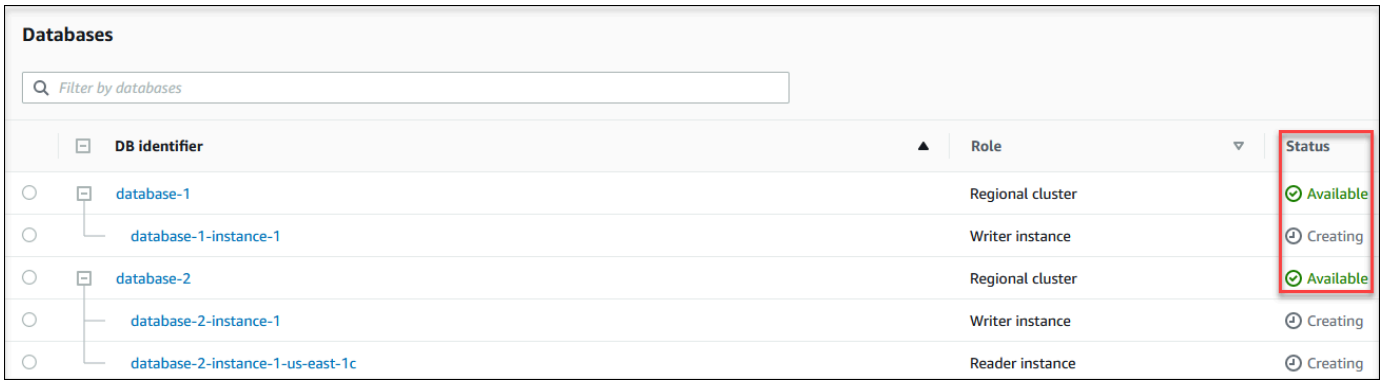
数据库集群状态	已计费	描述
Upgrading	计费	数据库集群引擎或操作系统版本正在升级。

控制台

查看数据库集群的状态

1. 登录AWS Management Console并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Databases (数据库)。

   将显示 Databases（数据库）页面以及数据库集群的列表。对于每个数据库集群，显示状态值。



CLI

要仅查看数据库集群的状态，请在 AWS CLI 中使用以下查询。

```
aws rds describe-db-clusters --query 'DBClusters[*].[DBClusterIdentifier,Status]' --
output table
```

查看 Aurora 集群中的数据库实例状态

Aurora 集群 中的数据库实例的状态表示数据库实例的运行状况。您可以使用以下过程在 Amazon RDS 控制台、AWS CLI 命令或 API 操作中查看集群的数据库实例状态。

> **Note**
>
> Amazon RDS 还使用名为维护状态的另一种状态，此状态显示在 Amazon RDS 控制台的维护栏中。该值指示需要应用于数据库实例的任何维护修补程序的状态。维护状态独立于数据库实例状态。有关维护状态的更多信息，请参阅对数据库集群应用更新。

在下表中可找到数据库实例的可能状态值。此表还会显示是否对数据库实例和存储或者仅对存储向您计费，还是不向您计费。对于所有数据库实例状态，始终会针对备份使用向您计费。

数据库实例状态	已计费	描述
available	已计费	数据库实例正常且可用。
backing-up	已计费	当前正在备份数据库实例。
backtracking	已计费	当前正在回溯数据库实例。此状态仅适用于 Aurora MySQL。
configuring-enhanced-monitoring	已计费	正在对此数据库实例启用或禁用增强监控。
configuring-iam-database-auth	已计费	正在对此数据库实例启用或禁用 AWS Identity and Access Management（IAM）数据库身份验证。
configuring-log-exports	已计费	正在对此数据库实例启用或禁用向 Amazon CloudWatch Logs 发布日志文件。
converting-to-vpc	已计费	数据库实例正在从不在 Amazon Virtual Private Cloud (Amazon VPC) 中的数据库实例转换为在 Amazon VPC 中的数据库实例。
creating	不计费（非 PITR）	正在创建数据库实例。无法访问正在创建的数据库实例。

数据库实例状态	已计费	描述
	已计费（仅限 PITR）	如果您在时间点故障恢复（PITR）期间还原数据库，则会在数据库处于 creating 状态时对您计费。这是 creating 状态产生费用的唯一场景。
delete-precheck	不计费	Amazon RDS 正在验证只读副本是否正常运行且可以安全删除。
deleting	不计费	正在删除数据库实例。
failed	不计费	数据库实例已失败，Amazon RDS 无法恢复它。执行时间点还原，以还原至数据库实例的最近可还原时间，从而恢复数据。
inaccessible-encryption-credentials	不计费	无法访问或恢复用于加密或解密数据库实例的 AWS KMS key。
inaccessible-encryption-credentials-recoverable	对存储计费	无法访问用于加密或解密数据库实例的 KMS 密钥。但是，如果 KMS 密钥处于活动状态，则重新启动数据库实例可以恢复它。 有关更多信息，请参阅 加密 Amazon Aurora 数据库集群。
incompatible-create	不计费	Amazon RDS 正尝试创建数据库实例，但无法执行此操作，因为资源与您的数据库实例不兼容。这种情况在一定条件下会发生，例如，当数据库实例的实例配置文件没有正确的权限时。
incompatible-network	不计费	Amazon RDS 正尝试对数据库实例执行恢复操作，但无法执行此操作，因为 VPC 正处于一种阻止此操作完成的状态。例如，如果子网中的所有可用 IP 地址都在使用中，并且 Amazon RDS 无法为数据库实例获取 IP 地址，就会出现此状态。

数据库实例状态	已计费	描述
incompatible-option-group	已计费	Amazon RDS 尝试应用选项组更改，但却无法执行，并且 Amazon RDS 无法回滚到选项组之前的状态。有关更多信息，请查看数据库实例的近期事件列表。例如，如果选项组包含一个诸如 TDE 的选项以及数据库实例不包含加密信息时，上述情况可能会发生。
incompatible-parameters	已计费	Amazon RDS 无法启动数据库实例，因为在数据库实例的数据库参数组中指定的参数与数据库实例不兼容。恢复参数更改或使这些更改与数据库实例相兼容以重新访问数据库实例。有关不兼容参数的更多信息，请查看数据库实例的近期事件列表。
incompatible-restore	不计费	Amazon RDS 无法执行时间点还原。此状态的常见原因包括使用临时表 或使用带 MySQL 的 MyISAM 表。
insufficient-capacity	不计费	由于目前容量不足，Amazon RDS 无法创建实例。要在同一可用区中创建同一实例类型的数据库实例，请删除数据库实例，等待几个小时后，再尝试重新创建。或者，使用其他实例类或可用区创建新实例。
maintenance	已计费	Amazon RDS 正在对数据库实例应用维护更新。此状态用于 RDS 预先计划的实例级别的维护。
modifying	已计费	正在按照客户的请求修改数据库实例。
moving-to-vpc	已计费	数据库实例正移至新的 Amazon Virtual Private Cloud (Amazon VPC)。
rebooting	已计费	按照客户请求或需要重启数据库实例的 Amazon RDS 过程正在重启数据库实例。
resetting-master-credentials	已计费	正在按照客户请求重置数据库实例的主凭证。
renaming	已计费	正在按照客户请求重命名数据库实例。

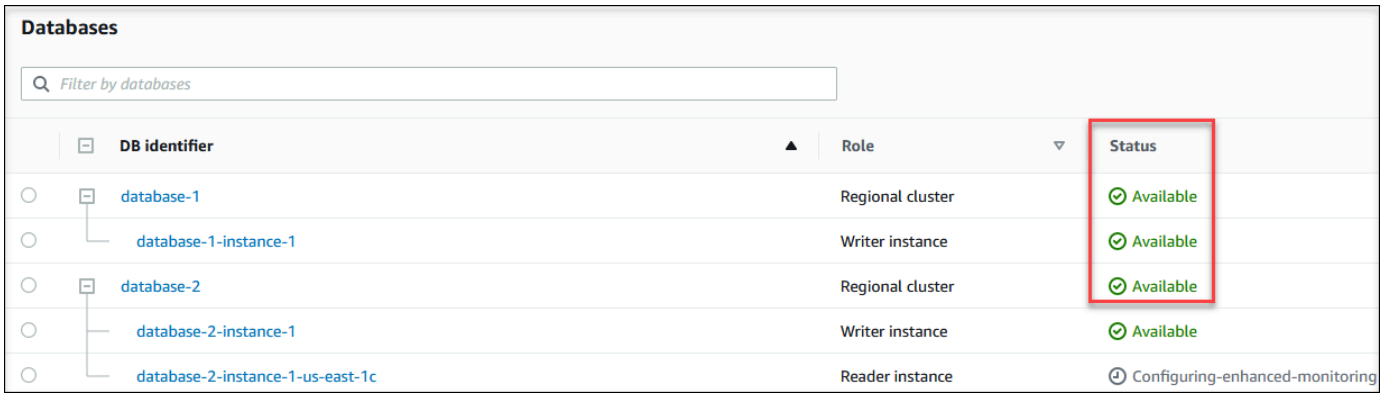
数据库实例状态	已计费	描述
restore-error	已计费	数据库实例在尝试还原到某个时间点或从快照还原时遇到错误。
starting	对存储计费	数据库实例正在启动。
stopped	对存储计费	数据库实例已停止。
stopping	对存储计费	正在停止数据库实例。
storage-config-upgrade	计费	正在升级数据库实例的存储文件系统配置。此状态仅适用于蓝绿部署中的绿色数据库或数据库实例只读副本。
storage-full	已计费	数据库实例达到了其存储分配容量。这是一种严重状态，我们推荐您立即修复该问题。为此，请通过修改数据库实例来扩展存储。要避免这种情况，请将 Amazon CloudWatch 警报设置为当存储空间逐渐减小时向您发出警告。
storage-initialization	计费	从快照还原后，数据库实例正在从 Amazon S3 加载数据块，以优化卷性能。它仍然可供操作，但在初始化完成之前，性能可能无法达到最佳。
storage-optimization	计费	Amazon RDS 正在优化数据库实例的存储。存储优化过程通常很短，但有时可能会达到甚至超过 24 小时。 在存储优化期间，数据库实例保持可用。存储优化进程在后台运行，不会影响实例的可用性。
upgrading	计费	数据库引擎或操作系统版本正在升级。

控制台

查看数据库实例的状态：

1. 登录AWS Management Console并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Databases (数据库)。

   将显示 Databases（数据库）页面以及数据库实例的列表。对于集群中的每个数据库实例，显示状态值。



CLI

要使用 AWS CLI 查看数据库实例及其状态信息，请使用 describe-db-instances 命令。例如，以下 AWS CLI 命令可列出所有数据库实例信息。

```
aws rds describe-db-instances
```

要查看特定数据库实例及其状态，请带以下选项调用 describe-db-instances 命令：

- `DBInstanceIdentifier` – 数据库实例的名称。

```
aws rds describe-db-instances --db-instance-identifier mydbinstance
```

要只是查看所有数据库集群的状态，请在 AWS CLI 中使用以下查询。

```
aws rds describe-db-instances --query 'DBInstances[*].
[DBInstanceIdentifier,DBInstanceStatus]' --output table
```

API

要使用 Amazon RDS API 查看数据库实例的状态，请调用 DescribeDBInstances 操作。

来自 Amazon Aurora 的建议

Amazon Aurora 为数据库资源（例如数据库实例、数据库集群、和数据库参数组）提供自动建议。这些建议通过分析数据库集群配置、数据库实例配置、使用和性能数据来提供最佳实践准则。

Amazon RDS 性能详情可监控特定指标，并自动通过分析指定资源可能存在问题的级别来创建阈值。当新的指标值在给定时间段内超过预定义的阈值时，性能详情会生成主动建议。此建议有助于防止数据库性能将来受到影响。例如，当连接到数据库的会话没有执行活动的工作，但可以保持阻止数据库资源时，将为 Aurora PostgreSQL 实例生成“空闲事务”建议。要获得主动建议，您必须开启性能详情，并设置付费套餐保留期。有关开启性能详情的信息，请参阅为 Aurora 打开和关闭性能详情。有关性能详情的定价和数据留存的更多信息，请参阅性能详情的定价和数据留存。

DevOps Guru for RDS 会监控某些指标，以检测指标的行为何时变得高度不寻常或异常。这些异常被报告为带有建议的被动见解。例如，DevOps Guru for RDS 可能会建议您考虑增加 CPU 容量或调查导致数据库负载的等待事件。DevOps Guru for RDS 还提供基于阈值的主动建议。对于这些建议，您必须开启 DevOps Guru for RDS。有关开启 DevOps Guru for RDS 的信息，请参阅打开 DevOps Guru 并指定资源覆盖范围。

建议将处于以下任何状态：活动、已忽略、待处理或已解决。已解决的建议有效期为 365 天。

您可以查看或取消建议。您可以立即应用基于配置的活动建议，将其安排在下一个维护时段，或将其忽略。对于基于阈值的主动建议和基于机器学习的被动建议，您需要查看建议的问题原因，然后执行建议的操作来修复问题。

在以下 AWS 区域中支持建议：

- 美国东部（俄亥俄州）
- 美国东部（弗吉尼亚州北部）
- 美国西部（加利福尼亚北部）
- 美国西部（俄勒冈州）
- 亚太地区（孟买）
- 亚太地区（首尔）
- 亚太地区（新加坡）
- 亚太地区（悉尼）
- 亚太地区（东京）
- 加拿大（中部）
- 欧洲地区（法兰克福）

- 欧洲地区（爱尔兰）
- 欧洲地区（伦敦）
- 欧洲地区（巴黎）
- 欧洲地区（斯德哥尔摩）
- 南美洲（圣保罗）

在以下各节中，学习查看、应用、忽略和修改来自 Amazon Aurora 的建议。

主题

- 查看 Amazon Aurora 建议
- 应用 Amazon Aurora 建议
- 忽略 Amazon Aurora 建议
- 将已忽略的 Amazon Aurora 建议修改为活动的建议
- Amazon Aurora 中的建议参考

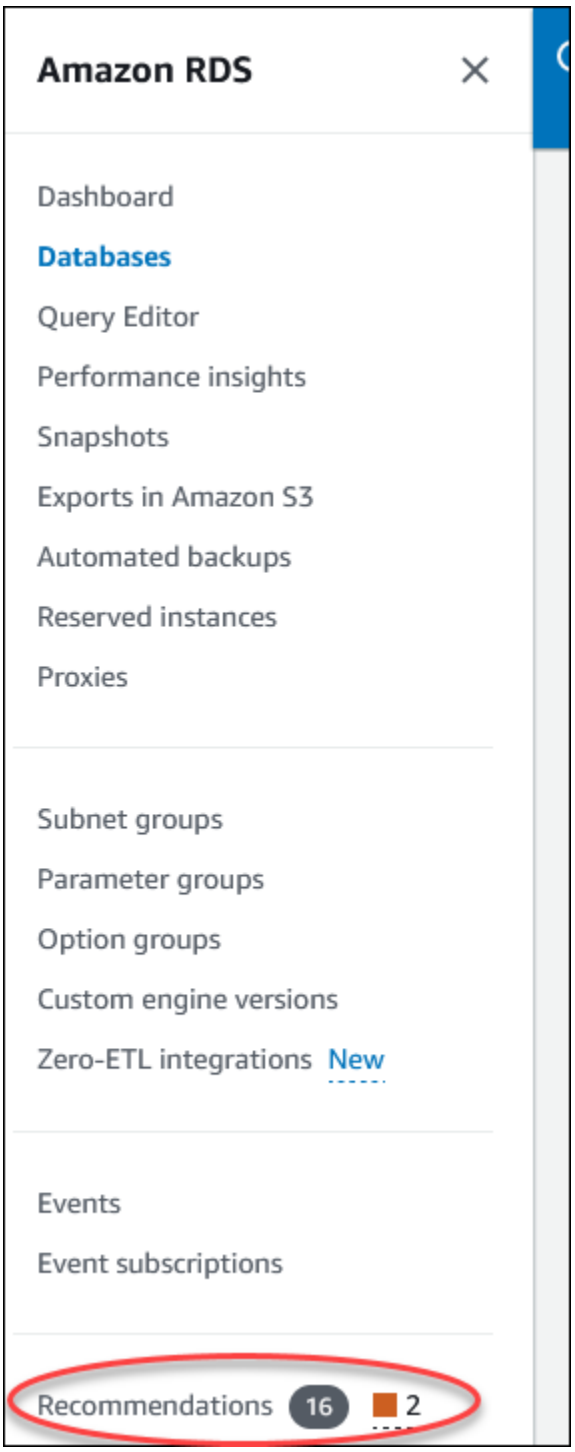
查看 Amazon Aurora 建议

使用 Amazon RDS 控制台，您可以查看 Amazon Aurora 针对您的数据库资源提出的建议。对于数据库集群，将显示针对该数据库集群及其实例的建议。

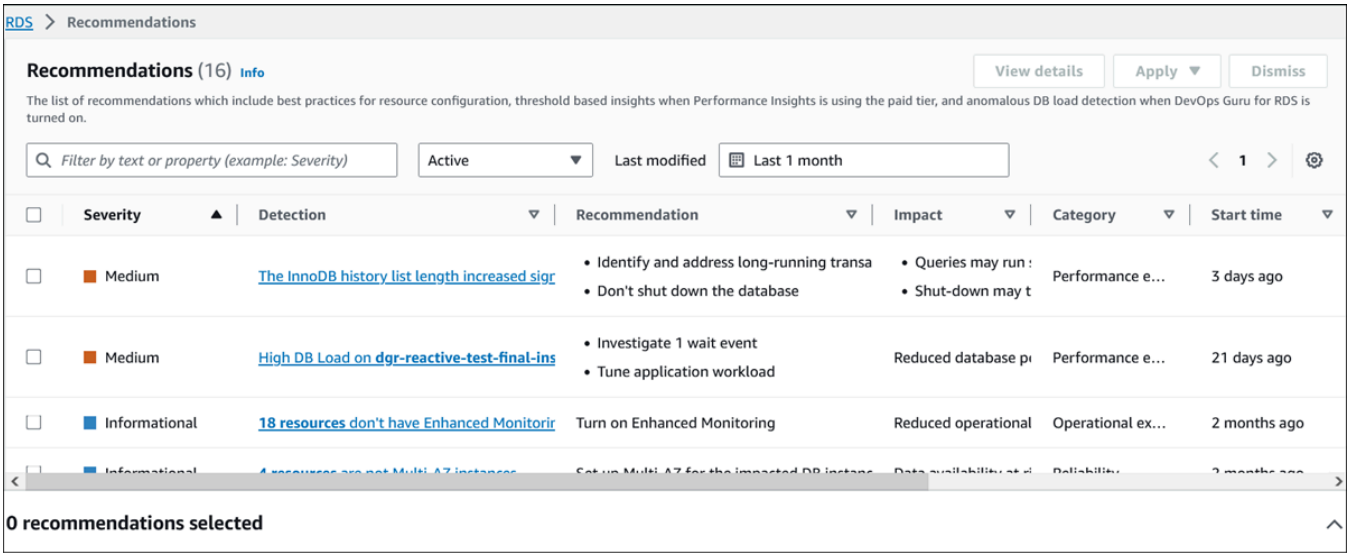
控制台

查看 Amazon Aurora 建议

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，执行以下任何一项操作：
   - 选择建议。建议旁边提供了针对您的资源的有效建议数量和上个月生成的严重性最高的建议数量。要查找每个严重性的有效建议数量，请选择显示最高严重性的数字。

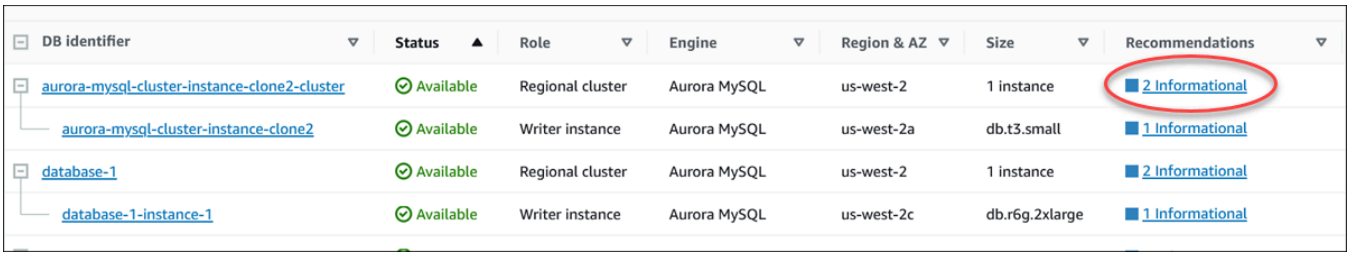


默认情况下，建议页面显示上个月的新建议列表。Amazon Aurora 会针对您账户中的所有资源提供建议，并按其严重性对建议进行排序。

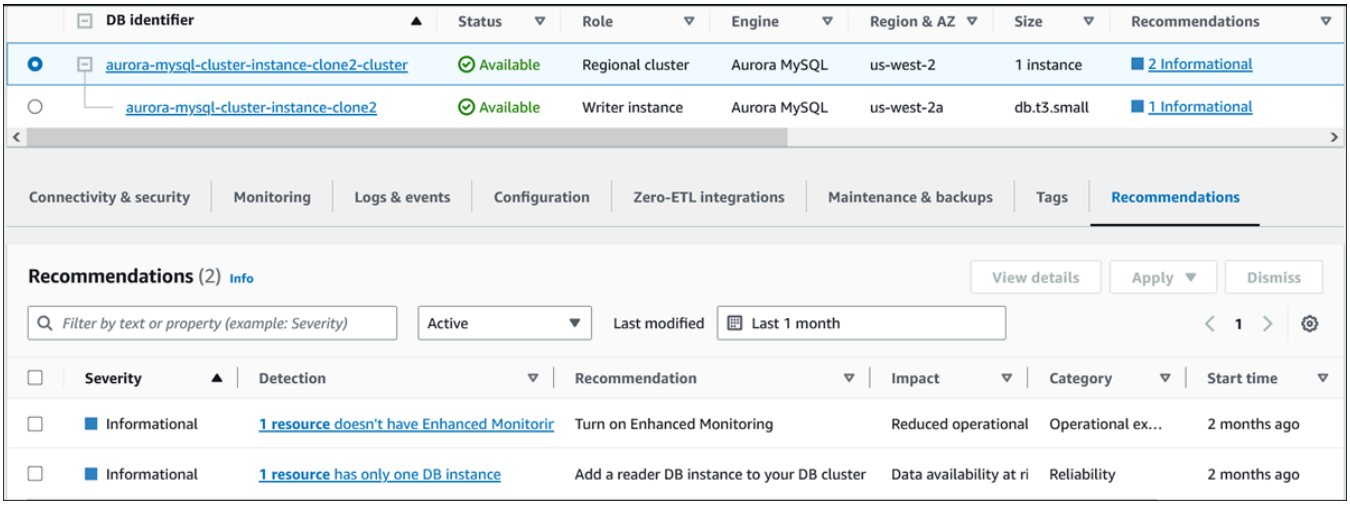


您可以选择建议以查看页面底部的部分，其中包含受影响的资源以及如何应用该建议的详细信息。

- 在数据库页中，为资源选择建议。



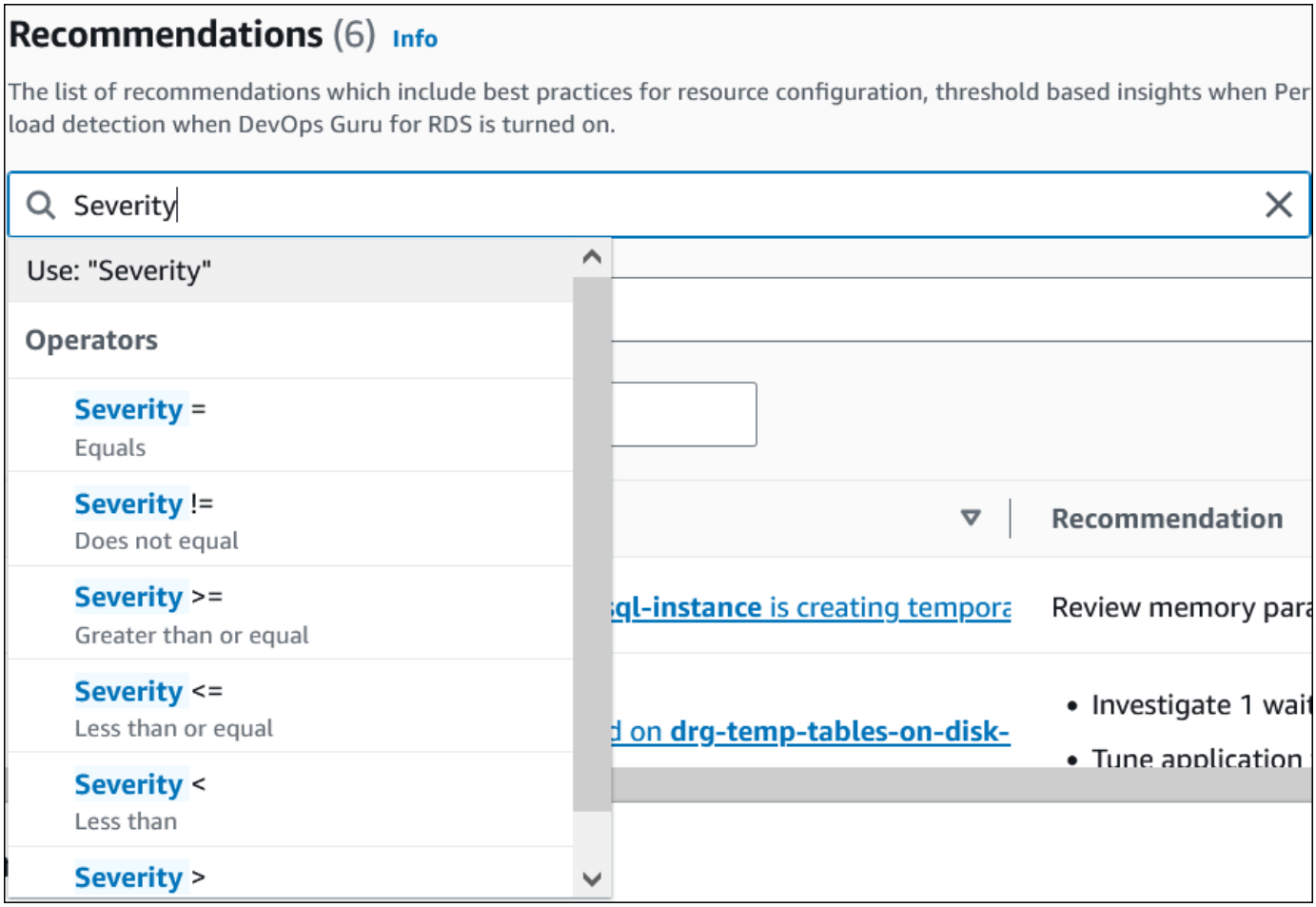
建议选项卡显示所选资源的建议及其详细信息。



以下详细信息可用于建议：

- 严重性 - 问题的含义级别。严重性级别为高、中、低和信息。
- 检测 - 受影响资源的数量和问题的简短描述。选择此链接可查看建议和分析详细信息。
- 建议 – 要应用的建议操作的简短描述。
- 影响 – 对不应用建议时可能产生的影响的简短描述。
- 类别 - 建议的类型。这些类别包括性能效率、安全性、可靠性、成本优化、卓越运营和可持续性。
- 状态 – 建议的当前状态。可能的状态为全部、活动、已忽略、已解决和待处理。
- 开始时间 – 问题开始的时间。例如，18 小时前。
- 上次修改时间 - 由于严重性更改而导致系统上次更新建议的时间，或者您对建议做出回应的时间。例如，10 小时前。
- 结束时间 - 问题结束的时间。对于任何持续存在的问题，不会显示此时间。
- 资源标识符 - 一个或多个资源的名称。

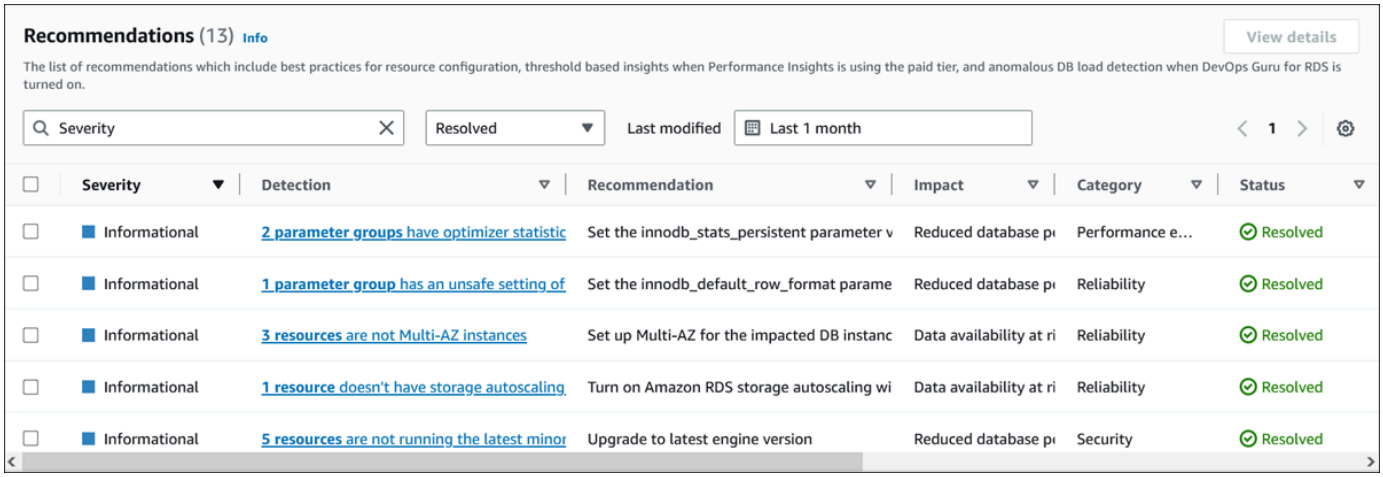
3. （可选）在字段中选择严重性或类别运算符以筛选建议列表。



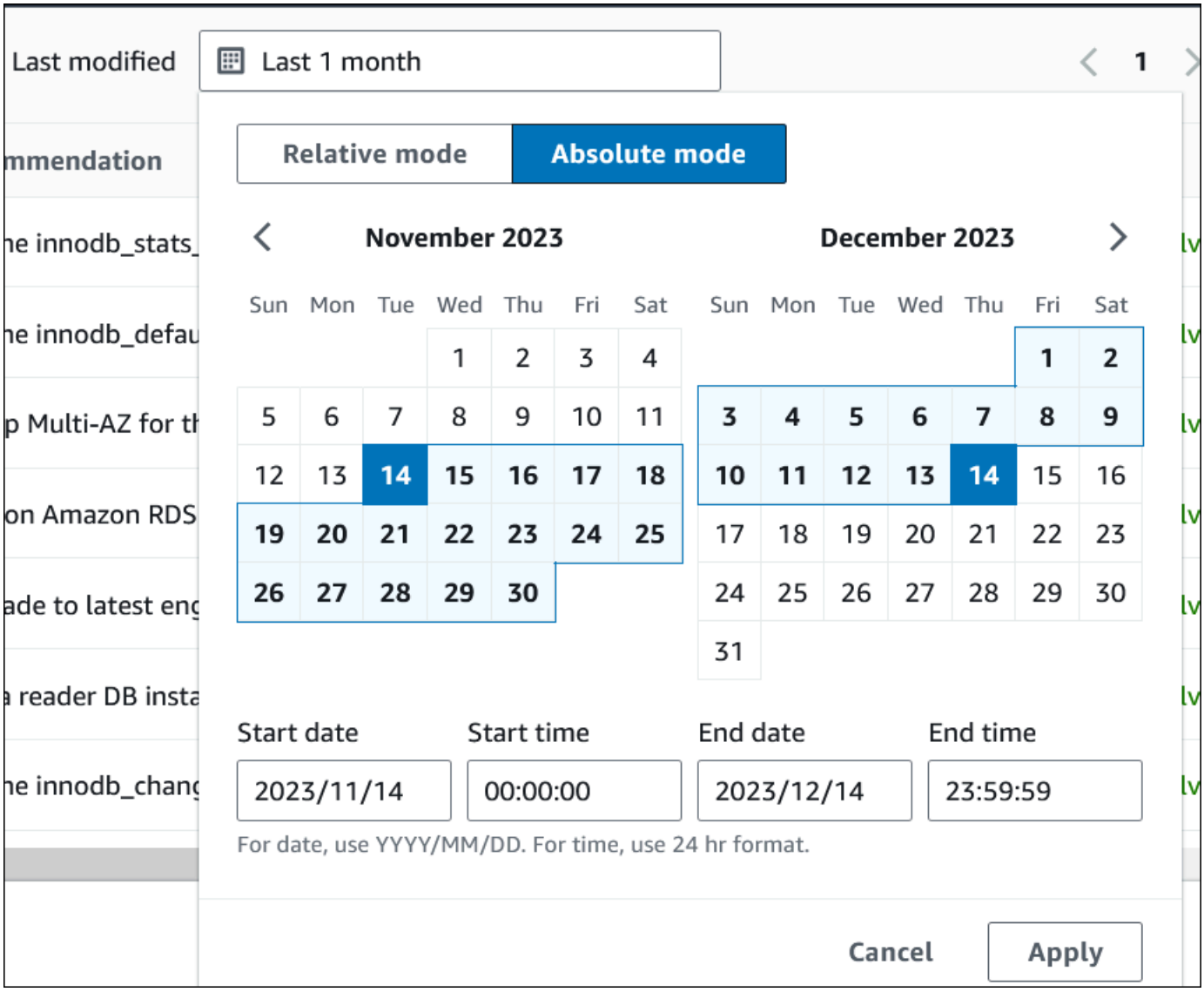
此时将显示所选操作的建议。

4. （可选）选择以下任何一个建议状态：

   - 活动（默认）– 显示您可以应用、计划在下一个维护时段应用或忽略的当前建议。
   - 全部 - 显示当前状态的所有建议。
   - 已忽略 - 显示已忽略的建议。
   - 已解决 - 显示已解决的建议。
   - 待处理 - 显示其建议的操作正在进行中或计划在下一个维护时段应用的建议。



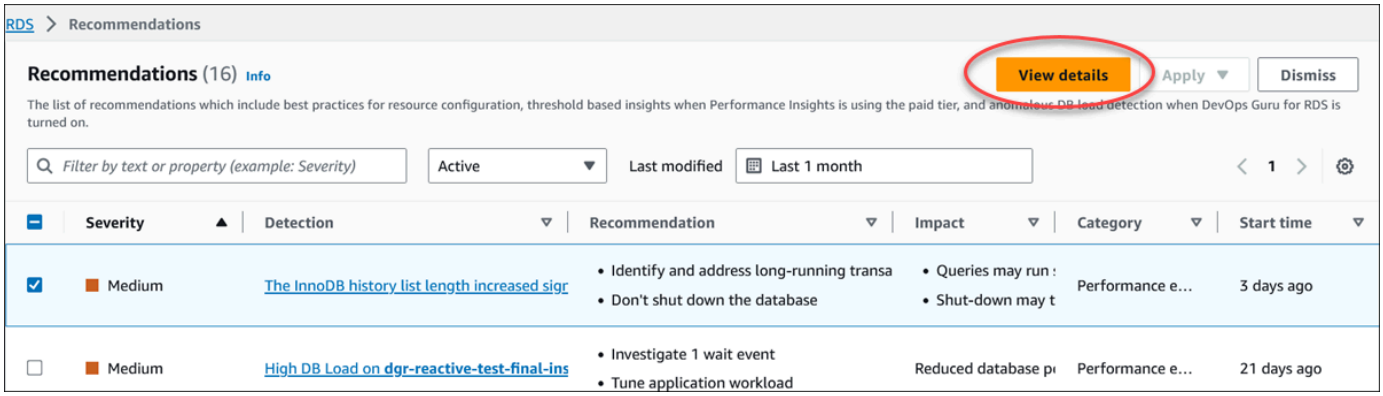
5. （可选）在上次修改时间中选择相对模式或绝对模式，以修改时间段。建议页面显示该时间段内生成的建议。默认时间段为最近 1 个月。在绝对模式下，您可以选择时间段，或者在开始日期和结束日期字段中输入时间。



将显示设定时间段的建议。

请注意，将范围设置为全部，即可查看您账户中资源的所有相关建议。

6. （可选）选择右侧的首选项以自定义要显示的详细信息。您可以选择页面大小、对文本换行以及允许或隐藏列。
7. （可选）选择建议，然后选择查看详细信息。



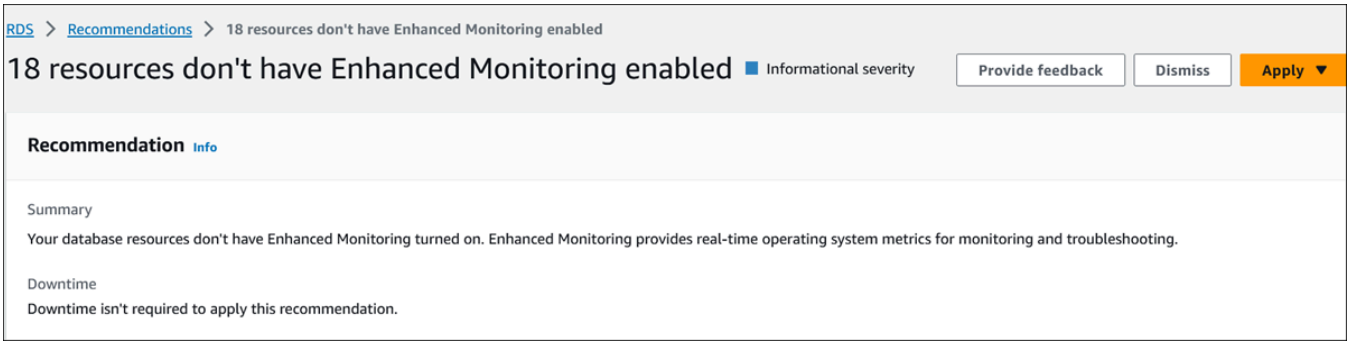
此时会显示建议详细信息页面。标题提供了检测到问题的资源总数和严重性。

有关基于异常的被动建议的详细信息页面上组件的信息，请参阅《Amazon DevOps Guru User Guide》中的 Viewing reactive anomalies。

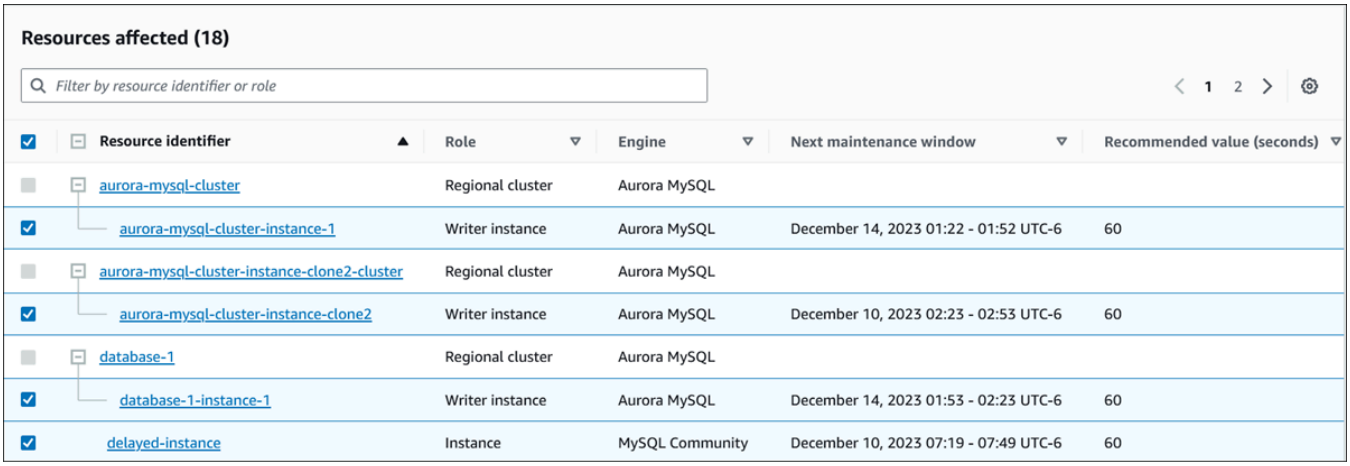
有关基于阈值的主动建议的详细信息页面上组件的信息，请参阅查看性能详情主动建议。

其它自动建议在建议详细信息页面上显示以下组件：

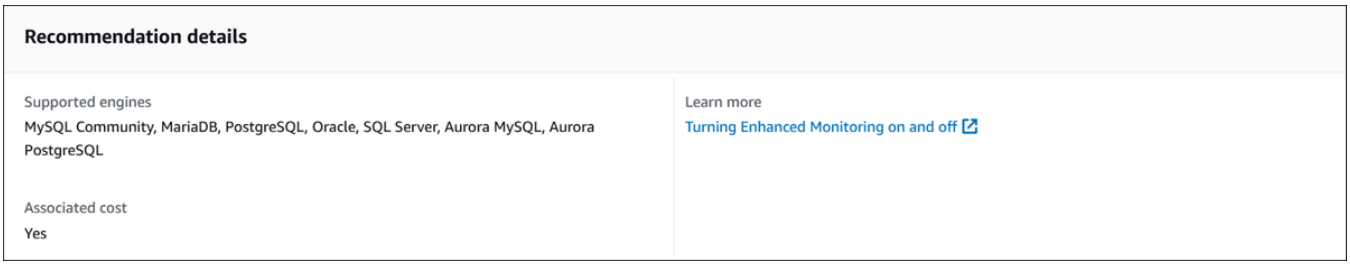
- 建议 – 建议的摘要以及是否需要停机才能应用建议。



- 受影响的资源 - 受影响资源的详细信息。



- 建议详细信息 – 支持的引擎信息、应用建议所需的任何相关费用以及用于了解更多信息的文档链接。



CLI

要查看数据库实例或数据库集群的 Amazon RDS 建议，请在 AWS CLI 中使用以下命令。

```
aws rds describe-db-recommendations
```

RDS API

要使用 Amazon RDS API 查看 Amazon RDS 建议，请使用 DescribeDBRecommendations 操作。

应用 Amazon Aurora 建议

要使用 Amazon RDS 控制台应用 Amazon Aurora 建议，请在详细信息页面中选择基于配置的建议或受影响的资源。然后，选择立即应用该建议，或将其安排在下一个维护时段。资源可能需要重启才能使更改生效。对于一些数据库参数组建议，您可能需要重启资源。

基于阈值的主动建议或基于异常的被动建议不具有应用选项，可能需要进一步审查。

控制台

应用基于配置的建议

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，执行以下任一操作：

   - 选择建议。

     此时将出现建议页面，其中包含所有建议的列表。
   - 选择数据库，然后在数据库页中为资源选择建议。

详细信息显示在所选建议的建议选项卡中。

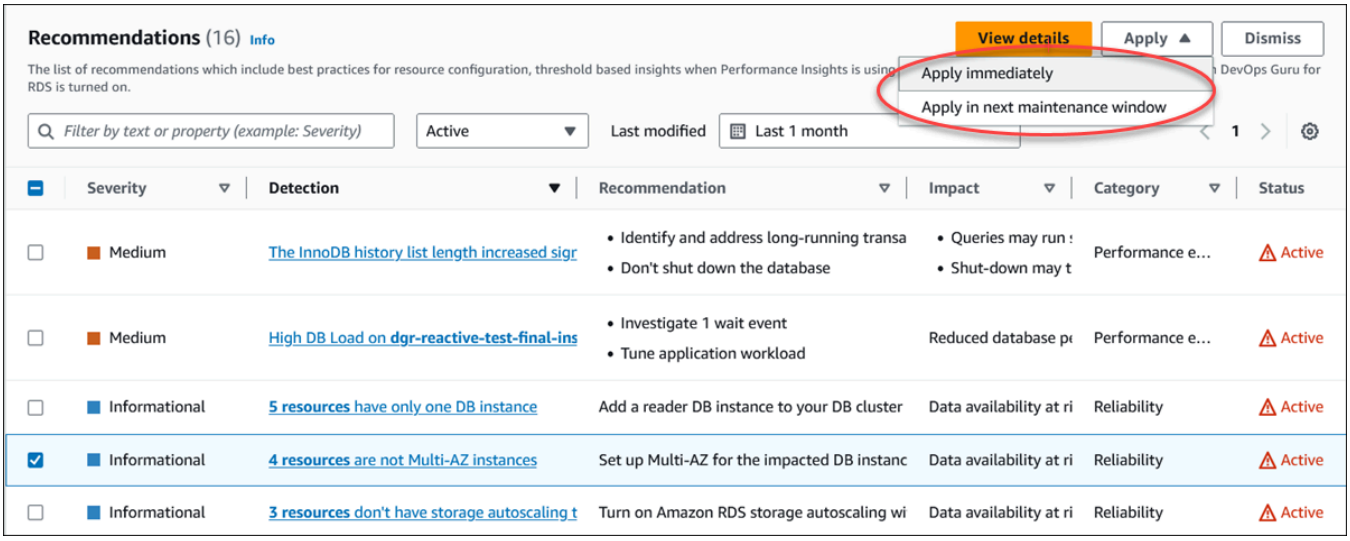
- 在建议页或数据库页的建议选项卡中，为活动的建议选择检测。

  此时会显示建议详细信息页面。

3. 在建议详细信息页面中选择一个建议或一个或多个受影响的资源，然后执行以下任一操作：

   - 选择应用，然后选择立即应用以立即应用建议。
   - 选择应用，然后选择在下一个维护时段内应用以计划在下一个维护时段中应用。

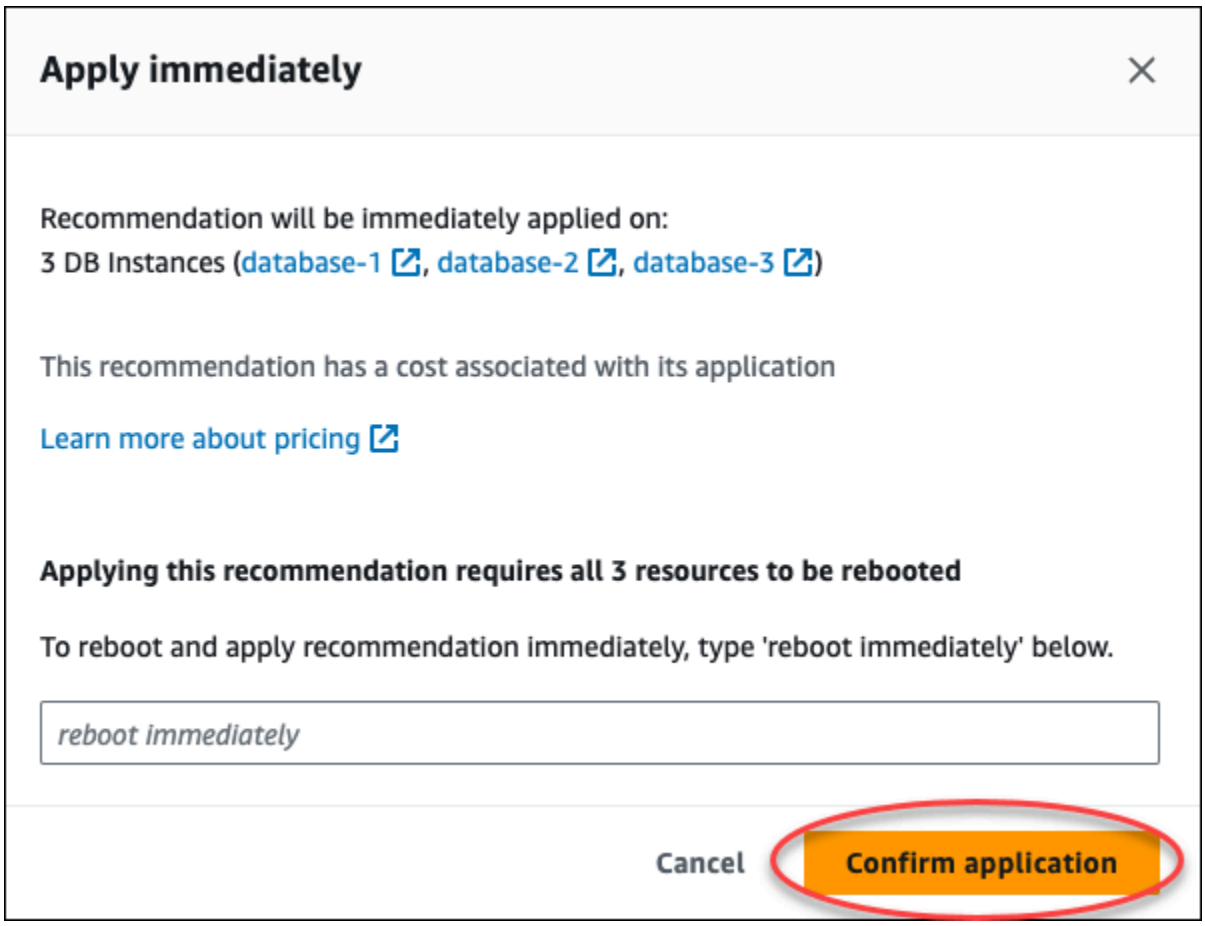
   在到达下一个维护时段之前，所选建议的状态将更新为待定。



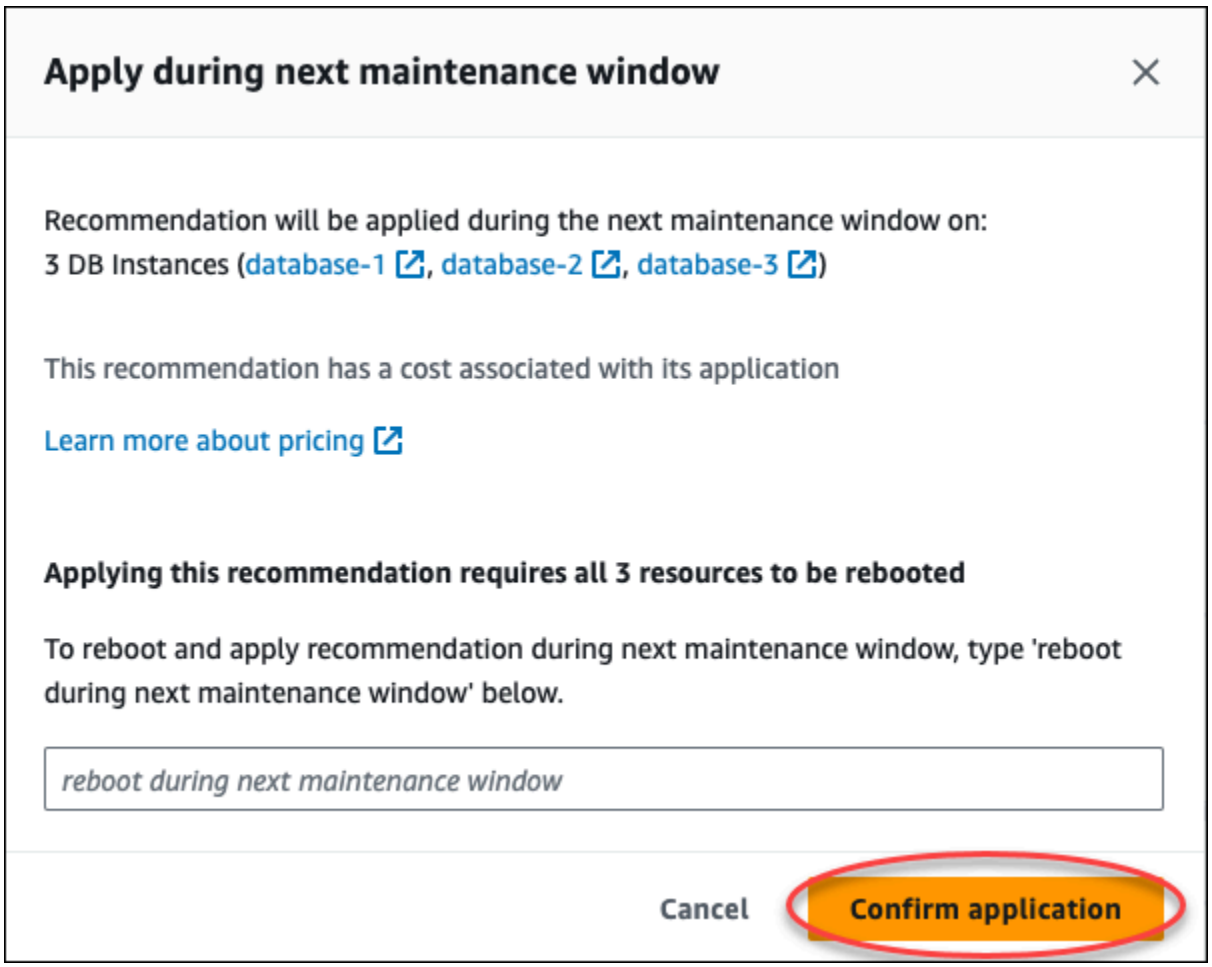
将显示一个确认窗口。

4. 选择确认应用以应用建议。此窗口将确认资源是需要自动重启还是手动重启才能使更改生效。

   以下示例显示了立即应用建议的确认窗口。

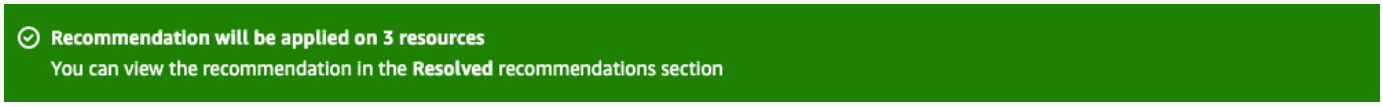


以下示例显示了计划在下一个维护时段中应用建议的确认窗口。



当应用的建议成功或失败时，横幅会显示一条消息。

以下示例显示了带有成功消息的横幅。



以下示例显示了带有失败消息的横幅。



RDS API

使用 Amazon RDS API 应用基于配置的 Aurora 建议

1. 使用 DescribeDBRecommendations 操作。输出中的 RecommendedActions 可以有一个或多个建议的操作。
2. 对步骤 1 中的每个建议操作使用 RecommendedAction 对象。输出包含 Operation 和 Parameters。

   以下示例显示了带有一个建议操作的输出。

```
   "RecommendedActions": [
       {
           "ActionId": "0b19ed15-840f-463c-a200-b10af1b552e3",
           "Title": "Turn on auto backup", // localized
           "Description": "Turn on auto backup for my-mysql-instance-1", //
localized
           "Operation": "ModifyDbInstance",
           "Parameters": [
               {
                   "Key": "DbInstanceIdentifier",
                   "Value": "my-mysql-instance-1"
               },
               {
                   "Key": "BackupRetentionPeriod",
                   "Value": "7"
               }
           ],
           "ApplyModes": ["immediately", "next-maintenance-window"],
           "Status": "applied"
       },
       ... // several others
   ],
```

3. 对步骤 2 的输出中的每个建议操作使用 operation 并输入 Parameters 值。
4. 步骤 2 中的操作成功后，使用 ModifyDBRecommendation 操作修改建议状态。

忽略 Amazon Aurora 建议

可以使用 Amazon RDS 控制台、AWS CLI 或 Amazon RDS API 忽略一个或多个 Amazon Aurora 建议。

控制台

忽略一个或多个建议

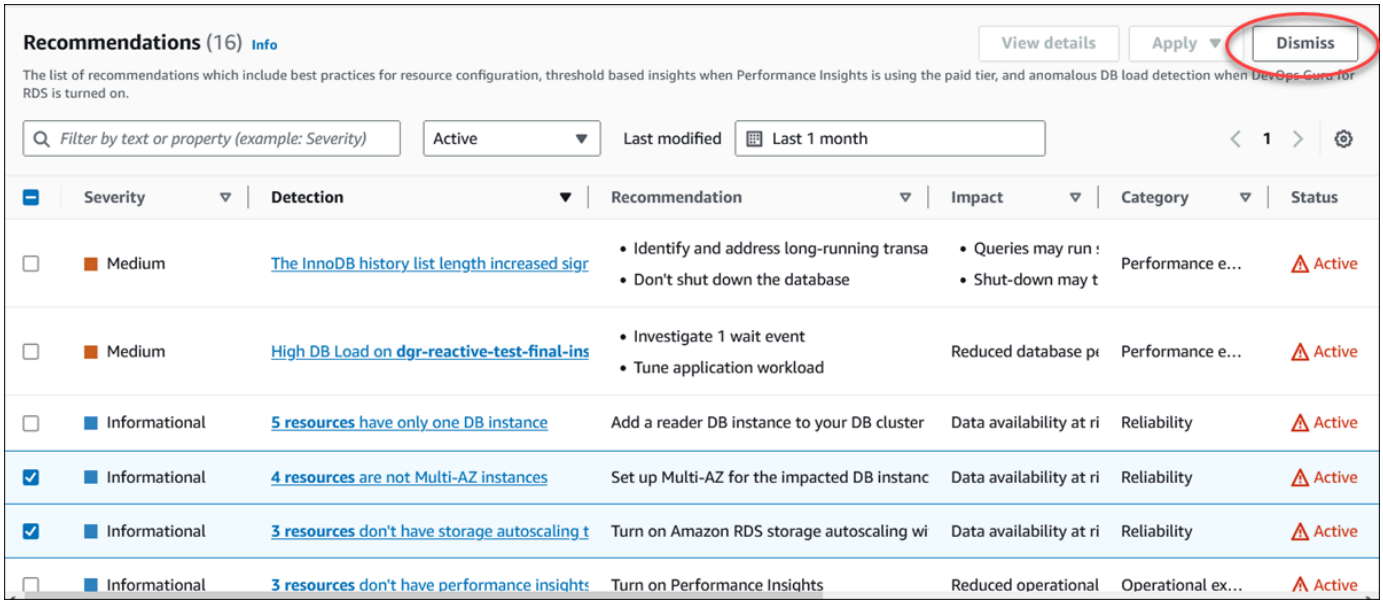
1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，执行以下任一操作：
   - 选择建议。

     此时将出现建议页面，其中包含所有建议的列表。
   - 选择数据库，然后在数据库页中为资源选择建议。

     详细信息显示在所选建议的建议选项卡中。
   - 在建议页或数据库页的建议选项卡中，为活动的建议选择检测。

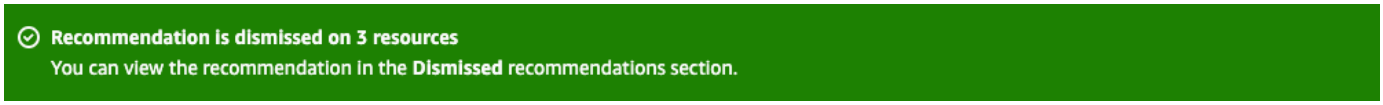
     建议详细信息页面显示受影响资源的列表。
3. 在建议详细信息页面中选择一个或多个建议，或者选择一个或多个受影响的资源，然后选择忽略。

   以下示例显示了建议页面，其中选定了多个要取消的活动建议。



当选定的一个或多个建议被忽略时，横幅会显示一条消息。

以下示例显示了带有成功消息的横幅。



以下示例显示了带有失败消息的横幅。



CLI

使用 AWS CLI 忽略 Aurora 建议

1. 运行 `aws rds describe-db-recommendations --filters "Name=status,Values=active"` 命令。

   输出提供了处于 `active` 状态的建议列表。
2. 查找要从步骤 1 中忽略的建议的 `recommendationId`。
3. 从步骤 2 中使用 `recommendationId` 运行命令 `>aws rds modify-db-recommendation --status dismissed --recommendationId <ID>` 以忽略建议。

RDS API

要使用 Amazon RDS API 忽略 Aurora 建议，请使用 ModifyDBRecommendation 操作。

将已忽略的 Amazon Aurora 建议修改为活动的建议

可以使用 Amazon RDS 控制台、AWS CLI 或 Amazon RDS API，将一个或多个已忽略的 Amazon Aurora 建议移至活动的建议。

控制台

将一个或多个已被忽略的建议移至活动的建议

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。

2. 在导航窗格中，执行以下任一操作：

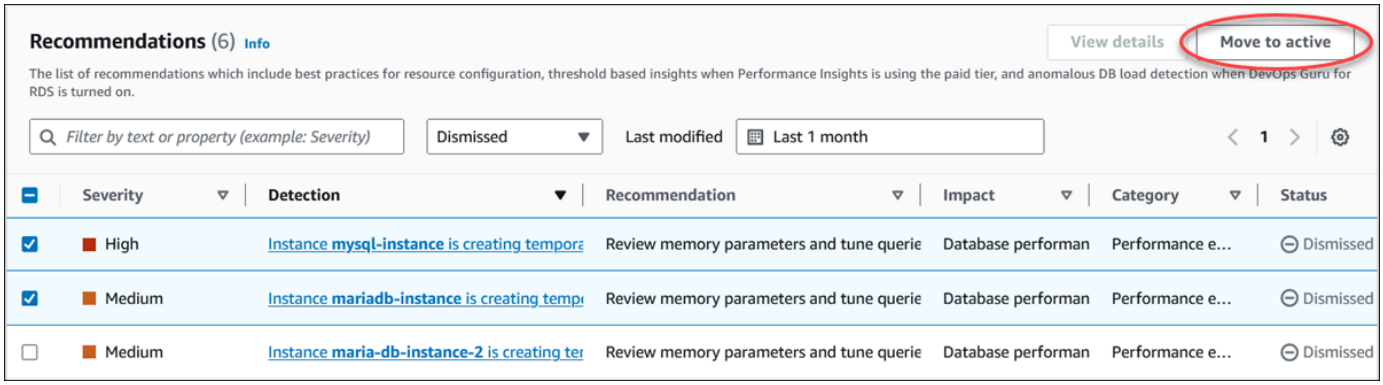
   - 选择建议。

     建议页面显示按您账户中所有资源的严重性排序的建议列表。

   - 选择数据库，然后在数据库页中为资源选择建议。

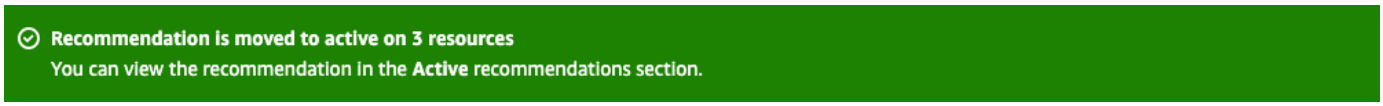
     建议选项卡显示所选资源的建议及其详细信息。

3. 从列表中选择一个或多个已忽略的建议，然后选择移至活动状态。



当将所选建议从已忽略状态移至活动状态时，横幅会显示一条成功或失败消息。

以下示例显示了带有成功消息的横幅。



以下示例显示了带有失败消息的横幅。



CLI

使用 AWS CLI 将已忽略的 Aurora 建议更改为活动建议

1. 运行 `aws rds describe-db-recommendations --filters "Name=status,Values=dismissed"` 命令。

   输出提供了处于 `dismissed` 状态的建议列表。

2. 查找要从步骤 1 中更改状态的建议的 `recommendationId`。

3. 从步骤 2 中使用 recommendationId 运行命令 >aws rds modify-db-recommendation --status active --recommendationId <ID> 以更改为活动建议。

RDS API

要使用 Amazon RDS API 将已忽略的 Aurora 建议更改为活动建议，请使用 ModifyDBRecommendation 操作。

Amazon Aurora 中的建议参考

Amazon Aurora 在创建或修改资源时，为资源生成建议。可以在下表中找到 Amazon Aurora 中的建议示例。

类型	描述	建议	需要停机	其他信息
资源自动备份已关闭	您的数据库实例没有开启自动备份。建议使用自动备份，因为它们可以实现数据库实例的时间点恢复。	开启自动备份，保留期最长为 14 天。	是	备份和还原 Aurora 数据库集群的概述 AWS 数据库博客上的 Demystifying Amazon RDS backup storage costs
需要引擎次要版本升级	您的数据库资源未运行最新次要数据库引擎版本。最新的次要版本包含最新的安全修复和其它改进。	升级到最新的引擎版本。	是	维护 Amazon Aurora 数据库集群
增强监控已关闭	您的数据库资源未开启增强监控。增强监控提供用于监控和故障排除的实时操作系统指标。	打开增强监控。	否	使用增强监控来监控操作系统指标
存储加密已关闭	Amazon RDS 支持使用您在 AWS Key Managemen	为数据库集群开启静态数据加密。	是	Amazon Aurora 中的安全性

类型	描述	建议	需要停机	其他信息
	t Service（AWS KMS）中管理的密钥对所有数据库引擎进行静态加密。在采用 Amazon RDS 加密的活动数据库实例上，静态存储在存储中的数据会加密，类似于自动备份、只读副本和快照。 如果在创建 Aurora 数据库集群时未开启加密，则必须将解密后的快照还原到加密的数据库集群。			
所有实例都在同一可用区中的数据库集群	数据库集群目前位于单个可用区中。使用多个可用区来提高可用性。	将数据库实例添加到数据库集群中的多个可用区。	否	Amazon Aurora 的高可用性
集群中具有异构实例大小的数据库实例	建议您对数据库集群中的所有数据库实例使用相同的数据库实例类和大小。	对数据库集群中的所有数据库实例使用相同的实例类和大小。	是	使用 Amazon Aurora 进行复制
集群中具有异构实例类的数据库实例	建议您对数据库集群中的所有数据库实例使用相同的数据库实例类和大小。	对数据库集群中的所有数据库实例使用相同的实例类和大小。	是	使用 Amazon Aurora 进行复制

类型	描述	建议	需要停机	其他信息
集群中具有异构参数组的数据库实例	我们建议数据库集群中的所有数据库实例使用同一数据库参数组。	将数据库实例和与数据库集群中的写入器实例关联的数据库参数组相关联。	否	Amazon Aurora 的参数组
Amazon RDS 数据库集群有一个数据库实例	向数据库集群再添加至少一个数据库实例，以提高可用性和性能。	将读取器数据库实例添加到数据库集群。	否	Amazon Aurora 的高可用性
性能详情已关闭	性能详情用于监控数据库实例负载，有助于您分析和解决数据库性能问题。建议您开启性能详情。	开启性能详情。	否	在 Amazon Aurora 上使用性能详情监控数据库负载
需要更新 RDS 资源主要版本	不支持使用数据库引擎当前主要版本的数据库。建议您升级到包含新功能和增强功能的最新主要版本。	升级至数据库引擎的最新主要版本。	是	Amazon Aurora 更新 在 Amazon Aurora 中创建蓝绿部署
数据库集群只支持最高 64TiB 的卷	您的数据库集群支持最高 64TiB 的卷。最新引擎版本对于您的数据库集群支持高达 128TiB 的卷。建议您将数据库集群的引擎版本升级到最新版本，以支持高达 128TiB 的卷。	升级数据库集群的引擎版本以支持高达 128TiB 的卷。	是	Amazon Aurora 大小限制

类型	描述	建议	需要停机	其他信息
数据库集群的所有读取器实例都在同一可用区中	在每个 AWS 区域内，可用区（AZ）是相互不同的位置，以便在发生中断时提供隔离。建议您将数据库集群中的主实例和读取器实例分配到多个可用区，以提高数据库集群的可用性。在创建集群时，您可以使用 AWS 管理控制台、AWS CLI 或 Amazon RDS API 创建多可用区集群。您可以通过添加新的读取器实例并指定不同的可用区，修改现有 Aurora 集群以转为多可用区集群。	您的数据库集群的所有读取实例都在同一可用区中。建议您将读取器实例分布在多个可用区中。这一分布可提高可用性，并通过减少客户端与数据库之间的网络延迟来缩短响应时间。	否	Amazon Aurora 的高可用性
数据库内存参数与默认值不同	数据库实例的内存参数与默认值明显不同。这些设置可能会影响性能并导致错误。 建议您将数据库实例的自定义内存参数重置为数据库参数组中的默认值。	将内存参数重置为其默认值。	否	Amazon Aurora 的参数组

类型	描述	建议	需要停机	其他信息
查询缓存参数已开启	当更改要求清除查询缓存时，您的数据库实例将显示为停滞状态。大多数工作负载不会受益于查询缓存。查询缓存已从 MySQL 8.0 及更高版本中删除。建议您将 query_cache_type 参数设置为 0。	在数据库参数组中将 query_cache_type 参数值设置为 0。	是	Amazon Aurora 的参数组
log_output 参数设置为表	如果 log_output 设置为 TABLE，则使用的存储比 log_output 设置为 FILE 时更多。建议您将此参数设置为 FILE，以避免达到存储大小限制。在 MySQL 8.4 及更高版本中默认设置为 FILE。	在数据库参数组中将 log_output 参数值设置为 FILE。	否	Aurora MySQL 数据库日志文件
autovacuum 参数已关闭	数据库集群 的 autovacuum 参数已关闭. 关闭 autovacuum 会增加表和索引膨胀并影响性能。 建议您在数据库参数组中开启 autovacuum。	在数据库集群参数组中开启 autovacuum 参数。	否	AWS 数据库博客上的 Understanding autovacuum in Amazon RDS for PostgreSQL environments

类型	描述	建议	需要停机	其他信息
synchronous_commit 参数已关闭	关闭 synchronous_commit 参数后，数据库崩溃可能会导致数据丢失。数据库的持久性受到威胁。 建议您开启此 synchronous_commit 参数。	在数据库参数组中开启 synchronous_commit 参数。	是	AWS 数据库博客上的 Amazon Aurora PostgreSQL parameters: Replication, security, and logging。
track_counts 参数已关闭	当 track_counts 参数处于关闭状态时，数据库不会收集数据库活动统计数据。Autovacuum 需要这些统计信息才能正常工作。 建议您将 track_counts 参数设置为 1。	将 track_counts 数据设置为 1。	否	PostgreSQL 的运行时统计数据
enable_indexonlyscan 参数已关闭	关闭仅限索引的扫描计划类型时，查询计划程序或优化程序无法使用该类型。 建议您将 enable_indexonlyscan 参数值设置为 1。	将 enable_indexonlyscan 参数值设置为 1。	否	PostgreSQL 的计划程序方法配置

类型	描述	建议	需要停机	其他信息
enable_indexscan 参数已关闭	关闭索引扫描计划类型时，查询计划程序或优化程序无法使用该类型。 建议您将 enable_indexscan 值设置为 1。	将 enable_indexscan 参数值设置为 1。	否	PostgreSQL 的计划程序方法配置
innodb_flush_log_at_trx 参数已关闭	您的数据库实例的 innodb_flush_log_at_trx 参数值不是安全的值。此参数控制向磁盘提交操作的持久性。 建议您将 innodb_flush_log_at_trx 参数设置为 1。	将 innodb_flush_log_at_trx 参数值设置为 1。	否	配置刷新日志缓冲区的频率
innodb_stats_persistent 参数已关闭	您的数据库实例未配置为将 InnoDB 统计信息持久保存到磁盘上。如果不存储统计数据，则每次实例重启和访问表时都会重新计算统计数据。这会导致查询执行计划的差异。您可以在表级别修改此全局参数的值。 建议您将 innodb_stats_persistent 参数值设置为 ON。	将 innodb_stats_persistent 参数值设置为 ON。	否	Amazon Aurora 的参数组

类型	描述	建议	需要停机	其他信息
innodb_open_files 参数较低	innodb_open_files 参数控制 InnoDB 一次可打开的文件数量。当 mysqld 运行时，InnoDB 会打开所有日志和系统表空间文件。 对于您的数据库实例，InnoDB 一次可打开的最大文件数的值较低。建议您将 innodb_open_files 参数设置为最小值 65。	将 innodb_open_files 参数设置为最小值 65。	是	InnoDB 对于 MySQL 打开的文件
max_user_connections 参数较低	对于您的数据库实例，每个数据库账户的最大同时连接数的值较低。 建议您将 max_user_connections 参数设置为大于 5 的数字。	将 max_user_connections 参数的值增加到大于 5 的数字。	是	设置 MySQL 的账户资源限制

类型	描述	建议	需要停机	其他信息
只读副本在可写模式下打开	您的数据库实例的只读副本处于可写模式，这允许来自客户端的更新。 建议您将 read_only 参数设置为 TrueIfReplica ，这样只读副本就不会处于可写模式。	将 read_only 参数值设置为 TrueIfReplica 。	否	Amazon Aurora 的参数组
innodb_default_row_format 参数设置不安全	您的数据库实例遇到一个已知问题：当索引超过 767 字节时，在低于 8.0.26 的 MySQL 版本中创建且 row_format 设置为 COMPACT 或 REDUNDANT 的表将不可访问且无法恢复。 建议您将 innodb_default_row_format 参数值设置为 DYNAMIC。	将 innodb_default_row_format 参数值设置为 DYNAMIC。	否	MySQL 8.0.26 中的变化

类型	描述	建议	需要停机	其他信息
general_logging 参数已开启	您的数据库实例已开启常规日志记录。在排查数据库问题时，此设置非常有用。但是，开启常规日志记录会增加 I/O 操作量和分配的存储空间量，这可能导致争用和性能降级。 请检查您对常规日志记录使用情况的要求。建议您将 general_logging 参数值设置为 0。	请检查您对常规日志记录使用情况的要求。如果它不是强制性的，建议您将 general_logging 参数值设置为 0。	否	Aurora MySQL 数据库日志概览
数据库集群的读取工作负载预调配不足	建议您向数据库集群添加读取器数据库实例，实例类和大小与集群中的写入器数据库实例相同。当前配置中有一个数据库实例的数据库负载持续很高，这主要是由读取操作造成的。通过向集群添加另一个数据库实例并将读取工作负载定向到数据库集群只读端点来分配这些操作。	向集群添加读取器数据库实例。	否	将 Aurora 副本添加到数据库集群 管理 Aurora 数据库集群的性能和扩展 Amazon RDS 定价

类型	描述	建议	需要停机	其他信息
RDS 实例的系统内存容量预调配不足	建议您调整查询以使用更少的内存，或者使用所分配内存更高的数据库实例类型。当实例内存不足时，数据库性能就会受到影响。	使用内存容量更高的数据库实例	是	AWS 数据库博客上的 Scaling Your Amazon RDS Instance Vertically and Horizontally Amazon RDS 实例类型 Amazon RDS 定价
RDS 实例的系统 CPU 容量预调配不足	建议您调整查询来使用更少的 CPU，或修改数据库实例来使用所分配的 vCPU 更高的数据库实例类。当运行数据库实例的 CPU 容量不足时，数据库性能可能会下降。	使用 CPU 容量更高的数据库实例	是	AWS 数据库博客上的 Scaling Your Amazon RDS Instance Vertically and Horizontally Amazon RDS 实例类型 Amazon RDS 定价
RDS 资源未正确地利用连接池	建议您启用 Amazon RDS 代理，以便高效地池化和共享现有数据库连接。如果您已经在为数据库使用代理，请正确地配置代理，来改善多个数据库实例之间的连接池和负载均衡。RDS 代理有助于降低连接耗尽和停机的风险，同时提高可用性和可扩展性。	启用 RDS 代理或修改现有的代理配置	否	AWS 数据库博客上的 Scaling Your Amazon RDS Instance Vertically and Horizontally 适用于 Aurora 的 Amazon RDS 代理 Amazon RDS 代理定价

在 Amazon RDS 控制台中查看指标

Amazon RDS 与 Amazon CloudWatch 集成以显示 RDS 控制台中的各种 Aurora 数据库集群 指标。有些指标适用于集群级别，而另一些指标适用于实例级别。有关实例级和集群级指标的说明，请参阅 Amazon Aurora 的指标参考。

对于 Aurora 数据库集群，将监控以下类别的指标：

- CloudWatch – 显示您可以在 RDS 控制台中访问的 Aurora 的 Amazon CloudWatch 指标。您也可以在 CloudWatch 控制台中访问这些指标。每个指标均包括一个图形，显示特定时间范围内监控的指标。有关 CloudWatch 指标的列表，请参阅 Amazon Aurora 的 Amazon CloudWatch 指标。
- 增强监控 - 显示 Aurora 数据库集群 打开增强监控时操作系统指标的摘要。RDS 将增强监控中的指标传输到您的 Amazon CloudWatch Logs 账户。每个操作系统指标均包括一个图形，显示特定时间范围内监控的指标。有关概述，请参阅使用增强监控来监控操作系统指标。有关增强监控指标的列表，请参阅 增强监控中的操作系统指标。
- OS 进程列表 – 显示您的数据库集群中运行的每个进程的详细信息。
- Performance Insights - 打开 Aurora 数据库集群中数据库实例的 Amazon RDS Performance Insights 控制面板。集群级别不支持 Performance Insights。有关 Performance Insights 概述，请参阅 在 Amazon Aurora 上使用性能详情监控数据库负载。有关 Performance Insights 指标的列表，请参阅 Amazon RDS 性能详情的 Amazon CloudWatch 指标。

Amazon RDS 现在在性能详情控制面板中提供性能详情和 CloudWatch 指标的合并视图。必须为您的数据库集群启用性能详情，才能使用此视图。您可以在监控选项卡中选择新的监控视图，也可以在导航窗格中选择性能详情。要查看有关选择此视图的说明，请参阅使用性能详情控制面板查看组合指标。

使用性能详情控制面板查看组合指标

> **Important**
>
> AWS 已宣布性能详情的生命周期终止日期为：2025 年 11 月 30 日。在此日期之后，Amazon RDS 将不再支持性能详情控制台体验、灵活的保留期（1-24 个月）及其关联定价。性能详情 API 将继续存在，定价不变。性能详情 API 的费用将与 CloudWatch 数据库洞察的费用一起显示在您的 AWS 账单中。
>
> 我们建议您在 2025 年 11 月 30 日之前，将使用性能详情付费套餐的所有数据库集群升级到数据库洞察的高级模式。有关升级到数据库洞察的高级模式的信息，请参阅开启适用于 Amazon Aurora 的数据库洞察的高级模式。
>
> 如果您不采取任何行动，则使用性能详情的数据库集群将默认为使用数据库洞察的标准模式。在数据库洞察的标准模式下，您可能会在 7 天后无法访问性能数据历史记录，并且可能无法在 Amazon RDS 控制台中使用执行计划和按需分析功能。在 2025 年 11 月 30 日之后，只有数据库洞察的高级模式将支持执行计划和按需分析。
>
> 借助 CloudWatch 数据库洞察，您可以监控数据库实例集的数据库负载，并对性能进行大规模分析和故障排除。有关数据库洞察的更多信息，请参阅使用 CloudWatch 数据库洞察监控 Amazon Aurora 数据库。有关定价信息，请参阅 Amazon CloudWatch 定价。

Amazon RDS 现在在性能详情控制面板中提供数据库实例的性能详情和 CloudWatch 指标的合并视图。您可以使用预配置的控制面板或创建自定义控制面板。预配置的控制面板提供最常用的指标，以帮助诊断数据库引擎的性能问题。或者，您可以创建一个自定义控制面板，其中包含数据库引擎的可满足分析需求的指标。然后，将此控制面板用于您的 AWS 账户中该数据库引擎类型的所有数据库实例。

您可以在监控选项卡中选择监控视图，也可以在导航窗格中选择性能详情。

必须为数据库集群开启性能详情，才能在性能详情控制面板中查看组合指标。有关开启性能详情的更多信息，请参阅为 Aurora 打开和关闭性能详情。

在以下各节中，您可以了解使用新版和旧版监控视图来显示性能详情和 CloudWatch 指标。

主题

- 从“监控”选项卡中选择新版监控视图
- 从性能详情页面中选择新版监控视图
- 使用性能详情创建自定义控制面板
- 使用性能详情选择预配置控制面板

从“监控”选项卡中选择新版监控视图

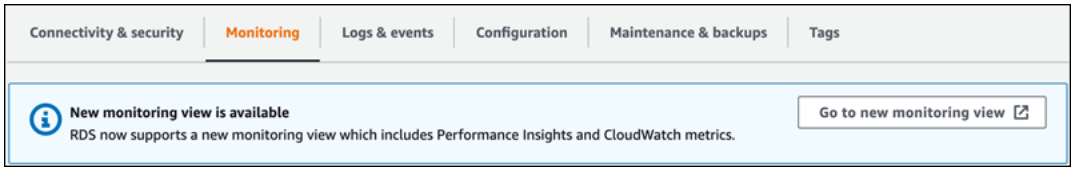
> **Important**
>
> AWS 已宣布性能详情的生命周期终止日期为：2025 年 11 月 30 日。在此日期之后，Amazon RDS 将不再支持性能详情控制台体验、灵活的保留期（1-24 个月）及其关联定价。性能详情 API 将继续存在，定价不变。性能详情 API 的费用将与 CloudWatch 数据库洞察的费用一起显示在您的 AWS 账单中。
>
> 我们建议您在 2025 年 11 月 30 日之前，将使用性能详情付费套餐的所有数据库集群升级到数据库洞察的高级模式。有关升级到数据库洞察的高级模式的信息，请参阅开启适用于 Amazon Aurora 的数据库洞察的高级模式。
>
> 如果您不采取任何行动，则使用性能详情的数据库集群将默认为使用数据库洞察的标准模式。在数据库洞察的标准模式下，您可能会在 7 天后无法访问性能数据历史记录，并且可能无法在 Amazon RDS 控制台中使用执行计划和按需分析功能。在 2025 年 11 月 30 日之后，只有数据库洞察的高级模式将支持执行计划和按需分析。
>
> 借助 CloudWatch 数据库洞察，您可以监控数据库实例集的数据库负载，并对性能进行大规模分析和故障排除。有关数据库洞察的更多信息，请参阅使用 CloudWatch 数据库洞察监控 Amazon Aurora 数据库。有关定价信息，请参阅 Amazon CloudWatch 定价。

从 Amazon RDS 控制台中，您可以选择新版监控视图来查看数据库实例的性能详情和 CloudWatch 指标。

在“监控”选项卡中选择新版监控视图

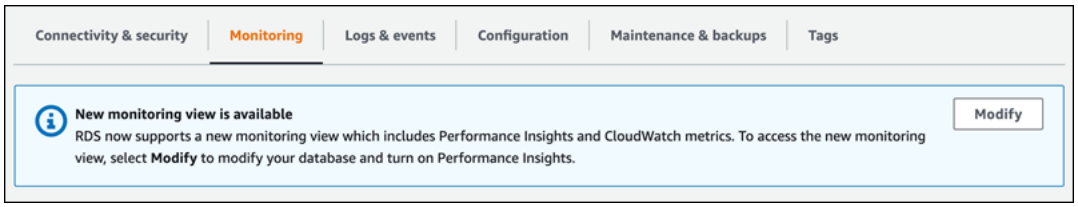
1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在左侧导航窗格中，选择数据库。
3. 选择要监控的 Aurora 数据库集群的名称。
4. 向下滚动并选择监控选项卡。

   此时会出现一条横幅，其中包含选择新监控视图的选项。以下示例显示了选择新监控视图的横幅。



5. 选择转到新的监控视图以打开性能详情控制面板，其中包含数据库集群的性能详情和 CloudWatch 指标。
6. （可选）如果对数据库实例关闭了性能详情，则会出现一个横幅，其中包含修改数据库实例和开启性能详情的选项。

   以下示例显示了在监控选项卡中修改数据库实例的横幅。



   选择修改以修改数据库实例并开启性能详情。有关开启性能详情的更多信息，请参阅[为 Aurora 打开和关闭性能详情]

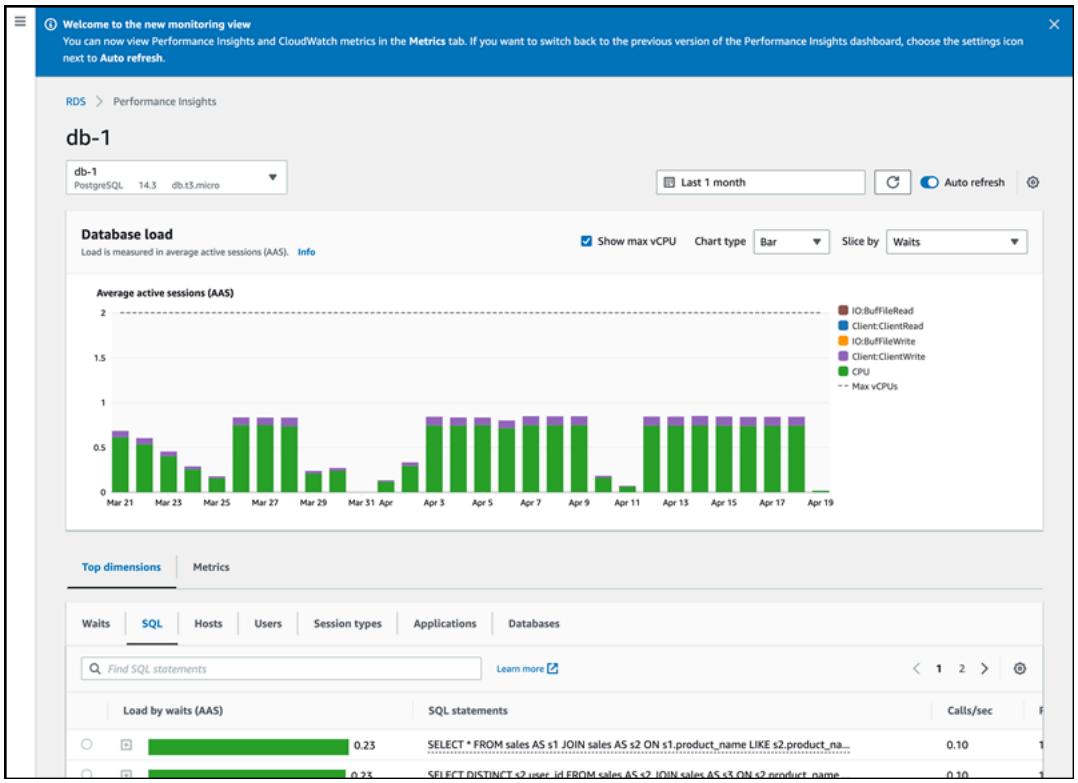
从性能详情页面中选择新版监控视图

> **Important**
>
> AWS 已宣布性能详情的生命周期终止日期为：2025 年 11 月 30 日。在此日期之后，Amazon RDS 将不再支持性能详情控制台体验、灵活的保留期（1-24 个月）及其关联定价。性能详情 API 将继续存在，定价不变。性能详情 API 的费用将与 CloudWatch 数据库洞察的费用一起显示在您的 AWS 账单中。
> 我们建议您在 2025 年 11 月 30 日之前，将使用性能详情付费套餐的所有数据库集群升级到数据库洞察的高级模式。有关升级到数据库洞察的高级模式的信息，请参阅[开启适用于 Amazon Aurora 的数据库洞察的高级模式]。
> 如果您不采取任何行动，则使用性能详情的数据库集群将默认为使用数据库洞察的标准模式。在数据库洞察的标准模式下，您可能会在 7 天后无法访问性能数据历史记录，并且可能无法在 Amazon RDS 控制台中使用执行计划和按需分析功能。在 2025 年 11 月 30 日之后，只有数据库洞察的高级模式将支持执行计划和按需分析。
> 借助 CloudWatch 数据库洞察，您可以监控数据库实例集的数据库负载，并对性能进行大规模分析和故障排除。有关数据库洞察的更多信息，请参阅[使用 CloudWatch 数据库洞察监控 Amazon Aurora 数据库]。有关定价信息，请参阅 [Amazon CloudWatch 定价]。

从 Amazon RDS 控制台中，您可以选择新版监控视图来查看数据库实例的性能详情和 CloudWatch 指标。

使用导航窗格中的性能详情选择新版监控视图

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在左侧导航窗格中，选择性能详情。
3. 选择一个数据库实例，以便查看性能详情控制面板，其中显示数据库实例的性能详情和 CloudWatch 指标。



使用性能详情创建自定义控制面板

> **Important**
>
> AWS 已宣布性能详情的生命周期终止日期为：2025 年 11 月 30 日。在此日期之后，Amazon RDS 将不再支持性能详情控制台体验、灵活的保留期（1-24 个月）及其关联定价。性能详情 API 将继续存在，定价不变。性能详情 API 的费用将与 CloudWatch 数据库洞察的费用一起显示在您的 AWS 账单中。
>
> 我们建议您在 2025 年 11 月 30 日之前，将使用性能详情付费套餐的所有数据库集群升级到数据库洞察的高级模式。有关升级到数据库洞察的高级模式的信息，请参阅开启适用于 Amazon Aurora 的数据库洞察的高级模式。

如果您不采取任何行动，则使用性能详情的数据库集群将默认为使用数据库洞察的标准模式。在数据库洞察的标准模式下，您可能会在 7 天后无法访问性能数据历史记录，并且可能无法在 Amazon RDS 控制台中使用执行计划和按需分析功能。在 2025 年 11 月 30 日之后，只有数据库洞察的高级模式将支持执行计划和按需分析。
借助 CloudWatch 数据库洞察，您可以监控数据库实例集的数据库负载，并对性能进行大规模分析和故障排除。有关数据库洞察的更多信息，请参阅使用 CloudWatch 数据库洞察监控 Amazon Aurora 数据库。有关定价信息，请参阅 Amazon CloudWatch 定价。

在新的监控视图中，您可以创建自定义控制面板，其中具有满足分析要求所需的指标。

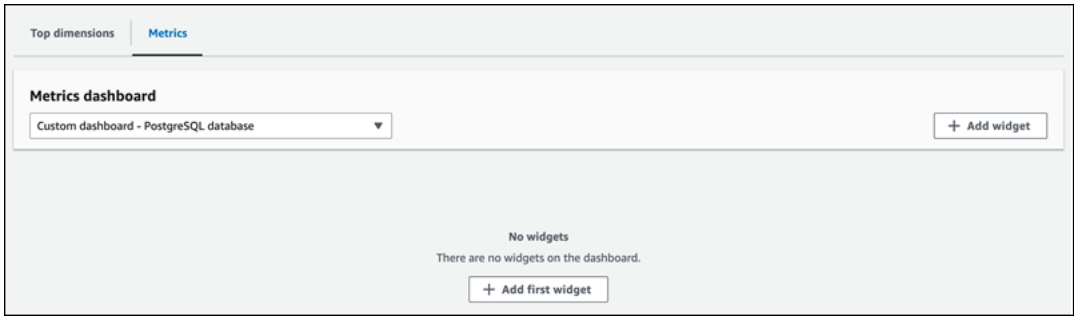
您可以通过为数据库实例选择性能详情和 CloudWatch 指标来创建自定义控制面板。您可以将此自定义控制面板用于 AWS 账户中属于相同数据库引擎类型的其他数据库实例。

> **Note**
>
> 自定义控制面板可支持多达 50 个指标。

使用小组件设置菜单编辑或删除控制面板，并移动小组件窗口或调整其大小。

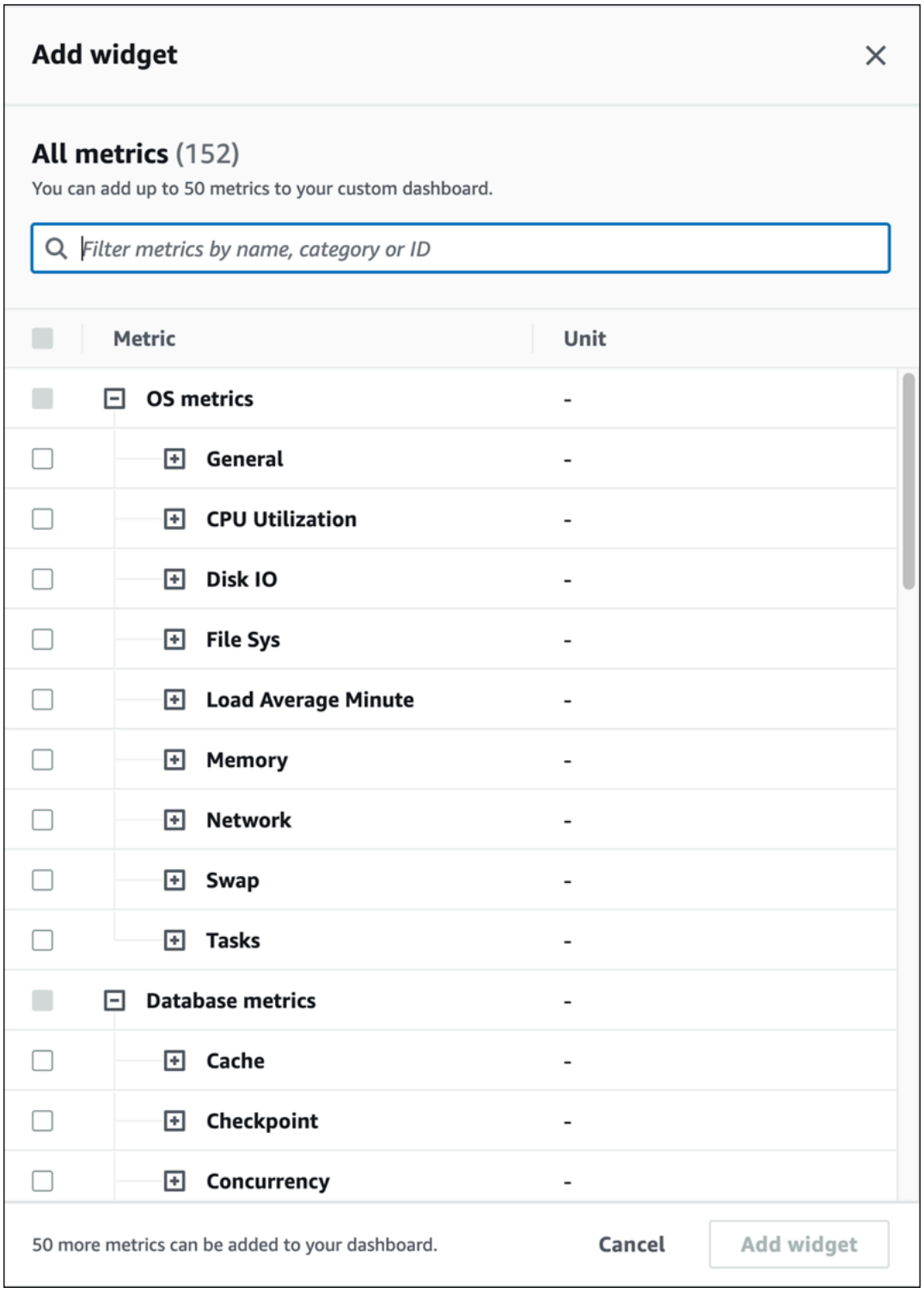
使用导航窗格中的性能详情创建自定义控制面板

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在左侧导航窗格中，选择性能详情。
3. 选择一个数据库实例。
4. 向下滚动到窗口中的指标选项卡。
5. 从下拉列表中选择自定义控制面板。以下显示了自定义控制面板的创建过程。



6. 选择添加小组件以打开添加小组件窗口。您可以在此窗口中打开和查看可用的操作系统（OS）指标、数据库指标和 CloudWatch 指标。

   以下示例显示了添加小组件窗口以及指标。



7. 选择要在控制面板中查看的指标，然后选择添加小组件。您可以使用搜索字段来查找特定的指标。

所选指标显示在您的控制面板上。

8. （可选）如果您要修改或删除控制面板，请选择小组件右上角的设置图标，然后在菜单中选择以下操作之一。

   - 编辑 - 修改窗口中的指标列表。为控制面板选择指标后，选择更新小组件。
   - 删除 - 删除小组件。在确认窗口中选择删除。

使用性能详情选择预配置控制面板

> **Important**
>
> AWS 已宣布性能详情的生命周期终止日期为：2025 年 11 月 30 日。在此日期之后，Amazon RDS 将不再支持性能详情控制台体验、灵活的保留期（1-24 个月）及其关联定价。性能详情 API 将继续存在，定价不变。性能详情 API 的费用将与 CloudWatch 数据库洞察的费用一起显示在您的 AWS 账单中。
>
> 我们建议您在 2025 年 11 月 30 日之前，将使用性能详情付费套餐的所有数据库集群升级到数据库洞察的高级模式。有关升级到数据库洞察的高级模式的信息，请参阅开启适用于 Amazon Aurora 的数据库洞察的高级模式。
>
> 如果您不采取任何行动，则使用性能详情的数据库集群将默认为使用数据库洞察的标准模式。在数据库洞察的标准模式下，您可能会在 7 天后无法访问性能数据历史记录，并且可能无法在 Amazon RDS 控制台中使用执行计划和按需分析功能。在 2025 年 11 月 30 日之后，只有数据库洞察的高级模式将支持执行计划和按需分析。
>
> 借助 CloudWatch 数据库洞察，您可以监控数据库实例集的数据库负载，并对性能进行大规模分析和故障排除。有关数据库洞察的更多信息，请参阅使用 CloudWatch 数据库洞察监控 Amazon Aurora 数据库。有关定价信息，请参阅 Amazon CloudWatch 定价。

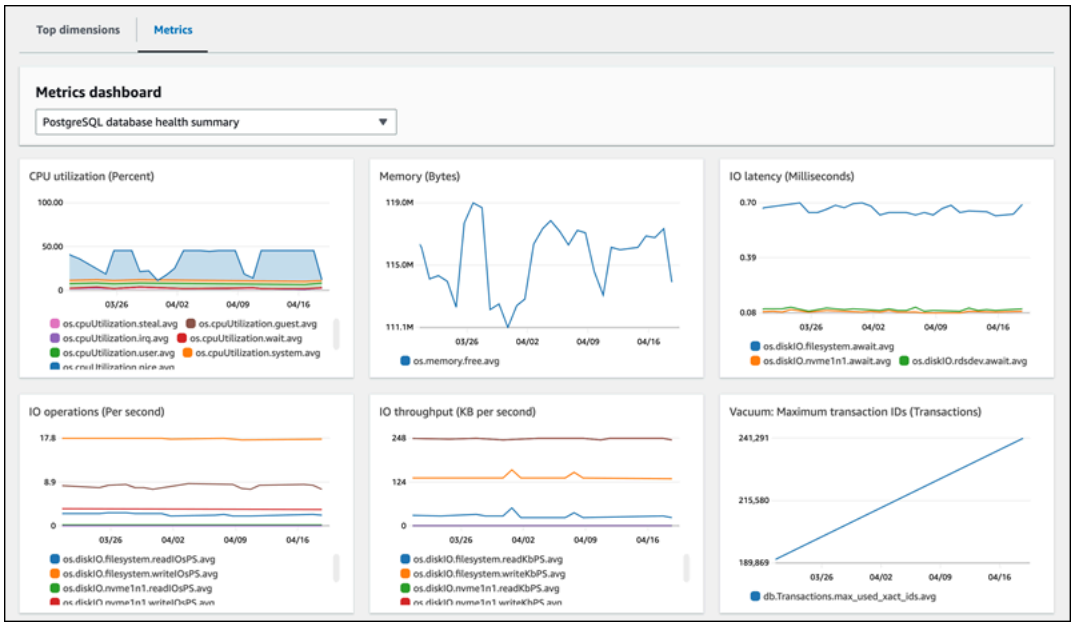
您可以使用预配置控制面板查看最常用的指标。此控制面板有助于诊断数据库引擎的性能问题，并将平均恢复时间从几小时缩短到几分钟。

> **Note**
>
> 无法编辑此控制面板。

使用导航窗格中的性能详情选择预配置控制面板

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在左侧导航窗格中，选择性能详情。
3. 选择一个数据库实例。
4. 向下滚动到窗口中的指标选项卡
5. 从下拉列表中选择预配置的控制面板。

   您可以在控制面板中查看数据库实例的指标。以下示例显示了一个预配置指标控制面板。



使用 Amazon CloudWatch 监控 Amazon Aurora 指标

Amazon CloudWatch 是一个指标存储库。此存储库可从 Amazon Aurora 收集原始数据，并将数据处理为易读的近乎实时的指标。有关发送到 CloudWatch 的 Amazon Aurora 指标的完整列表，请参阅 Amazon Aurora 的指标参考。

要大规模分析数据库性能并对其进行故障排除，请使用 CloudWatch 数据库洞察。

主题

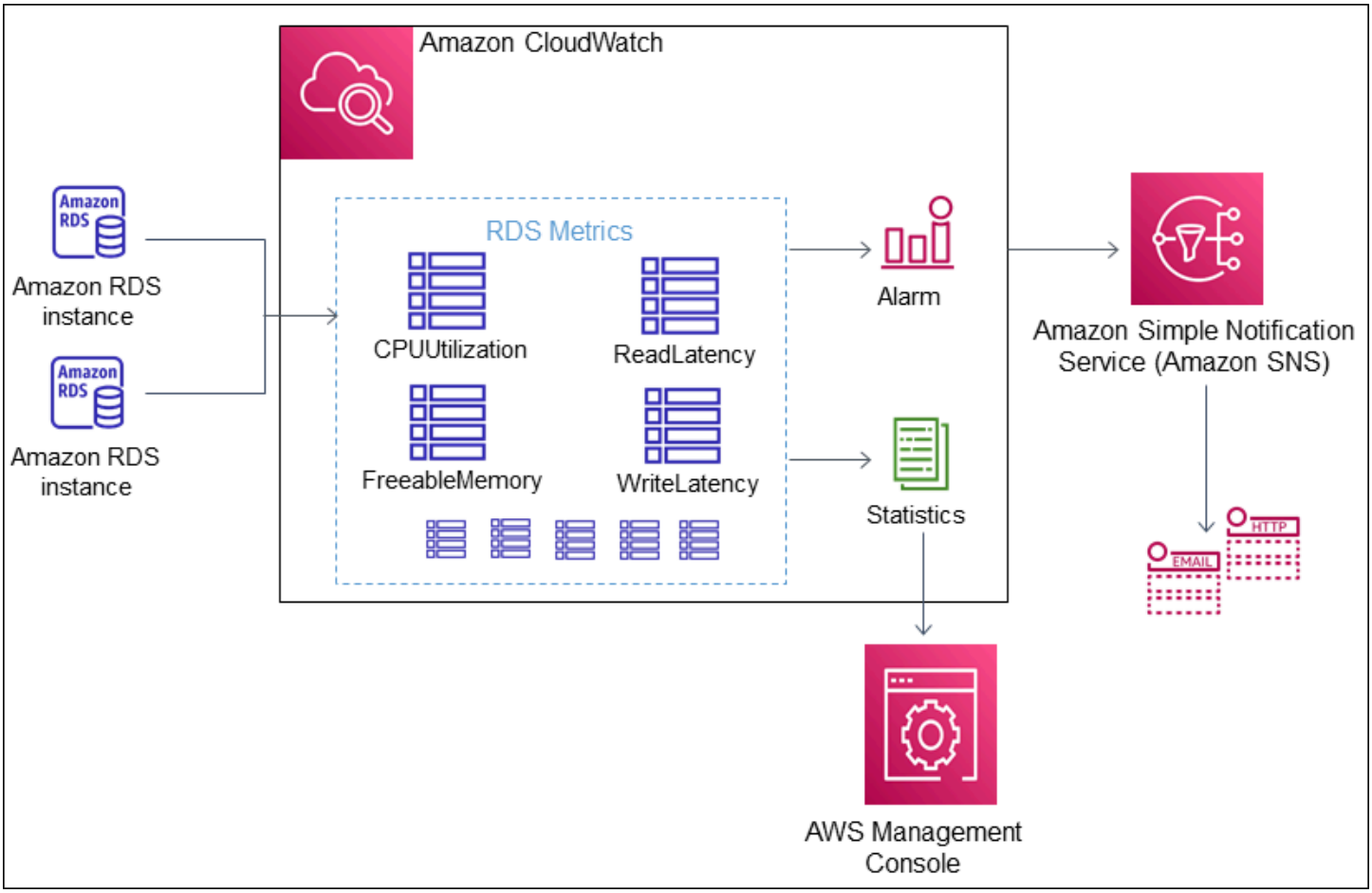
- Amazon Aurora 和 Amazon CloudWatch 概述
- 在 CloudWatch 控制台和 AWS CLI 中查看数据库集群指标
- 将 Performance Insights 指标导出到 CloudWatch
- 创建 CloudWatch 警报以监控 Amazon Aurora

Amazon Aurora 和 Amazon CloudWatch 概述

默认情况下，以 1 分钟为间隔自动将 Amazon Aurora 指标数据发送到 CloudWatch。例如，`CPUUtilization` 指标记录了一段时间内数据库实例的 CPU 使用率百分比。时间段为 60 秒（1 分钟）的数据点可用 15 天。这意味着您能够访问历史信息，了解您的 Web 应用程序或服务的执行情况。

现在，您可以将 Performance Insights 指标控制面板从 Amazon RDS 导出到 Amazon CloudWatch。您可以将预配置或自定义的指标控制面板导出为新的控制面板，也可以将其添加到现有的 CloudWatch 控制面板中。导出的控制面板可在 CloudWatch 控制台中查看。有关如何将 Performance Insights 指标控制面板导出到 CloudWatch 的更多信息，请参阅将 Performance Insights 指标导出到 CloudWatch。

如下图所示，您可以为 CloudWatch 指标设置警报。例如，您可能会创建一个警报，在实例的 CPU 使用率超过 70% 时发出信号。您可以配置 Amazon Simple Notification Service 以在超过阈值时向您发送电子邮件。



Amazon RDS 向 Amazon CloudWatch 发布以下类型的指标：

- 集群和实例级别的 Aurora 指标

  有关这些指标的表格，请参阅 Amazon Aurora 的 Amazon CloudWatch 指标。
- Performance Insights 指标

  有关这些指标的表格，请参阅 Amazon RDS 性能详情的 Amazon CloudWatch 指标 和 性能详情计数器指标。
- 增强监控指标（发布到 Amazon CloudWatch Logs）

  有关这些指标的表格，请参阅 增强监控中的操作系统指标。
- 您的 AWS 账户中 Amazon RDS 服务配额的用量指标

  有关这些指标的表格，请参阅 Amazon Aurora 的 Amazon CloudWatch 用量指标。有关 Amazon RDS 配额的更多信息，请参阅 Amazon Aurora 的配额和限制。

有关 CloudWatch 的更多信息，请参阅 Amazon CloudWatch 用户指南 中的什么是 Amazon CloudWatch？。有关 CloudWatch 指标保留的更多信息，请参阅指标保留。

在 CloudWatch 控制台和 AWS CLI 中查看数据库集群指标

在下文中，您可以了解有关如何使用 CloudWatch 查看数据库实例指标的详细信息。有关使用 CloudWatch Logs 实时监控数据库实例的操作系统指标的信息，请参阅使用增强监控来监控操作系统指标。

在使用 Amazon Aurora 资源时，Amazon Aurora 每分钟向 Amazon CloudWatch 发送一次指标和维度。

现在，您可以将 Performance Insights 指标控制面板从 Amazon RDS 导出到 Amazon CloudWatch，然后在 CloudWatch 控制台中查看这些指标。有关如何将 Performance Insights 指标控制面板导出到 CloudWatch 的更多信息，请参阅将 Performance Insights 指标导出到 CloudWatch。

使用以下过程在 CloudWatch 控制台和 CLI 中查看 Amazon Aurora 的指标。

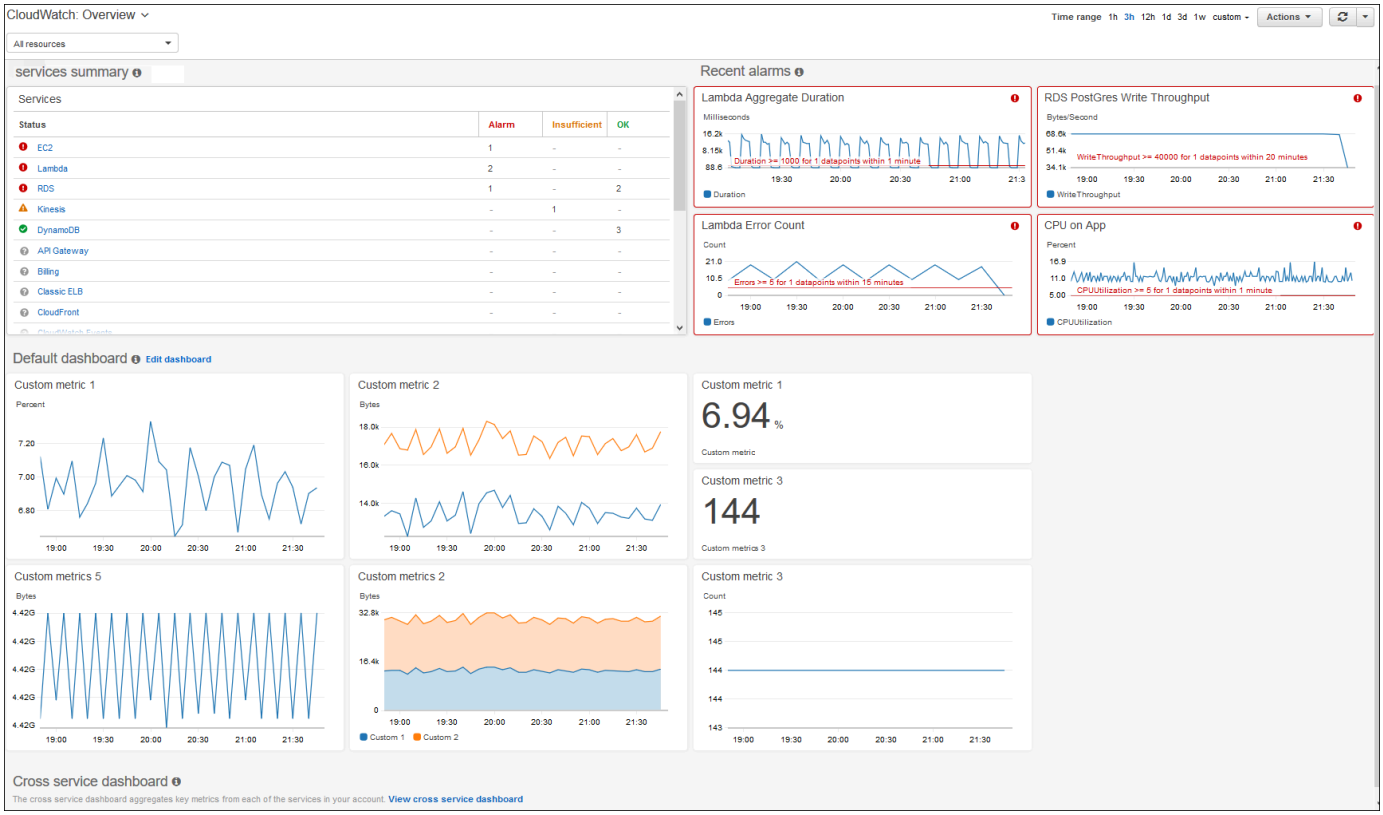
控制台

使用 Amazon CloudWatch 控制台查看指标

指标的分组首先依据服务命名空间，然后依据每个命名空间内的各种维度组合。

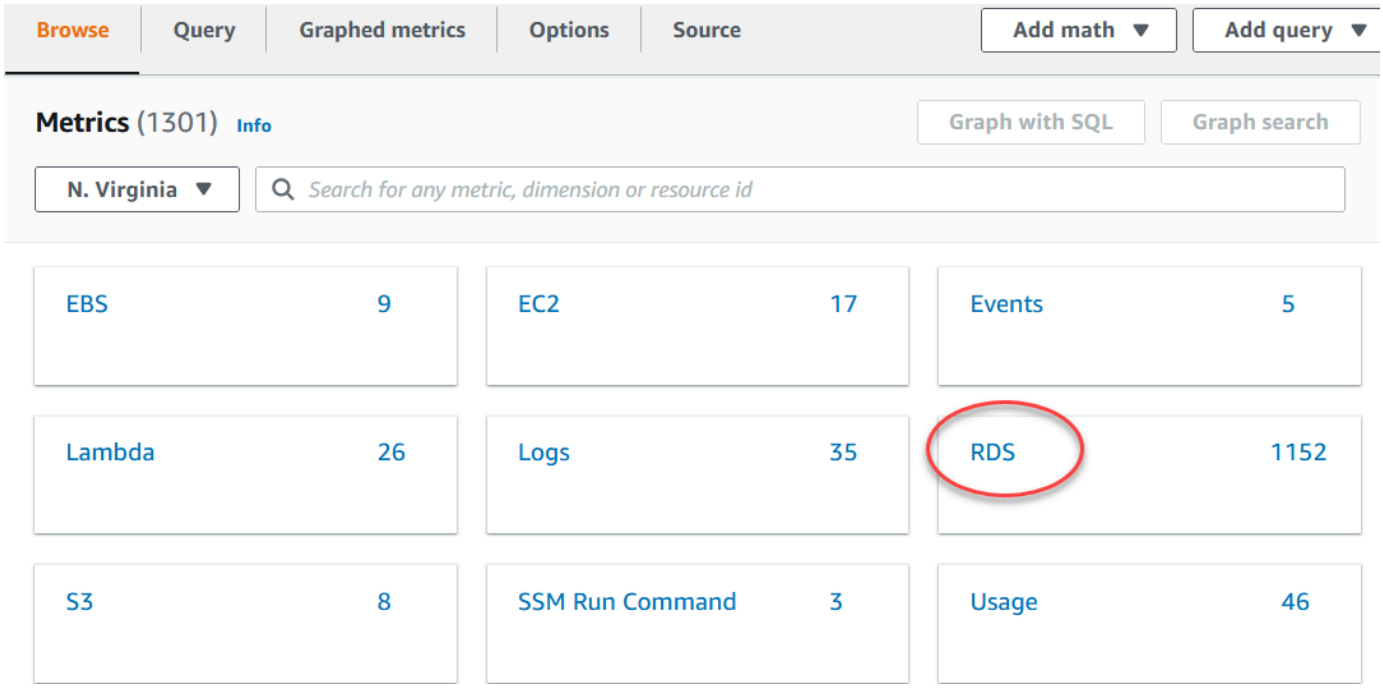
1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。

将显示 CloudWatch 概览主页。



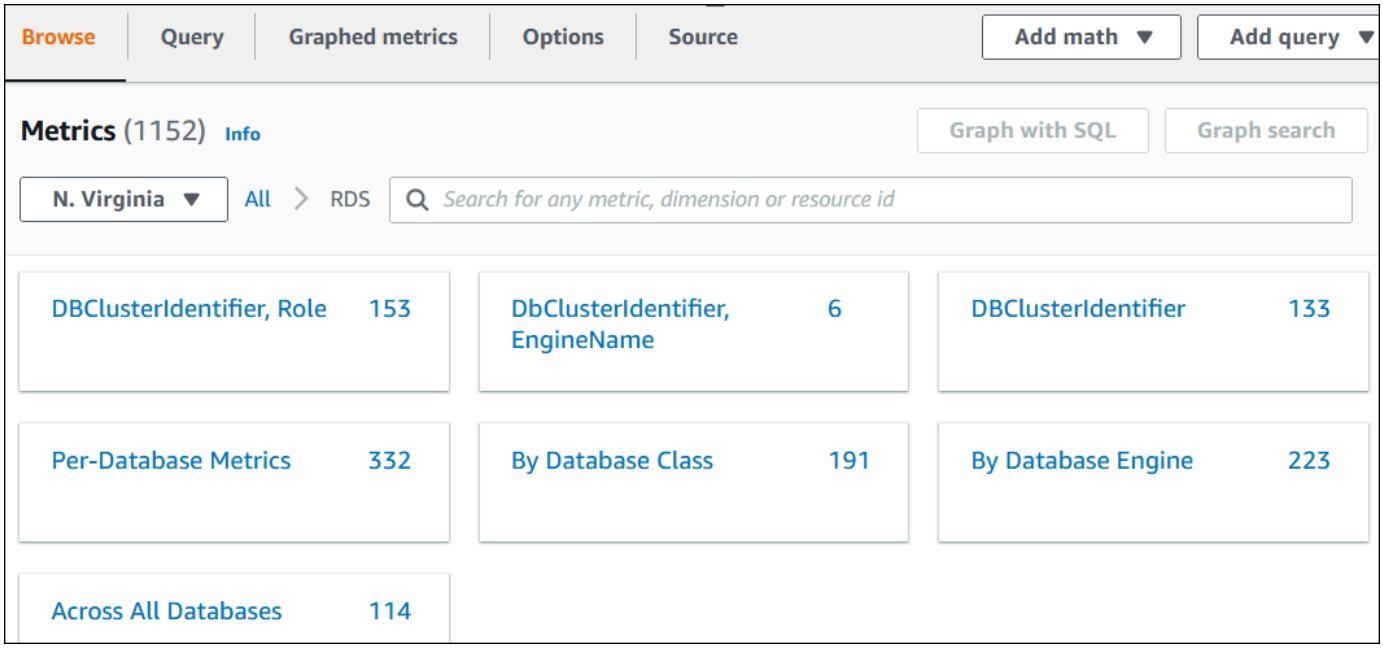
2. 如果需要，更改 AWS 区域。从导航栏中，选择您的 AWS 资源所在的 AWS 区域。有关更多信息，请参阅区域和端点。

3. 在导航窗格中，选择 Metrics（指标），然后选择 All metrics（所有指标）。

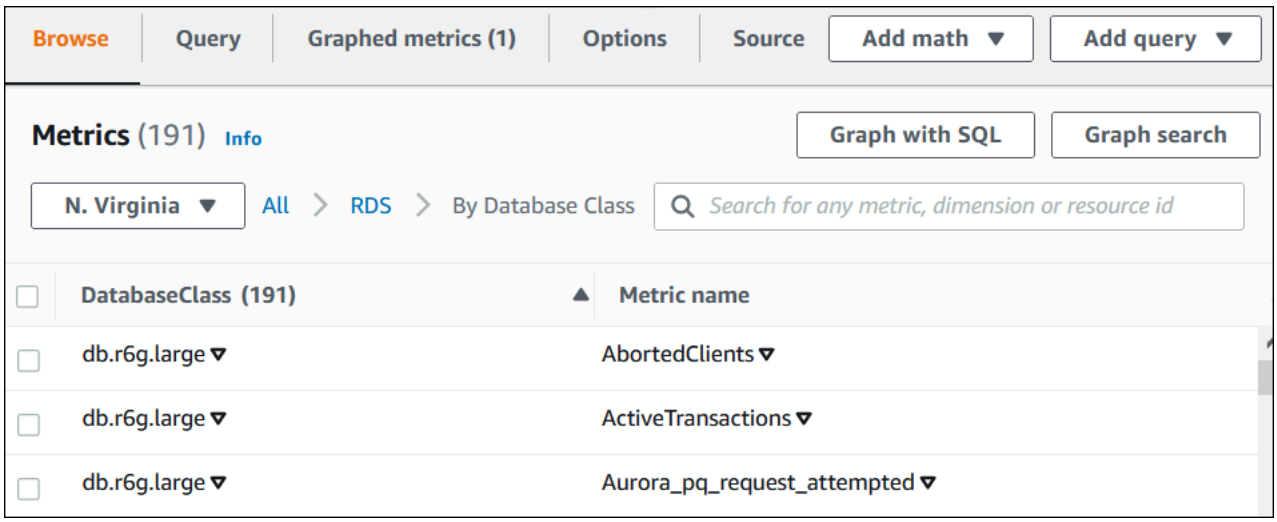


4. 向下滚动并选择 RDS 指标命名空间。

   该页面将显示 Amazon Aurora 维度。有关这些维度的说明，请参阅 Aurora 的 Amazon CloudWatch 维度。



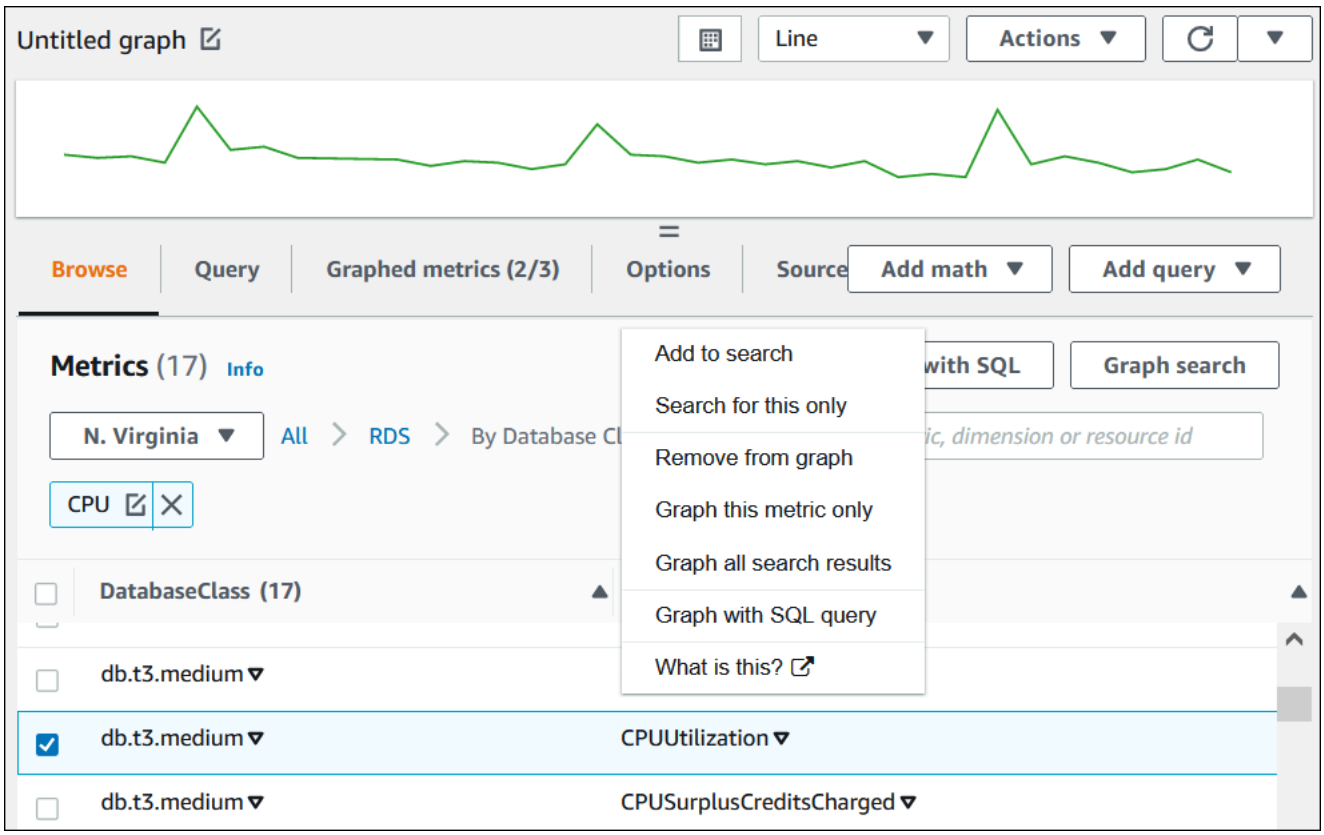
5. 选择指标维度，例如，按数据库类。



6. 执行以下任一操作：

   - 要对指标进行排序，请使用列标题。
   - 要为指标绘制图表，请选中该指标旁的复选框。
   - 要按资源进行筛选，请选择资源 ID，然后选择添加到搜索。
   - 要按指标进行筛选，请选择指标名称，然后选择添加到搜索。

   以下示例筛选 db.t3.medium 类并绘制 CPUUtilization 指标。



您可以了解有关如何使用 CloudWatch 指标分析 Aurora PostgreSQL 资源使用情况的详细信息。有关更多信息，请参阅 使用 Amazon CloudWatch 指标分析 Aurora PostgreSQL 的资源使用情况

AWS CLI

若要通过使用 AWS CLI 获取指标信息，请使用 CloudWatch 命令 `list-metrics`。在以下示例中，您将列出 AWS/RDS 命名空间中的所有指标。

```
aws cloudwatch list-metrics --namespace AWS/RDS
```

要获取指标数据，请使用命令 `get-metric-data`。

以下示例以 5 分钟为间隔，获取实例 `my-instance` 在特定 24 小时时段内的 `CPUUtilization` 统计数据。

使用以下内容创建 JSON 文件 `CPU_metric.json`。

```
{
   "StartTime" : "2023-12-25T00:00:00Z",
```

```
   "EndTime" : "2023-12-26T00:00:00Z",
   "MetricDataQueries" : [{
     "Id" : "cpu",
     "MetricStat" : {
    "Metric" : {
        "Namespace" : "AWS/RDS",
        "MetricName" : "CPUUtilization",
        "Dimensions" : [{ "Name" : "DBInstanceIdentifier" , "Value" : my-instance}]
    },
       "Period" : 360,
       "Stat" : "Minimum"
     }
   }]
}
```

Example

对于 Linux、macOS 或 Unix：

```
aws cloudwatch get-metric-data \
    --cli-input-json file://CPU_metric.json
```

对于 Windows：

```
aws cloudwatch get-metric-data ^
     --cli-input-json file://CPU_metric.json
```

示例输出如下：

```
{
    "MetricDataResults": [
        {
            "Id": "cpu",
            "Label": "CPUUtilization",
            "Timestamps": [
                "2023-12-15T23:48:00+00:00",
                "2023-12-15T23:42:00+00:00",
                "2023-12-15T23:30:00+00:00",
                "2023-12-15T23:24:00+00:00",
                ...
            ],
            "Values": [
```

```
                13.299778337027714,
                13.677507543049558,
                14.24976250395827,
                13.02521708695145,
                ...
            ],
            "StatusCode": "Complete"
        }
    ],
    "Messages": []
}
```

有关更多信息，请参阅《Amazon CloudWatch 用户指南》中的获取指标统计数据。

将 Performance Insights 指标导出到 CloudWatch

Performance Insights 允许您将数据库实例的预配置或自定义指标控制面板导出到 Amazon CloudWatch。您可以将指标控制面板导出为新的控制面板，也可以将其添加到现有的 CloudWatch 控制面板中。当您选择将控制面板添加到现有 CloudWatch 控制面板时，可以创建标题标签，以便指标单独显示在 CloudWatch 控制面板的某个部分中。

您还可以在 CloudWatch 控制台中查看导出的指标控制面板。如果您在导出 Performance Insights 指标控制面板后向其添加了新指标，则必须再次导出此控制面板，然后才能在 CloudWatch 控制台中查看新指标。

您还可以在 Performance Insights 控制面板中选择一个指标小组件，然后在 CloudWatch 控制台中查看指标数据。

有关在 CloudWatch 控制台中查看指标的更多信息，请参阅 在 CloudWatch 控制台和 AWS CLI 中查看数据库集群指标。

在以下各节中，将性能详情指标导出到 CloudWatch 作为新的或现有控制面板，并在 CloudWatch 中查看性能详情指标。

主题

- 将 Performance Insights 指标作为新的控制面板导出到 CloudWatch
- 向现有的 CloudWatch 控制面板添加 Performance Insights 指标
- 在 CloudWatch 中查看 Performance Insights 指标小组件

将 Performance Insights 指标作为新的控制面板导出到 CloudWatch

从 Performance Insights 控制面板中选择预配置或自定义的指标控制面板，然后将其作为新控制面板导出到 CloudWatch。您可以在 CloudWatch 控制台中查看这些导出的控制面板。

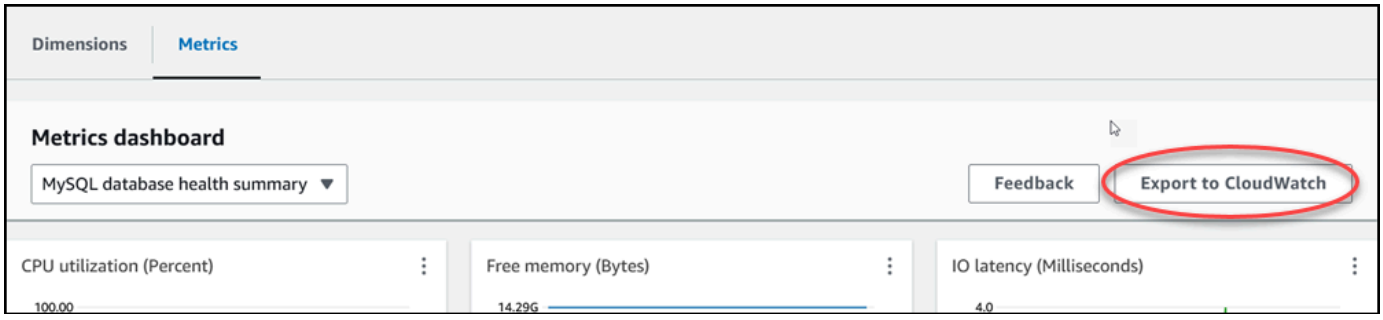
将 Performance Insights 指标控制面板作为新控制面板导出到 CloudWatch

1. 通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在左侧导航窗格中，选择 Performance Insights。
3. 选择一个数据库实例。

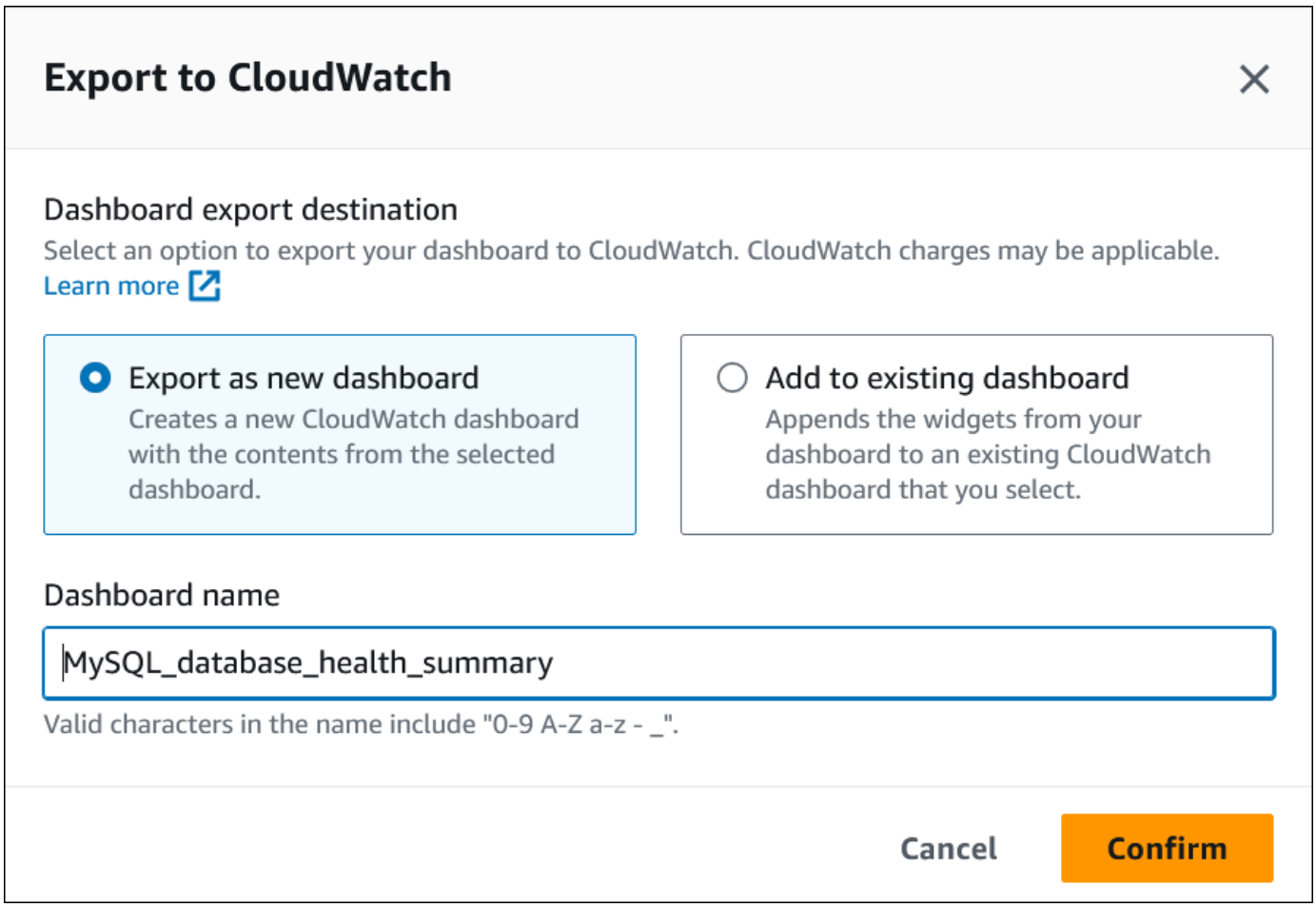
   将显示该数据库实例的 Performance Insights 控制面板。
4. 向下滚动并选择指标。

   默认情况下，会显示包含 Performance Insights 指标的预配置控制面板。
5. 选择预配置或自定义控制面板，然后选择导出到 CloudWatch。

   随后会显示导出到 CloudWatch 窗口。

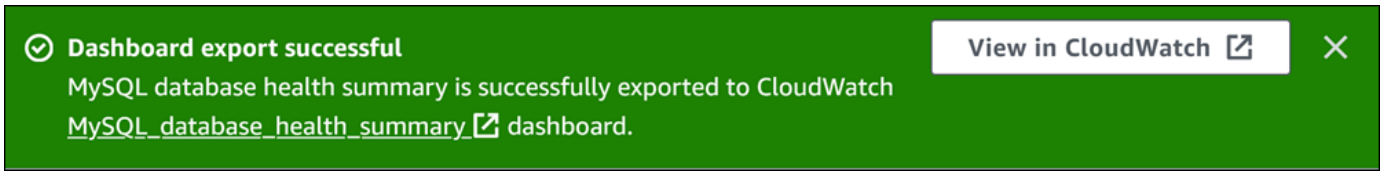


6. 选择导出为新控制面板。



7. 在控制板名称字段中输入新控制面板的名称，然后选择确认。

   控制面板导出成功后，横幅会显示一条消息。



8. 选择横幅中的链接或在 CloudWatch 中查看，即可在 CloudWatch 控制台中查看指标控制面板。

向现有的 CloudWatch 控制面板添加 Performance Insights 指标

向现有的 CloudWatch 控制面板添加预配置或自定义指标控制面板。您可以向指标控制面板添加标签，使其单独显示在 CloudWatch 控制面板的某个部分中。

将指标导出到现有的 CloudWatch 控制面板

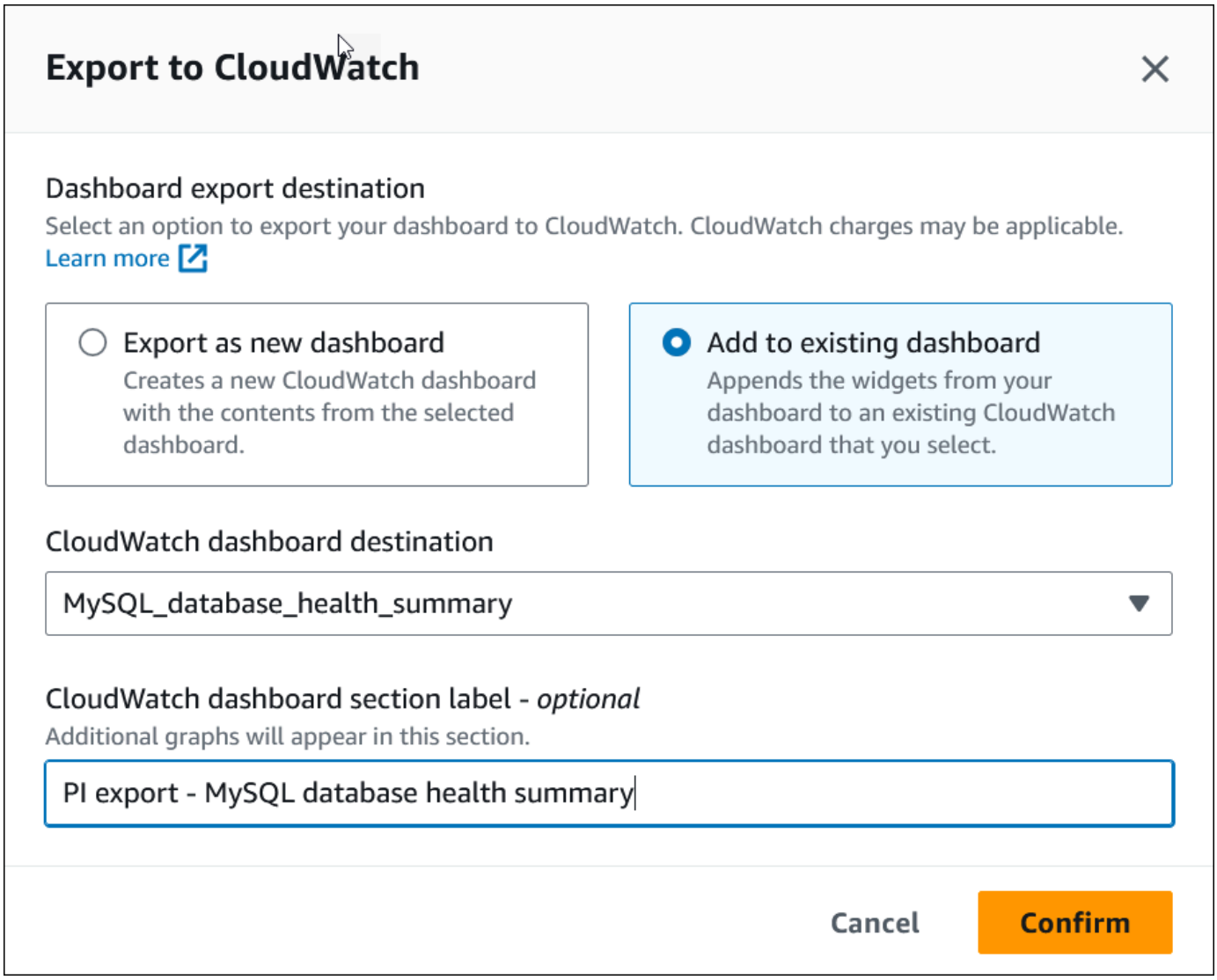
1. 通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。

2. 在左侧导航窗格中，选择 Performance Insights。
3. 选择一个数据库实例。

   将显示该数据库实例的 Performance Insights 控制面板。
4. 向下滚动并选择指标。

   默认情况下，会显示包含 Performance Insights 指标的预配置控制面板。
5. 选择预配置或自定义控制面板，然后选择导出到 CloudWatch。

   随后会显示导出到 CloudWatch 窗口。
6. 选择添加到现有控制面板。



7. 指定控制面板目标和标签，然后选择确认。

- CloudWatch 控制面板目标 - 选择现有的 CloudWatch 控制面板。
- CloudWatch 控制面板部分标签 - 可选 - 输入要在 CloudWatch 控制面板的此部分中显示的 Performance Insights 指标的名称。

控制面板导出成功后，横幅会显示一条消息。

8. 选择横幅中的链接或在 CloudWatch 中查看，即可在 CloudWatch 控制台中查看指标控制面板。

在 CloudWatch 中查看 Performance Insights 指标小组件

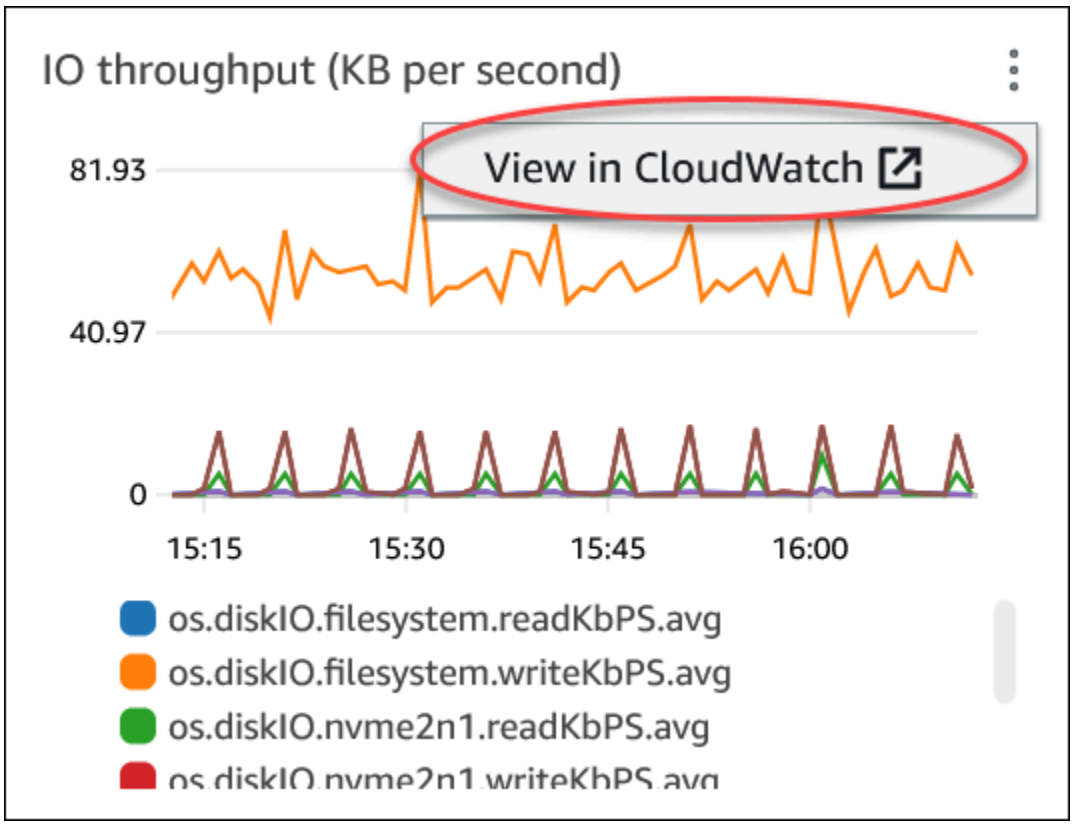
在 Amazon RDS Performance Insights 控制面板中选择一个 Performance Insights 指标小组件，然后在 CloudWatch 控制台中查看指标数据。

导出指标小组件并在 CloudWatch 控制台中查看指标数据

1. 通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在左侧导航窗格中，选择 Performance Insights。
3. 选择一个数据库实例。

   将显示该数据库实例的 Performance Insights 控制面板。
4. 向下滚动到指标。

   默认情况下，会显示包含 Performance Insights 指标的预配置控制面板。
5. 选择一个指标小组件，然后在菜单中选择在 CloudWatch 中查看。



指标数据显示在 CloudWatch 控制台中。

创建 CloudWatch 警报以监控 Amazon Aurora

您可以创建在警报改变状态时发送 Amazon SNS 消息的 CloudWatch 警报。警报会监控您指定的时间段内的某个指标。此外，警报会根据指标值在多个时间段内对比给定阈值的情况执行一项或多项操作。操作是向 Amazon SNS 主题或 Amazon EC2 Auto Scaling 策略发送的通知。

警报只会调用操作进行持续的状态变更。CloudWatch 警报不会仅仅因为处于特定状态而调用操作。该状态必须改变并在指定数量的时间段内一直保持。

> **Note**
>
> 对于 Aurora，请使用 `WRITER` 或 `READER` 角色指标设置警报，而不是依赖特定数据库实例的指标。Aurora 数据库实例角色可以随着时间的推移而更改。您可以在 CloudWatch 控制台中找到这些基于角色的指标。
>
> Aurora Auto Scaling 会根据 `READER` 角色指标自动设置警报。有关 Aurora Auto Scaling 的更多信息，请参阅 Amazon Aurora Auto Scaling 与 Aurora 副本结合使用。

您可以使用 CloudWatch 控制台中的 DB_PERF_INSIGHTS 指标数学函数来查询 Amazon RDS 以获取性能详情计数器指标。DB_PERF_INSIGHTS 函数还包括以亚分钟为间隔的 DBLoad 指标。您可以根据这些指标设置 CloudWatch 警报。

有关如何创建警报的更多详细信息，请参阅针对 AWS 数据库中的性能详情计数器指标创建警报。

使用 AWS CLI 设置警报

- 调用 `put-metric-alarm`。有关更多信息，请参阅 AWS CLI Command Reference。

使用 CloudWatch API 设置警报

- 调用 `PutMetricAlarm`。有关更多信息，请参阅 Amazon CloudWatch API 参考。

有关设置 Amazon SNS 主题和创建告警的更多信息，请参阅使用 Amazon CloudWatch 告警。

使用 CloudWatch 数据库洞察监控 Amazon Aurora 数据库

使用数据库洞察监控 Amazon Aurora 数据库实例的实例集的数据库负载（DB Load）。数据库负载可衡量数据库中会话活动的级别。您可以使用数据库洞察来大规模分析 Amazon Aurora 数据库的性能并排查其故障。

借助数据库洞察，您可以直观呈现数据库实例集中的数据库负载，并按等待状态、SQL 语句、主机或用户来筛选负载。

默认情况下，RDS 为您的 Amazon Aurora 数据库启用数据库洞察的标准模式。当您为数据库集群开启数据库洞察的高级模式时，RDS 会为集群中的每个数据库实例启用数据库洞察。

有关在 Amazon CloudWatch 控制台中使用数据库洞察的信息，请参阅《Amazon CloudWatch 用户指南》中的 CloudWatch 数据库洞察。

定价

有关定价的信息，请参阅 Amazon CloudWatch 定价。

主题

- 支持数据库洞察的 Amazon Aurora 数据库引擎、区域和实例类
- 开启适用于 Amazon Aurora 的数据库洞察的高级模式
- 开启适用于 Amazon Aurora 的数据库洞察的标准模式
- 使用适用于 Amazon Aurora 的数据库洞察配置您的数据库以监控慢速 SQL 查询
- 适用于 Amazon Aurora 的数据库洞察的注意事项

支持数据库洞察的 Amazon Aurora 数据库引擎、区域和实例类

下表提供支持数据库洞察的 Amazon Aurora 数据库引擎。

Amazon Aurora 数据库引擎	支持的引擎版本和区域	实例类限制
Amazon Aurora MySQL 兼容版	有关适用于 Aurora MySQL 的数据库洞察的版本和区域可用性的更多信息，请参阅 适用于	数据库洞察有以下引擎类限制： • db.t2：不支持

Amazon Aurora 数据库引擎	支持的引擎版本和区域	实例类限制
	Aurora MySQL 的 Performance Insights。	• db.t3：不支持 • db.t4g.micro 和 db.t4g.small – 不支持
Amazon Aurora PostgreSQL 兼容版	有关适用于 Aurora PostgreSQL 的数据库洞察的版本和区域可用性的更多信息，请参阅 适用于 Aurora PostgreSQL 的 Performance Insights。	不适用
Aurora PostgreSQL Limitless Database	有关将 Database Insights 用于 Aurora PostgreSQL 无限制数据库的更多信息，请参阅使用 CloudWatch Database Insights 监控 Aurora PostgreSQL 无限制数据库。	不适用

数据库洞察支持 Amazon Aurora Serverless v2。

支持数据库洞察功能的 Amazon Aurora 数据库引擎、区域和实例类

下表提供支持数据库洞察功能的 Amazon Aurora 数据库引擎。

功能	定价套餐	支持的区域	支持的数据库引擎	支持的实例类
Performance Insights 的 SQL 统计数据	全部	全部	全部	全部
分析一段时间内的数据库性能	仅限付费套餐	全部	全部	全部，但 db.server

功能	定价套餐	支持的区域	支持的数据库引擎	支持的实例类
				less（Aurora Serverless v2）除外

功能	定价套餐	支持的区域	支持的数据库引擎	支持的实例类
查看性能详情主动建议	仅限付费套餐	• 美国东部（俄亥俄州） • 美国东部（弗吉尼亚州北部） • 美国西部（加利福尼亚北部） • 美国西部（俄勒冈州） • 亚太地区（孟买） • 亚太地区（首尔） • 亚太地区（新加坡） • 亚太地区（悉尼） • 亚太地区（东京） • 加拿大（中部） • 欧洲地区（法兰克福） • 欧洲地区（爱尔兰） • 欧洲地区（伦敦） • 欧洲地区（巴黎）	全部	全部，但 db.serverless（Aurora Serverless v2）除外

功能	定价套餐	支持的区域	支持的数据库引擎	支持的实例类
		• 欧洲地区（斯德哥尔摩） • 南美洲（圣保罗）		

支持数据库洞察的 Amazon Aurora 区域

Aurora 在以下 AWS 区域中支持数据库洞察。

- 美国东部（弗吉尼亚州北部）
- 美国东部（俄亥俄州）
- 美国西部（加利福尼亚北部）
- 美国西部（俄勒冈州）
- 非洲（开普敦）
- 亚太地区（香港）
- 亚太地区（海得拉巴）
- 亚太地区（雅加达）
- 亚太地区（马来西亚）
- 亚太地区（墨尔本）
- 亚太地区（孟买）
- 亚太地区（大阪）
- 亚太地区（首尔）
- 亚太地区（新加坡）
- 亚太地区（悉尼）
- 亚太地区（东京）
- 加拿大（中部）
- 加拿大西部（卡尔加里）
- 欧洲地区（法兰克福）

- 欧洲地区（爱尔兰）
- 欧洲地区（伦敦）
- 欧洲地区（米兰）
- 欧洲地区（巴黎）
- 欧洲地区（西班牙）
- 欧洲地区（斯德哥尔摩）
- 欧洲（苏黎世）
- 以色列（特拉维夫）
- 中东（巴林）
- 中东（阿联酋）
- 南美洲（圣保罗）
- AWS GovCloud（美国东部）
- AWS GovCloud（美国西部）

开启适用于 Amazon Aurora 的数据库洞察的高级模式

要开启适用于 Amazon Aurora 的数据库洞察的高级模式，请使用以下过程。

在创建数据库集群时开启数据库洞察的高级模式

在为 Amazon Aurora 创建数据库时，开启数据库洞察的高级模式。

Console

在控制台中，您可以在创建数据库集群时开启数据库洞察的高级模式。数据库洞察的设置适用于数据库集群中的所有数据库实例。

在使用控制台创建数据库集群时，开启数据库洞察的高级模式

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 选择数据库。
3. 选择创建数据库。
4. 在数据库洞察部分，选择高级模式。然后，选择以下选项：

- 保留 – 保留性能详情数据的时间。对于数据库洞察的高级模式，保留期必须为 15 个月。
- AWS KMS key - 指定您的 KMS 密钥。性能详情使用您的 KMS 密钥来加密所有潜在的敏感数据。正在传输的数据和静态数据都会被加密。有关更多信息，请参阅 加密 Amazon Aurora 资源。

5. 选择创建数据库。

AWS CLI

要在创建数据库集群时开启数据库洞察的高级模式，请调用 create-db-cluster AWS CLI 命令并提供以下值：

- `--database-insights-mode advanced`，开启数据库洞察的高级模式。
- `--engine`：数据库集群的数据库引擎。
- `--db-cluster-identifier`：数据库集群的标识符。
- `--enable-performance-insights`，开启数据库洞察的性能详情。
- `--performance-insights-retention-period`：数据库集群的数据的保留期。要开启数据库洞察，保留期必须至少为 465 天。

以下示例在创建数据库集群时启用数据库洞察的高级模式。

对于 Linux、macOS 或 Unix：

```
aws rds create-db-cluster \
    --database-insights-mode advanced \
    --engine aurora-postgresql \
    --db-cluster-identifier sample-db-identifier \
    --enable-performance-insights \
    --performance-insights-retention-period 465
```

对于 Windows：

```
aws rds create-db-cluster ^
    --database-insights-mode advanced ^
    --engine aurora-postgresql ^
    --db-cluster-identifier sample-db-identifier ^
    --enable-performance-insights ^
    --performance-insights-retention-period 465
```

RDS API

要在创建数据库集群时开启数据库洞察的高级模式，请为 CreateDBCluster Amazon RDS API 操作指定以下参数。

- `DatabaseInsightsMode` 到 `advanced`
- `EnablePerformanceInsights` 到 `True`
- `PerformanceInsightsRetentionPeriod` 至少为 465 天

在修改数据库集群时开启数据库洞察的高级模式

在为 Amazon Aurora 修改数据库时，开启数据库洞察。修改数据库集群来启用数据库洞察的高级模式不会导致停机。

> **Note**
>
> 要启用数据库洞察，数据库集群中的每个数据库实例都必须具有相同的性能详情和增强型监控设置。

Console

在控制台中，您可以在修改数据库集群时开启数据库洞察的高级模式。数据库洞察的设置适用于数据库集群中的所有数据库实例。

在使用控制台修改数据库集群时，开启数据库洞察的高级模式

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 选择数据库。
3. 选择数据库集群，然后选择修改。
4. 在数据库洞察部分，选择高级模式。然后，选择以下选项：
   - 保留 – 保留性能详情数据的时间。对于数据库洞察的高级模式，保留期必须为 15 个月。
   - AWS KMS key - 指定您的 KMS 密钥。性能详情使用您的 KMS 密钥来加密所有潜在的敏感数据。正在传输的数据和静态数据都会被加密。有关更多信息，请参阅“加密 Amazon Aurora 资源”。
5. 选择继续。

6. 对于修改计划，选择立即应用。如果您选择在下一个计划维护时段内应用，则您的数据库将忽略此设置，并立即开启数据库洞察的高级模式。
7. 选择修改集群。

AWS CLI

要在修改数据库集群时开启数据库洞察的高级模式，请调用 modify-db-cluster AWS CLI 命令并提供以下值：

- `--database-insights-mode advanced`，开启数据库洞察的高级模式。
- `--db-cluster-identifier`：数据库集群的标识符。
- `--enable-performance-insights`，开启数据库洞察的性能详情。
- `--performance-insights-retention-period`：数据库集群的数据的保留期。要开启数据库洞察的高级模式，保留期必须至少为 465 天。

以下示例在修改数据库集群时启用数据库洞察的高级模式。

对于 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster \
    --database-insights-mode advanced \
    --db-cluster-identifier sample-db-identifier \
    --enable-performance-insights \
    --performance-insights-retention-period 465
```

对于 Windows：

```
aws rds modify-db-cluster ^
    --database-insights-mode advanced ^
    --db-cluster-identifier sample-db-identifier ^
    --enable-performance-insights ^
    --performance-insights-retention-period 465
```

RDS API

要在修改数据库集群时开启数据库洞察的高级模式，请为您的 ModifyDBCluster Amazon RDS API 操作指定以下参数。

- `DatabaseInsightsMode` 到 `advanced`

- `EnablePerformanceInsights` 到 `True`
- `PerformanceInsightsRetentionPeriod` 至少为 465 天

开启适用于 Amazon Aurora 的数据库洞察的标准模式

要开启适用于 Amazon Aurora 的数据库洞察的标准模式，请使用以下过程。

在创建数据库集群时开启数据库洞察的标准模式

在为 Amazon Aurora 创建数据库时，开启数据库洞察的标准模式。

Console

在控制台中，您可以在创建数据库集群时开启数据库洞察的标准模式。数据库洞察的设置适用于数据库集群中的所有数据库实例。

在使用控制台创建数据库集群时，开启数据库洞察的标准模式

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 选择数据库。
3. 选择创建数据库。
4. 在数据库洞察部分，选择标准模式。然后，从以下选项中进行选择，打开或关闭性能详情：
   - 要关闭性能详情，请取消选择启用性能详情。
   - 要开启性能详情，请选择启用性能详情。要配置性能详情，请指定以下选项：
     - 保留 – 保留性能详情数据的时间。保留期必须至少为 7 天。
     - AWS KMS key - 指定您的 KMS 密钥。性能详情使用您的 KMS 密钥来加密所有潜在的敏感数据。正在传输的数据和静态数据都会被加密。有关更多信息，请参阅 加密 Amazon Aurora 资源。
5. 选择创建数据库。

AWS CLI

要在创建数据库集群时打开数据库洞察的标准模式，请调用 create-db-cluster AWS CLI 命令并提供以下值：

- `--database-insights-mode standard`，开启数据库洞察的标准模式。

- `--engine`：数据库集群的数据库引擎。
- `--db-cluster-identifier`：数据库集群的标识符。
- `--enable-performance-insights` 或 `--no-enable-performance-insights`，打开和关闭性能详情。如果指定 `--enable-performance-insights`，则还必须指定 `--performance-insights-retention-period`，即数据库集群的数据的保留期。保留期必须至少为 7 天。

以下示例在创建数据库集群时启用数据库洞察的标准模式和性能详情。

对于 Linux、macOS 或 Unix：

```
aws rds create-db-cluster \
    --database-insights-mode standard \
    --engine aurora-postgresql \
    --db-cluster-identifier sample-db-identifier \
    --enable-performance-insights \
    --performance-insights-retention-period 7
```

对于 Windows：

```
aws rds create-db-cluster ^
    --database-insights-mode standard ^
    --engine aurora-postgresql ^
    --db-cluster-identifier sample-db-identifier ^
    --enable-performance-insights ^
    --performance-insights-retention-period 7
```

以下示例在创建数据库集群时启用数据库洞察的标准模式，但禁用性能详情。

对于 Linux、macOS 或 Unix：

```
aws rds create-db-cluster \
    --database-insights-mode standard \
    --engine aurora-postgresql \
    --db-cluster-identifier sample-db-identifier \
    --no-enable-performance-insights
```

对于 Windows：

```
aws rds create-db-cluster ^
```

```
    --database-insights-mode standard ^
    --engine aurora-postgresql ^
    --db-cluster-identifier sample-db-identifier ^
    --no-enable-performance-insights
```

RDS API

要在创建数据库集群时开启数据库洞察的标准模式，请为您的 CreateDBCluster Amazon RDS API 操作指定以下参数。

- `DatabaseInsightsMode` 到 `standard`
- 将 `EnablePerformanceInsights` 设置为 `True` 或 `False`。如果将 `EnablePerformanceInsights` 设置为 `True`，则必须将 `PerformanceInsightsRetentionPeriod` 设置为至少 7 天。

在修改数据库集群时，开启数据库洞察的标准模式

在为 Amazon Aurora 修改数据库时，开启数据库洞察的标准模式。修改数据库集群来启用数据库洞察的标准模式不会导致停机。

> **Note**
>
> 要启用数据库洞察，数据库集群中的每个数据库实例都必须具有相同的性能详情和增强型监控设置。

Console

在控制台中，您可以在修改数据库集群时开启数据库洞察的标准模式。数据库洞察的设置适用于数据库集群中的所有数据库实例。

在使用控制台修改数据库集群时，开启数据库洞察的标准模式

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 选择数据库。
3. 选择数据库集群，然后选择修改。
4. 在数据库洞察部分，选择标准模式。然后从以下选项中进行选择：

- 要关闭性能详情，请取消选择启用性能详情。
- 要开启性能详情，请选择启用性能详情。要配置性能详情，请指定以下选项：
  - 保留 – 保留性能详情数据的时间。保留期必须至少为 7 天。
  - AWS KMS key - 指定您的 KMS 密钥。性能详情使用您的 KMS 密钥来加密所有潜在的敏感数据。正在传输的数据和静态数据都会被加密。有关更多信息，请参阅“加密 Amazon Aurora 资源”。

5. 选择继续。
6. 对于修改计划，选择立即应用。如果您选择在下一个计划维护时段内应用，则您的数据库将忽略此设置，并立即开启数据库洞察的标准模式。
7. 选择修改集群。

AWS CLI

要在修改数据库集群时开启数据库洞察的标准模式，请调用 modify-db-cluster AWS CLI 命令并提供以下值：

- `--database-insights-mode standard`，开启数据库洞察的标准模式。
- `--db-cluster-identifier`：数据库集群的标识符。
- `--enable-performance-insights` 或 `--no-enable-performance-insights`，打开和关闭性能详情。如果指定 `--enable-performance-insights`，则还必须指定 `--performance-insights-retention-period`，即数据库集群的数据的保留期。保留期必须至少为 7 天。

以下示例在修改数据库集群时启用数据库洞察的标准模式并启用性能详情。

对于 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster \
    --database-insights-mode standard \
    --db-cluster-identifier sample-db-identifier \
    --enable-performance-insights \
    --performance-insights-retention-period 7
```

对于 Windows：

```
aws rds modify-db-cluster ^
```

```
    --database-insights-mode standard ^
    --db-cluster-identifier sample-db-identifier ^
    --enable-performance-insights ^
    --performance-insights-retention-period 7
```

以下示例在修改数据库集群时启用数据库洞察的标准模式并禁用性能详情。

对于 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster \
    --database-insights-mode standard \
    --db-cluster-identifier sample-db-identifier \
    --no-enable-performance-insights
```

对于 Windows：

```
aws rds modify-db-cluster ^
    --database-insights-mode standard ^
    --db-cluster-identifier sample-db-identifier ^
    --no-enable-performance-insights
```

RDS API

要在修改数据库集群时开启数据库洞察的标准模式，请为您的 ModifyDBCluster Amazon RDS API 操作指定以下参数。

- `DatabaseInsightsMode` 到 `standard`
- 将 `EnablePerformanceInsights` 设置为 `True` 或 `False`。如果将 `EnablePerformanceInsights` 设置为 `True`，则必须将 `PerformanceInsightsRetentionPeriod` 设置为至少 7 天。

使用适用于 Amazon Aurora 的数据库洞察配置您的数据库以监控慢速 SQL 查询

要监控数据库的慢速 SQL 查询，可以使用“数据库洞察”控制面板中的慢速 SQL 查询部分。在配置数据库以监控慢速 SQL 查询之前，慢速 SQL 查询部分为空。

有关在“数据库洞察”控制面板中监控慢速 SQL 查询的更多信息，请参阅《Amazon CloudWatch 用户指南》中的查看数据库实例控制面板以获取 CloudWatch 数据库洞察。

要使用数据库洞察配置您的数据库以监控慢速 SQL 查询，请完成以下步骤：

1. 启用到 CloudWatch Logs 的日志导出。
2. 创建或修改数据库集群的数据库集群参数组。

有关配置日志导出的信息，请参阅《Amazon Aurora 用户指南》中的将数据库日志发布到 Amazon CloudWatch Logs。

要创建或修改数据库集群参数组，请参阅以下主题。

- 在 Amazon Aurora 中创建数据库集群参数组
- 在 Amazon Aurora 中修改数据库集群参数组中的参数

Amazon Aurora MySQL

要将您的 Amazon Aurora MySQL 数据库集群配置为监控慢速 SQL 查询，请设置以下参数。

- `slow_query_log` – 设置为 `1`
- `long_query_time` – 设置为 `1.0`
- `log_output` – 设置为 `FILE`

Amazon Aurora PostgreSQL

要将您的 Amazon Aurora PostgreSQL 数据库集群配置为监控慢速 SQL 查询，请设置以下参数。请注意，设置这些参数可能会降低数据库集群的性能。

- `log_min_duration_statement` – 设置为 `1000`
- `log_statement` – 设置为 `none`
- `log_destination` – 设置为 `stderr`

> **Note**
>
> 对于 Aurora MySQL，可以将参数 `long_query_time` 配置为 1 微秒的粒度。例如，您可以将此参数设置为 `0.000001`。根据数据库实例上的查询量，参数 `long_query_time` 的值可能会降低性能。从值 `1.0` 开始，然后根据工作负载进行调整。当您将此参数设置为 `0` 时，数据库洞察会记录所有查询。

有关 Aurora MySQL 和 Aurora PostgreSQL 日志的信息，请参阅以下内容。

- Aurora MySQL 数据库日志文件
- Aurora PostgreSQL 数据库日志文件

适用于 Amazon Aurora 的数据库洞察的注意事项

以下是适用于 Amazon Aurora 的数据库洞察的注意事项。

- 您无法管理数据库集群中数据库实例的数据库洞察。
- 要启用数据库洞察的高级模式，您必须启用性能详情并将性能详情的保留期设置为至少 465 天（15 个月）。除了数据库洞察的成本外，将性能详情的保留期设置为 15 个月没有额外成本。有关数据库洞察的定价信息，请参阅 Amazon CloudWatch 定价。
- 要启用数据库洞察，数据库集群中的每个数据库实例都必须具有相同的性能详情和增强型监控设置。
- 修改数据库集群来启用数据库洞察的任一模式都不会导致停机。

在 Amazon Aurora 上使用性能详情监控数据库负载

Important

AWS 已宣布性能详情的生命周期终止日期为：2025 年 11 月 30 日。在此日期之后，Amazon RDS 将不再支持性能详情控制台体验、灵活的保留期（1-24 个月）及其关联定价。性能详情 API 将继续存在，定价不变。性能详情 API 的费用将与 CloudWatch 数据库洞察的费用一起显示在您的 AWS 账单中。

我们建议您在 2025 年 11 月 30 日之前，将使用性能详情付费套餐的所有数据库集群升级到数据库洞察的高级模式。有关升级到数据库洞察的高级模式的信息，请参阅开启适用于 Amazon Aurora 的数据库洞察的高级模式。

如果您不采取任何行动，则使用性能详情的数据库集群将默认为使用数据库洞察的标准模式。在数据库洞察的标准模式下，您可能会在 7 天后无法访问性能数据历史记录，并且可能无法在 Amazon RDS 控制台中使用执行计划和按需分析功能。在 2025 年 11 月 30 日之后，只有数据库洞察的高级模式将支持执行计划和按需分析。

借助 CloudWatch 数据库洞察，您可以监控数据库实例集的数据库负载，并对性能进行大规模分析和故障排除。有关数据库洞察的更多信息，请参阅使用 CloudWatch 数据库洞察监控 Amazon Aurora 数据库。有关定价信息，请参阅 Amazon CloudWatch 定价。

Performance Insights 在现有 Amazon Aurora 监控功能的基础上进行了扩展，以便展示并帮助您分析集群性能。利用 Performance Insights 控制面板，您可以可视化 Amazon Aurora 集群负载上的数据库负载，并按等待状态、SQL 语句、主机或用户来筛选负载。有关将 Performance Insights 与 Amazon DocumentDB 结合使用的信息，请参阅 Amazon DocumentDB 开发人员指南。

主题

- Amazon Aurora 上的 Performance Insights 概述
- 为 Aurora 打开和关闭性能详情
- Aurora MySQL 上性能详情的性能架构概述
- 为 Performance Insights 配置访问策略
- 使用 Performance Insights 控制面板分析指标
- 查看性能详情主动建议
- 使用 Performance Insights API for Aurora 检索指标
- 使用 AWS CloudTrail 记录 Performance Insights 调用
- 性能详情 API 和接口 VPC 端点（AWS PrivateLink）

Amazon Aurora 上的 Performance Insights 概述

Important

AWS 已宣布性能详情的生命周期终止日期为：2025 年 11 月 30 日。在此日期之后，Amazon RDS 将不再支持性能详情控制台体验、灵活的保留期（1-24 个月）及其关联定价。性能详情 API 将继续存在，定价不变。性能详情 API 的费用将与 CloudWatch 数据库洞察的费用一起显示在您的 AWS 账单中。

我们建议您在 2025 年 11 月 30 日之前，将使用性能详情付费套餐的所有数据库集群升级到数据库洞察的高级模式。有关升级到数据库洞察的高级模式的信息，请参阅开启适用于 Amazon Aurora 的数据库洞察的高级模式。

如果您不采取任何行动，则使用性能详情的数据库集群将默认为使用数据库洞察的标准模式。在数据库洞察的标准模式下，您可能会在 7 天后无法访问性能数据历史记录，并且可能无法在 Amazon RDS 控制台中使用执行计划和按需分析功能。在 2025 年 11 月 30 日之后，只有数据库洞察的高级模式将支持执行计划和按需分析。

借助 CloudWatch 数据库洞察，您可以监控数据库实例集的数据库负载，并对性能进行大规模分析和故障排除。有关数据库洞察的更多信息，请参阅使用 CloudWatch 数据库洞察监控 Amazon Aurora 数据库。有关定价信息，请参阅 Amazon CloudWatch 定价。

默认情况下，RDS 在控制台创建向导中为所有 Amazon RDS 引擎启用 Performance Insights。如果您在数据库集群级别打开 Performance Insights，RDS 会为集群中的每个数据库实例启用 Performance Insights。如果一个数据库实例中有多个数据库，则 Performance Insights 会聚合性能数据。

可在以下视频中大致了解 Amazon Aurora 的 Performance Insights。

使用 Performance Insights 分析 Amazon Aurora PostgreSQL 的性能

主题

- 数据库负载
- 最大 CPU
- 支持性能详情的 Amazon Aurora 数据库引擎、区域和实例类
- 性能详情的定价和数据留存

数据库负载

数据库负载（DB 负载）衡量数据库中的会话活动级别。DBLoad 是 Performance Insights 中的关键指标，Performance Insights 每秒都会收集数据库负载。

主题

- 活动的会话
- 平均活动会话数
- 平均活动执行数
- 尺寸

活动的会话

数据库会话表示的是应用程序与关系数据库的对话。活动的会话 是已将作业提交到数据库引擎并且正在等待响应的连接。

当会话在 CPU 上运行或等待资源变为可用以便继续执行时，该会话即处于活动状态。例如，活动的会话可能会等待页面（或块）读入内存中，然后占用 CPU 以从页面读取数据。

平均活动会话数

平均活动会话数（AAS）是 Performance Insights 中 DBLoad 指标的单位。它衡量数据库上有多少个会话同时处于活动状态。

Performance Insights 每秒都会对并行运行查询的会话数进行采样。对于每个活动的会话，Performance Insights 都会收集以下数据：

- SQL 语句
- 会话状态（正在 CPU 上运行或正在等待）
- Host
- 运行 SQL 的用户

Performance Insights 通过以下方式计算 AAS：对于特定时间段内，将会话总数除以样本数。例如，下表显示了正在运行的查询的 5 个连续样本，其间隔为 1 秒。

示例	运行查询的会话数	AAS	计算
1	2	2	总计 2 个会话/1 个样本
2	0	1	总计 2 个会话/2 个样本
3	4	2	总计 6 个会话/3 个样本
4	0	1.5	总计 6 个会话/4 个样本
5	4	2	总计 10 个会话/5 个样本

在前面的示例中，该时间间隔的数据库负载为 2 AAS。这一测量结果表明，在采集 5 个样本的时间间隔内，在任何给定时间，平均有 2 个会话处于活动状态。

平均活动执行数

每秒的平均活动执行数 (AAE) 与平均活动会话数相关。为了计算平均活动执行数，Performance Insights 使用查询的总执行时间除以时间间隔。下表显示了上表中同一查询的平均活动执行数计算。

运行时间（秒）	总执行时间（秒）	AAE	计算
60	120	2	120 秒执行时间/60 秒运行时间
120	120	1	120 秒执行时间/120 秒运行时间
180	380	2.1.1	380 秒执行时间/180 秒运行时间
240	380	1.58	380 秒执行时间/240 秒运行时间
300	600	2	600 秒执行时间/300 秒运行时间

在大多数情况下，查询的平均活动会话数和平均活动执行数大致相同。但是，由于计算所用的数据是不同的数据源，因此计算通常略有不同。

尺寸

db.load 指标不同于其他时间序列指标，因为您可以将它分为称为维度的子组件。您可以将维度视为 DBLoad 指标的不同特征的“切片依据”类别。

诊断性能问题时，以下维度通常最有用：

主题

- 等待事件
- 主要 SQL

有关 Aurora 引擎维度的完整列表，请参阅 按维度切片的数据库负载。

等待事件

等待事件 会导致 SQL 语句等待特定事件发生，然后才能继续运行。等待事件是数据库负载的一个重要维度（或称类别），因为它们指示了工作受阻的位置。

每个活动的会话要么会在 CPU 上运行，要么仍在等待。例如，在搜索内存寻找缓冲区、执行计算或运行过程代码时，会话都会占用 CPU。当会话不占用 CPU 时，它们可能正在等待内存缓冲区变为空闲、等待要读取的数据文件或等待将要写入的日志。会话等待资源的时间越长，它在 CPU 上运行的时间就越少。

当您优化数据库时，经常尝试找出会话正在等待的资源。例如，两三个等待事件可能会占据数据库负载的 90%。这一测量结果表明，平均而言，活动会话花费了大部分时间用于等待少量资源。如果您能找出导致这些等待的原因，就可以尝试提出解决方案了。

等待事件因数据库引擎而异：

- 有关 Aurora MySQL 的最常见等待事件的列表，请参阅 Aurora MySQL 等待事件。要了解如何使用这些等待事件进行优化，请参阅 优化 Aurora MySQL。
- 有关所有 MySQL 等待事件的信息，请参阅 MySQL 文档中的等待事件摘要表。
- 有关 Aurora PostgreSQL 的最常见等待事件的列表，请参阅 Amazon Aurora PostgreSQL 等待事件。要了解如何使用这些等待事件进行优化，请参阅 使用 Aurora PostgreSQL 的等待事件进行优化。
- 有关所有 PostgreSQL 等待事件的信息，请参阅 PostgreSQL 文档中的统计数据收集器 > 等待事件表。

主要 SQL

等待事件显示运行中的瓶颈，而主要 SQL 则显示哪些查询对数据库负载的影响最大。例如，当前可能正在数据库上运行着许多查询，但某一个查询可能会占用 99% 的数据库负载。在这种情况下，高负载可能表示查询存在问题。

默认情况下，Performance Insights 控制台会显示导致数据库负载的主要 SQL 查询。控制台还显示每个语句的相关统计信息。要诊断特定语句的性能问题，可以检查其执行计划。

最大 CPU

在控制面板中，数据库负载图表会收集、聚合和显示会话信息。要查看活动会话是否超过最大 CPU，请查看它们与最大 vCPU 线的关系。Performance Insights 根据数据库实例的 vCPU（虚拟 CPU）内核数，来决定最大 vCPU 值。对于 Aurora Serverless v2，最大 vCPU 表示 vCPU 的估计数量。

一次只能在 vCPU 上运行一个进程。如果进程数超过 vCPU 的数量，则进程开始排队。当排队增加时，数据库性能就会降低。如果数据库负载经常高于最大 vCPU 线并且主要等待状态为 CPU，则表示 CPU 过载。在这种情况下，您可能需要限制与实例的连接数，优化具有高 CPU 负载的任何 SQL 查询，或考虑使用更大的实例类。如果始终有大量实例处于任何等待状态，则表示可能存在要解决的瓶颈或资源争用问题。即使数据库负载未越过最大 vCPU 线，也可能会出现此问题。

支持性能详情的 Amazon Aurora 数据库引擎、区域和实例类

Important

AWS 已宣布性能详情的生命周期终止日期为：2025 年 11 月 30 日。在此日期之后，Amazon RDS 将不再支持性能详情控制台体验、灵活的保留期（1-24 个月）及其关联定价。性能详情 API 将继续存在，定价不变。性能详情 API 的费用将与 CloudWatch 数据库洞察的费用一起显示在您的 AWS 账单中。

我们建议您在 2025 年 11 月 30 日之前，将使用性能详情付费套餐的所有数据库集群升级到数据库洞察的高级模式。有关升级到数据库洞察的高级模式的信息，请参阅开启适用于 Amazon Aurora 的数据库洞察的高级模式。

如果您不采取任何行动，则使用性能详情的数据库集群将默认为使用数据库洞察的标准模式。在数据库洞察的标准模式下，您可能会在 7 天后无法访问性能数据历史记录，并且可能无法在 Amazon RDS 控制台中使用执行计划和按需分析功能。在 2025 年 11 月 30 日之后，只有数据库洞察的高级模式将支持执行计划和按需分析。

借助 CloudWatch 数据库洞察，您可以监控数据库实例集的数据库负载，并对性能进行大规模分析和故障排除。有关数据库洞察的更多信息，请参阅使用 CloudWatch 数据库洞察监控 Amazon Aurora 数据库。有关定价信息，请参阅 Amazon CloudWatch 定价。

下表提供支持性能详情的 Amazon Aurora 数据库引擎。

Amazon Aurora 数据库引擎	支持的引擎版本和区域	实例类限制
Amazon Aurora MySQL 兼容版	有关适用于 Aurora MySQL 的性能详情的版本和区域可用性的更多信息，请参阅 适用于 Aurora MySQL 的 Performance Insights。	Performance Insights 有以下引擎类限制： • db.t2：不支持 • db.t3：不支持 • db.t4g.micro 和 db.t4g.small – 不支持
Amazon Aurora PostgreSQL 兼容版	有关适用于 Aurora PostgreSQL 的性能详情的版本和区域可用性的更多信息，请参阅 适用于 Aurora PostgreSQL 的 Performance Insights。	不适用

支持性能详情功能的 Amazon Aurora 数据库引擎、区域和实例类

下表提供支持性能详情功能的 Amazon Aurora 数据库引擎。

功能	定价套餐	支持的区域	支持的数据库引擎	支持的实例类
Performance Insights 的 SQL 统计数据	全部	全部	全部	全部
分析一段时间内的数据库性能	仅限付费套餐	全部	全部	全部，但 db.serverless（Aurora

功能	定价套餐	支持的区域	支持的数据库引擎	支持的实例类
				Serverless v2 ）除外

功能	定价套餐	支持的区域	支持的数据库引擎	支持的实例类
查看性能详情主动建议	仅限付费套餐	• 美国东部（俄亥俄州） • 美国东部（弗吉尼亚州北部） • 美国西部（加利福尼亚北部） • 美国西部（俄勒冈州） • 亚太地区（孟买） • 亚太地区（首尔） • 亚太地区（新加坡） • 亚太地区（悉尼） • 亚太地区（东京） • 加拿大（中部） • 欧洲地区（法兰克福） • 欧洲地区（爱尔兰） • 欧洲地区（伦敦） • 欧洲地区（巴黎）	全部	全部，但 db.serverless（Aurora Serverless v2）除外

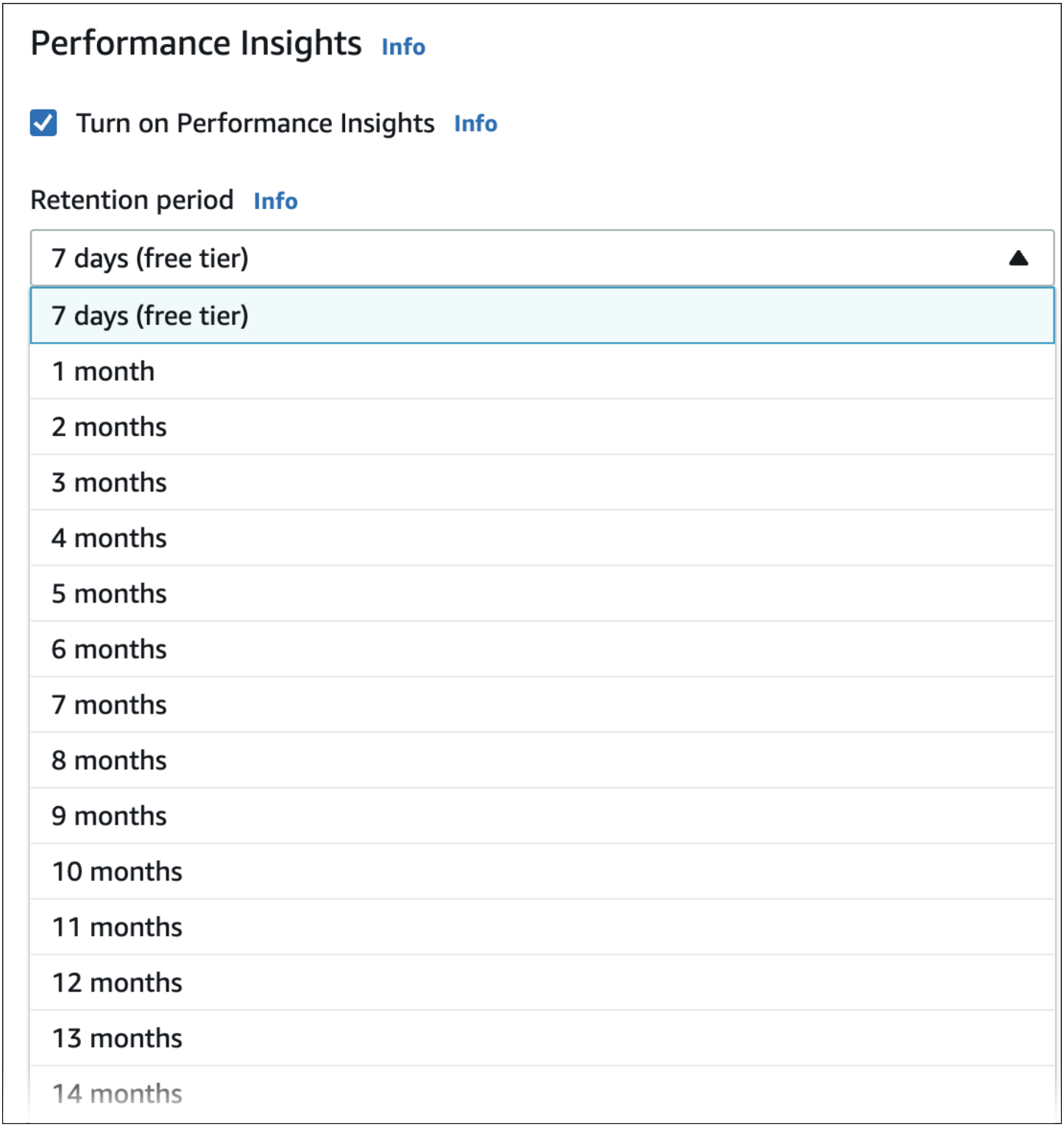
功能	定价套餐	支持的区域	支持的数据库引擎	支持的实例类
		• 欧洲地区（斯德哥尔摩） • 南美洲（圣保罗）		

性能详情的定价和数据留存

原定设置情况下，性能详情提供免费套餐，其中包括 7 天的性能数据历史记录和每月 100 万个 API 请求。您也可以购买更长的保留期。有关完整的定价信息，请参阅性能详情定价。

在 RDS 控制台中，您可以为性能详情数据选择以下任何保留期：

- 原定设置（7 天）
- ***n*** 个月，其中 ***n*** 是 1–24 之间的数字



要了解如何使用 AWS CLI 设置保留期，请参阅???。

为 Aurora 打开和关闭性能详情

Important

AWS 已宣布性能详情的生命周期终止日期为：2025 年 11 月 30 日。在此日期之后，Amazon RDS 将不再支持性能详情控制台体验、灵活的保留期（1-24 个月）及其关联定价。性能详情 API 将继续存在，定价不变。性能详情 API 的费用将与 CloudWatch 数据库洞察的费用一起显示在您的 AWS 账单中。

我们建议您在 2025 年 11 月 30 日之前，将使用性能详情付费套餐的所有数据库集群升级到数据库洞察的高级模式。有关升级到数据库洞察的高级模式的信息，请参阅开启适用于 Amazon Aurora 的数据库洞察的高级模式。

如果您不采取任何行动，则使用性能详情的数据库集群将默认为使用数据库洞察的标准模式。在数据库洞察的标准模式下，您可能会在 7 天后无法访问性能数据历史记录，并且可能无法在 Amazon RDS 控制台中使用执行计划和按需分析功能。在 2025 年 11 月 30 日之后，只有数据库洞察的高级模式将支持执行计划和按需分析。

借助 CloudWatch 数据库洞察，您可以监控数据库实例集的数据库负载，并对性能进行大规模分析和故障排除。有关数据库洞察的更多信息，请参阅使用 CloudWatch 数据库洞察监控 Amazon Aurora 数据库。有关定价信息，请参阅 Amazon CloudWatch 定价。

您可以在创建数据库集群，为其打开性能详情。如果需要，可以稍后通过控制台修改数据库集群来关闭性能详情。打开和关闭性能详情不会导致停机、重启或失效转移。

Note

Performance Schema 是 Aurora MySQL 使用的一个可选性能工具。如果打开或关闭 Performance Schema，则需要重新启动。但是，如果您打开或关闭性能详情，则无需重新启动。有关更多信息，请参阅 Aurora MySQL 上性能详情的性能架构概述。

如果将性能详情与 Aurora Global Database 一起使用，则您需要在每个 AWS 区域中分别为数据库启用性能详情。有关更多信息，请参阅 使用 Amazon RDS 性能详情监控 Amazon Aurora Global Database。

性能详情代理占用数据库主机上有限的 CPU 和内存。当数据库负载较高时，代理将通过降低收集数据的频率来限制性能影响。

Console

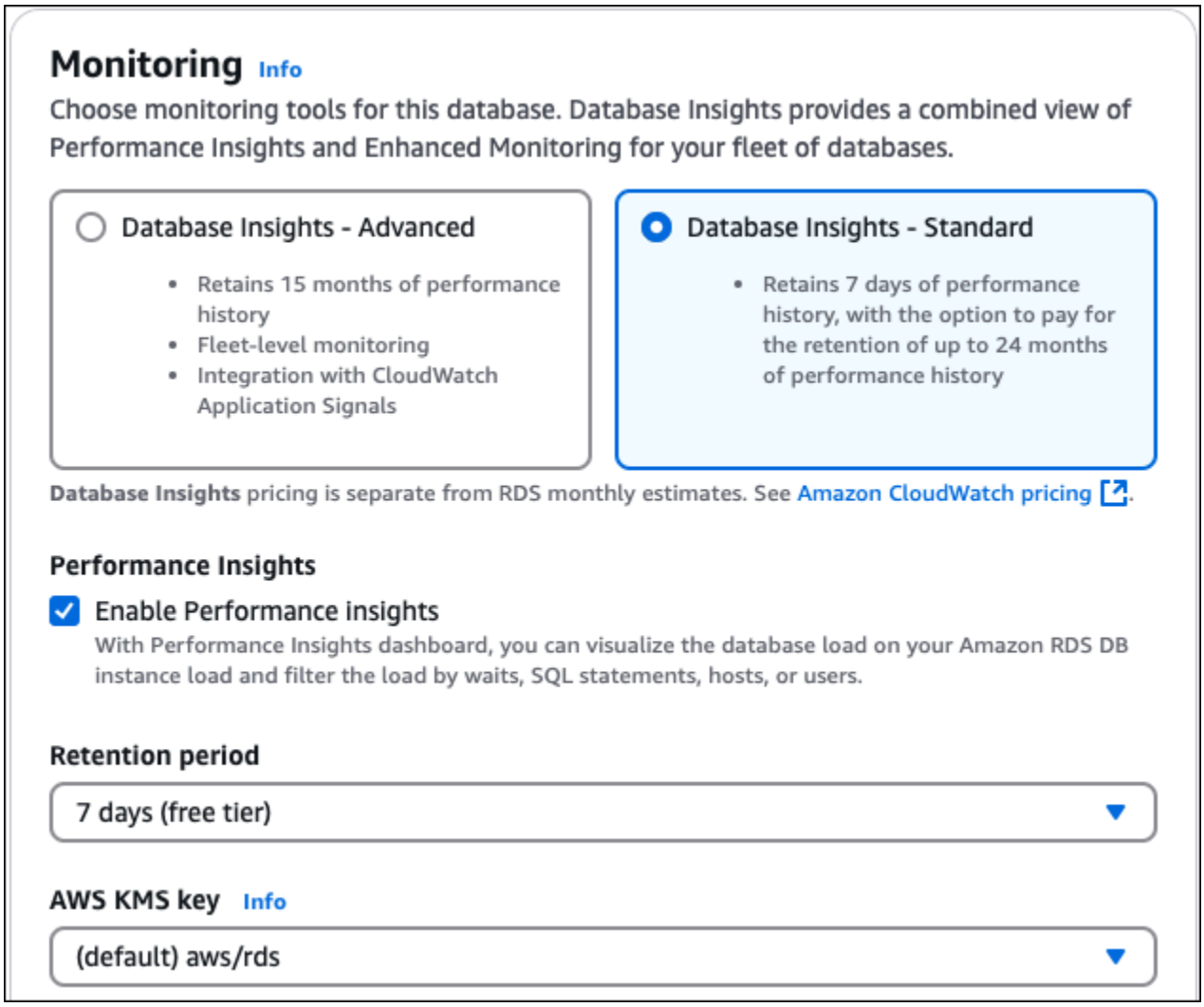
在控制台中，您可以在创建数据库集群时打开或关闭性能详情。启用性能详情可让您管理数据库集群的性能详情设置和选项。集群级别设置适用于集群中的所有数据库实例。

创建数据库集群时打开或关闭性能详情

创建新的数据库集群后，Amazon RDS 默认情况下会启用性能详情。要为数据库集群关闭性能详情，请选择数据库洞察 - 标准选项，然后取消选择启用性能详情选项。

要创建数据库集群，请按照创建 Amazon Aurora 数据库集群中数据库引擎的说明操作。

以下屏幕截图显示了性能详情部分。



如果您选择启用性能详情，您有以下选项：

- 保留期（仅适用于数据库洞察的标准模式）：保留性能详情数据的时间量。免费套餐中的保留设置为 Default (7 days) [默认（7 天）]。要将性能数据保留更长时间，请指定 1–24 个月。有关保留期的更多信息，请参阅性能详情的定价和数据留存。
- AWS KMS key - 指定您的 AWS KMS key。性能详情使用您的 KMS 密钥来加密所有潜在的敏感数据。正在传输的数据和静态数据都会被加密。有关更多信息，请参阅 更改性能详情的 AWS KMS 策略。

在修改数据库集群时打开或关闭性能详情

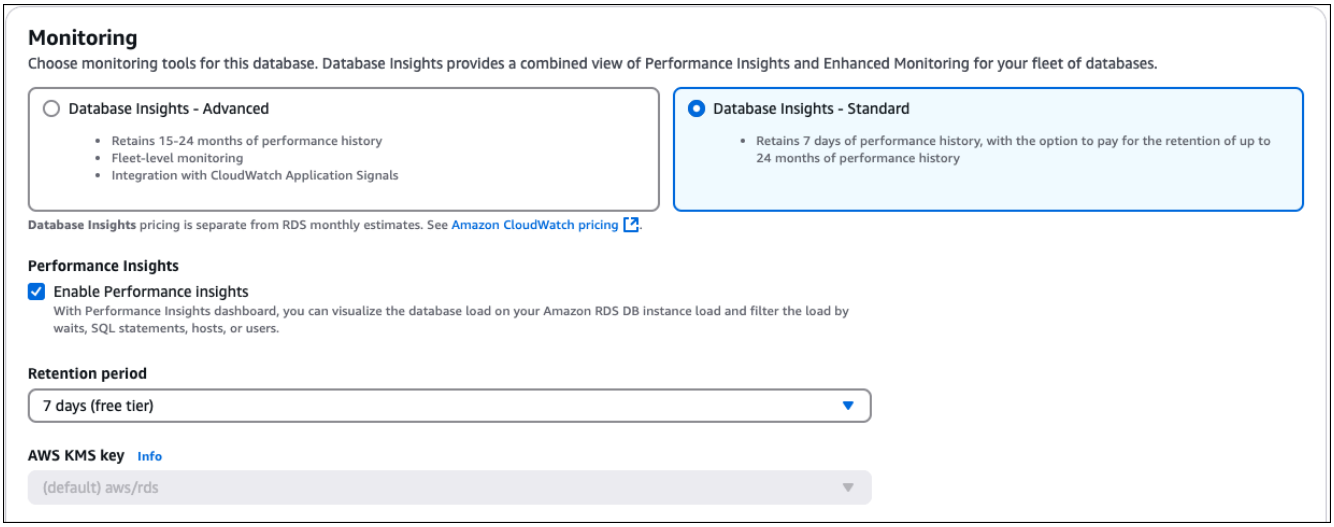
在控制台中，您可以修改数据库集群来管理性能详情。

使用控制台为数据库集群管理性能详情

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 选择数据库。
3. 选择数据库集群，然后选择修改。
4. 要开启性能详情，请选择启用性能详情。要为数据库集群关闭性能详情，请选择数据库洞察 - 标准选项，然后取消选择启用性能详情选项。

   如果您选择启用性能详情，您有以下选项：

   - 保留期（仅适用于数据库洞察的标准模式）：保留性能详情数据的时间量。免费套餐中的保留设置为 Default (7 days) [默认（7 天）]。要将性能数据保留更长时间，请指定 1–24 个月。有关保留期的更多信息，请参阅性能详情的定价和数据留存。
   - AWS KMS key - 指定您的 KMS 密钥。性能详情使用您的 KMS 密钥来加密所有潜在的敏感数据。正在传输的数据和静态数据都会被加密。有关更多信息，请参阅 加密 Amazon Aurora 资源。



5. 选择继续。
6. 对于修改计划，请选择“立即应用”。如果您选择“Apply during the next scheduled maintenance window”（在下一个计划的维护时段内应用），则您的实例将忽略此设置并立即打开性能详情。
7. 选择修改实例。

AWS CLI

使用 create-db-instance AWS CLI 命令时，通过指定 `--enable-performance-insights` 可开启性能详情，并将 `--database-insights-mode` 设置为 `advanced` 或 `standard`。要关闭性能详情，请指定 `--no-enable-performance-insights` 并将 `database-insights-mode` 设置为 `standard`。

您还可以使用以下 AWS CLI 命令指定这些值：

- create-db-cluster
- modify-db-cluster
- create-db-instance-read-replica
- modify-db-instance
- restore-db-instance-from-s3

使用 AWS CLI 为数据库集群管理性能详情

- 调用 modify-db-cluster AWS CLI 命令并提供以下值：
  - `--db-cluster-identifier` – 数据库集群中的数据库实例的名称。
  - `--enable-performance-insights` 以打开或 `--no-enable-performance-insights` 以关闭
  - `database-insights-mode`：数据库集群的数据库洞察模式。要关闭性能详情，请将此值设置为 `standard`。

  以下示例为 `sample-db-cluster` 打开性能详情。

  对于 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster \
 --database-insights-mode standard \
    --db-cluster-identifier sample-db-instance \
    --enable-performance-insights
```

  对于 Windows：

```
aws rds modify-db-cluster ^
 --database-insights-mode standard ^
    --db-cluster-identifier sample-db-instance ^
    --enable-performance-insights
```

在 CLI 中打开性能详情时，可以选择通过 `--performance-insights-retention-period` 选项指定保留性能详情数据的天数。您可以指定 7、*month* * 31（其中 *month* 为 1–23 之间的数字）或 731。例如，如果您想将性能数据保留 3 个月，请指定 93，也就是 3 * 31。默认值为 7 天。有关保留期的更多信息，请参阅性能详情的定价和数据留存。

以下示例为 `sample-db-cluster` 开启性能详情并指定将性能详情数据保留 93 天（3 个月）。

对于 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster \
 --database-insights-mode standard \
    --db-cluster-identifier sample-db-instance \
    --enable-performance-insights \
```

```
    --performance-insights-retention-period 93
```

对于 Windows：

```
aws rds modify-db-cluster ^
 --database-insights-mode standard ^
    --db-cluster-identifier sample-db-instance ^
    --enable-performance-insights ^
    --performance-insights-retention-period 93
```

如果您指定保留期（如 94 天），这不是一个有效值，则 RDS 会发出错误。

```
An error occurred (InvalidParameterValue) when calling the CreateDBInstance
 operation:
Invalid Performance Insights retention period. Valid values are: [7, 31, 62, 93,
 124, 155, 186, 217,
248, 279, 310, 341, 372, 403, 434, 465, 496, 527, 558, 589, 620, 651, 682, 713, 731]
```

> **Note**
>
> 对于未在集群级别管理性能详情的数据库集群中的实例，只能切换性能详情。

RDS API

在使用 CreateDBInstance 操作 Amazon RDS API 操作在数据库集群中创建新数据库实例时，通过将 `EnablePerformanceInsights` 设置为 `True` 来打开性能详情。要关闭性能详情，请将 `EnablePerformanceInsights` 设置为 `False` 并将 `DatabaseInsightsMode` 设置为 `standard`。

您还可以使用以下 API 操作指定 `EnablePerformanceInsights` 值：

- CreateDBCluster
- ModifyDBCluster
- ModifyDBInstance
- CreateDBInstanceReadReplica
- RestoreDBInstanceFromS3

在打开性能详情时，可以通过 PerformanceInsightsRetentionPeriod 参数指定性能详情数据的保留时间，以天为单位。您可以指定 7、*month* * 31（其中 *month* 为 1–23 之间的数字）或 731。例如，如果您想将性能数据保留 3 个月，请指定 93，也就是 3 * 31。默认值为 7 天。有关保留期的更多信息，请参阅性能详情的定价和数据留存。

Aurora MySQL 上性能详情的性能架构概述

Performance Schema 是一项可选功能，用来监控 Aurora MySQL 低细节层次的运行时性能。性能架构旨在将对数据库性能的影响降至最低。Performance Insights 是一个单独的功能，无论有没有 Performance Schema，您都可以使用该功能。

主题

- 性能架构概览
- 性能详情和性能架构
- 让 Performance Insights 自动管理 Performance Schema
- 重启对 Performance Schema 的影响
- 确定性能详情是否正在管理 Performance Schema
- 为 Aurora MySQL 开启 Performance Schema

性能架构概览

Performance Schema 监控 Aurora MySQL 数据库中的事件。事件是一项消耗时间的数据库服务器操作，已进行了分析，以便收集计时信息。以下为事件示例：

- 函数调用
- 等待操作系统
- SQL 执行阶段
- SQL 语句组

PERFORMANCE_SCHEMA 存储引擎是实施 Performance Schema 功能的一种机制。该引擎使用数据库源代码中的分析来收集事件数据。该引擎将事件存储在 performance_schema 数据库的 memory-only 表中。您可以查询 performance_schema，就像您可以查询任何其他表一样。有关更多信息，请参阅 MySQL 参考手册中的 MySQL Performance Schema。

性能详情和性能架构

Performance Insights 和 Performance Schema 是独立的功能，但它们相互关联。Aurora MySQL 的性能详情的行为取决于性能架构是否已开启，如果已开启，则取决于性能详情是否自动管理性能架构。下表描述了该行为。

性能架构已开启	性能详情管理模式	性能详情行为
是	自动	• 收集详细的低级别监控信息 • 每秒收集活动会话指标 • 显示按详细等待事件分类的数据库负载，您可以用它来识别瓶颈
是	手动	• 收集等待事件和每 SQL 指标 • 每秒收集活动会话指标 • 报告用户状态（如插入和发送），这不能帮助您识别瓶颈
否	不适用	• 不收集等待事件、按 SQL 指标或其他详细的低级别监控信息 • 每五秒收集活动会话指标，而不是每秒收集一次 • 报告用户状态（如插入和发送），这不能帮助您识别瓶颈

让 Performance Insights 自动管理 Performance Schema

在打开 Performance Insights 的情况下创建 Aurora MySQL 数据库实例时，也会打开 Performance Schema。在这种情况下，性能详情会自动管理您的性能架构参数。这是建议的配置。

当 Performance Insights 自动管理 Performance Schema 时，`performance_schema` 的来源是 `System default`。

> **Note**
>
> t4g.medium 实例类不支持自动管理性能架构。

您也可以手动管理 Performance Schema。如果选择此选项，请根据下表中的值设置参数。

参数名称	参数值
performance_schema	1（Source（源）列具有值 Modified）
performance-schema-consumer-events-waits-current	ON
performance-schema-instrument	wait/%=ON
performance_schema_consumer_global_instrumentation	1
performance_schema_consumer_thread_instrumentation	1

如果手动更改 performance_schema 参数值，然后稍后想恢复为自动管理，请参阅为 Aurora MySQL 开启 Performance Schema。

> **Important**
>
> 当 Performance Insights 打开 Performance Schema 时，它不会更改参数组值。但是，正在运行的数据库实例上的值会更改。查看更改后的值的唯一方法是运行 SHOW GLOBAL VARIABLES 命令。

重启对 Performance Schema 的影响

Performance Insights 和 Performance Schema 对数据库实例重启的要求不同：

Performance Schema

要打开或关闭此功能，您必须重启数据库实例。

性能详情

要打开或关闭此功能，您不需要重启数据库实例。

如果当前未打开 Performance Schema，并且您在不重启数据库实例的情况下打开 Performance Insights，则不会打开 Performance Schema。

确定性能详情是否正在管理 Performance Schema

要了解性能详情当前是否正在管理所有受支持引擎版本的 Performance Schema，请查看下表。

performance_schema 参数设置	“源”列设置	性能详情正在管理 Performance Schema？
`0`	`System default`	是
`0` 或者 `1`	`Modified`	否

在以下过程中，您可以确定性能详情是否正在自动管理 Performance Schema。

自动确定性能详情是否正在管理 Performance Schema

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 选择参数组。
3. 选择数据库实例的参数组名称。
4. 在搜索栏中输入 **`performance_schema`**。
5. 检查源是否为系统默认设置，以及值是否为 0。如果是这样，性能详情将自动管理 Performance Schema。

   在此处所示的示例中，性能详情未自动管理 Performance Schema。



为 Aurora MySQL 开启 Performance Schema

假设已为数据库实例打开性能详情，但当前未管理 Performance Schema。如果要让性能详情自动管理 Performance Schema，请完成以下步骤。

配置 Performance Schema 以进行自动管理

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 选择参数组。
3. 选择数据库实例的参数组名称。
4. 选择编辑。
5. 在搜索栏中输入 **performance_schema**。
6. 选择 performance_schema 参数。
7. 选择设置为默认值。
8. 选择将值设置为默认值进行确认。
9. 选择保存更改。
10. 重启数据库实例。

> **Important**
>
> 无论何时打开或关闭 Performance Schema，都要确保重启数据库实例。

有关修改实例参数的更多信息，请参阅在 Amazon Aurora 中修改数据库参数组中的参数。有关控制面板的更多信息，请参阅 使用 Performance Insights 控制面板分析指标。有关 MySQL Performance Schema 的更多信息，请参阅 MySQL 文档中的 MySQL Performance Schema（适用于 8.0）和 MySQL Performance Schema（适用于 8.4）。

为 Performance Insights 配置访问策略

要访问 Performance Insights，主体必须拥有 AWS Identity and Access Management (IAM) 的相应权限。您可以通过以下方式授予访问权限：

- 将 AmazonRDSPerformanceInsightsReadOnly 托管式策略附加到权限集或角色，以访问 Performance Insights API 的所有只读操作。附加以下 CloudWatch 权

限：GetMetricStatistics、ListMetrics 和 GetMetricData。有关 CloudWatch 权限的更多信息，请参阅 Amazon CloudWatch 权限参考。

- 将 AmazonRDSPerformanceInsightsFullAccess 托管式策略附加到权限集或角色，以访问 Performance Insights API 的所有操作。附加以下 CloudWatch 权限：GetMetricStatistics、ListMetrics 和 GetMetricData。有关 CloudWatch 权限的更多信息，请参阅 Amazon CloudWatch 权限参考。
- 创建自定义 IAM 策略并将其附加到权限集或角色。

如果您在开启性能详情时指定了客户自主管理型密钥，请确保账户中的用户对 AWS KMS key具有 kms:Decrypt 和 kms:GenerateDataKey 权限。

在以下各节中，将 AWS 托管式策略附加到 IAM 主体，创建自定义 IAM 策略，更改 AWS KMS 策略，并授予对性能详情的精细访问权限。

主题

- 将 AmazonRDSPerformanceInsightsReadOnly 策略附加到 IAM 主体
- 将 AmazonRDSPerformanceInsightsFullAccess 策略附加到 IAM 主体
- 为 Performance Insights 创建自定义 IAM 策略
- 更改性能详情的 AWS KMS 策略
- 授予 Performance Insights 精细访问权限

将 AmazonRDSPerformanceInsightsReadOnly 策略附加到 IAM 主体

AmazonRDSPerformanceInsightsReadOnly 是 AWS 托管策略，可以授予对 Amazon RDS Performance Insights API 的所有只读操作的访问权限。

如果将 AmazonRDSPerformanceInsightsReadOnly 附加到权限集或角色，接收人可以使用 Performance Insights 以及其他控制台功能。

有关更多信息，请参阅 AWS 托管式策略：AmazonRDSPerformanceInsightsReadOnly。

将 AmazonRDSPerformanceInsightsFullAccess 策略附加到 IAM 主体

AmazonRDSPerformanceInsightsFullAccess 是 AWS 托管策略，可以授予对 Amazon RDS Performance Insights API 的所有操作的访问权限。

如果将 AmazonRDSPerformanceInsightsFullAccess 附加到权限集或角色，接收人可以使用 Performance Insights 以及其他控制台功能。

有关更多信息，请参阅 AWS 托管式策略：AmazonRDSPerformanceInsightsFullAccess。

为 Performance Insights 创建自定义 IAM 策略

对于没有 AmazonRDSPerformanceInsightsReadOnly 或 AmazonRDSPerformanceInsightsFullAccess 策略的用户，可以通过创建或修改用户托管式 IAM 策略，来授予对 Performance Insights 的访问权限。在将策略附加到 IAM 权限集或角色时，接收人可以使用 Performance Insights。

创建自定义策略

1. 打开 IAM 控制台：https://console.aws.amazon.com/iam/。
2. 在导航窗格中，选择策略。
3. 选择创建策略。
4. 在创建策略页面上，选择 JSON 选项。
5. 复制并粘贴《AWS 托管策略参考指南》中的 JSON 策略文档部分提供的有关 AmazonRDSPerformanceInsightsReadOnly 或 AmazonRDSPerformanceInsightsFullAccess 策略的文本。
6. 选择查看策略。
7. 为策略提供名称并可以选择提供描述，然后选择创建策略。

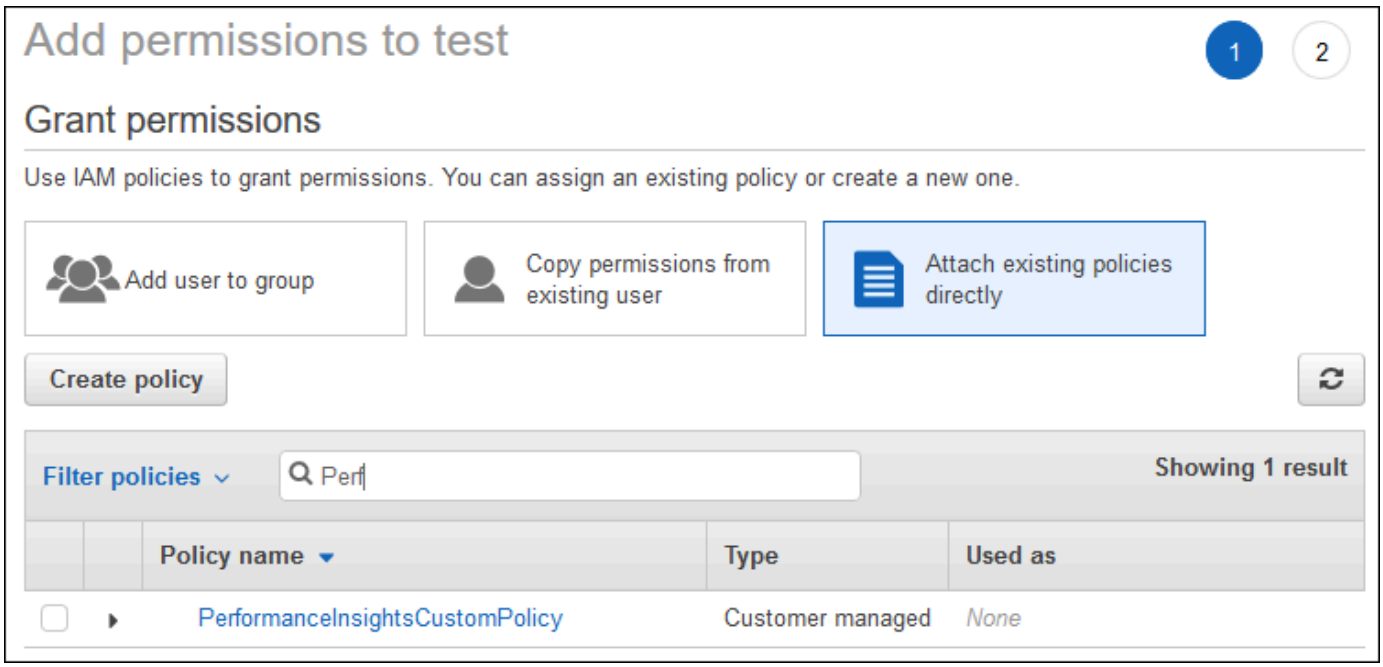
现在，可以将策略附加到权限集或角色。以下过程假设您已经有一个可用于此目的的用户。

将策略附加到用户

1. 打开 IAM 控制台：https://console.aws.amazon.com/iam/。
2. 在导航窗格中，选择 Users。
3. 从列表中选择现有用户。

> **Important**
>
> 要使用 Performance Insights，请确保除了自定义策略之外，您还有权访问 Amazon RDS。例如，AmazonRDSPerformanceInsightsReadOnly 预定义策略提供了对 Amazon RDS 的只读访问权限。有关更多信息，请参阅“使用策略管理访问”。

4. 在 Summary (摘要) 页上，选择 Add permissions (添加权限)。
5. 选择直接附加现有策略。对于搜索，键入策略名称的前几个字符，如下图所示。



6. 选择策略，然后选择 Next: Review。
7. 选择 Add permissions (添加权限)。

更改性能详情的 AWS KMS 策略

Performance Insights 使用 AWS KMS key 加密敏感数据。通过 API 或控制台启用 Performance Insights 时，您可以执行以下任一操作：

- 选择默认 AWS 托管式密钥。

  Amazon RDS 为新的数据库实例使用 AWS 托管式密钥。Amazon RDS 将为您的 AWS 账户创建 AWS 托管式密钥。您的 AWS 账户在每个 AWS 区域都有用于 Amazon RDS 的不同 AWS 托管式密钥。

- 选择客户托管密钥。

  如果您指定一个客户托管密钥，则您账户中调用 Performance Insights API 的用户需要在 KMS 密钥具有 `kms:Decrypt` 和 `kms:GenerateDataKey` 权限。您可以通过 IAM policy 配置这些权限。但是，我们建议您通过 KMS 密钥策略来管理这些权限。有关更多信息，请参阅《AWS Key Management Service 开发人员指南》中的在 AWS KMS 中使用密钥策略。

Example

以下示例显示了如何将语句添加到 KMS 密钥策略。这些语句可以访问 Performance Insights。您可能需要更改一些限制，这取决于您使用 KMS 密钥的方式。在将语句添加到您的策略之前，请删除所有注释。

```
{
"Version" : "2012-10-17",
 "Id" : "your-policy",
 "Statement" : [ {
    //This represents a statement that currently exists in your policy.
 }
 ....,
 //Starting here, add new statement to your policy for Performance Insights.
 //We recommend that you add one new statement for every RDS instance
{
    "Sid" : "Allow viewing RDS Performance Insights",
    "Effect": "Allow",
    "Principal": {
        "AWS": [
            //One or more principals allowed to access Performance Insights
            "arn:aws:iam::444455556666:role/Role1"
        ]
    },
    "Action": [
        "kms:Decrypt",
        "kms:GenerateDataKey"
    ],
    "Resource": "*",
    "Condition" : {
        "StringEquals" : {
            //Restrict access to only RDS APIs (including Performance Insights).
            //Replace region with your AWS Region.
            //For example, specify us-west-2.
            "kms:ViaService" : "rds.region.amazonaws.com"
        },
        "ForAnyValue:StringEquals": {
            //Restrict access to only data encrypted by Performance Insights.
            "kms:EncryptionContext:aws:pi:service": "rds",
            "kms:EncryptionContext:service": "pi",

            //Restrict access to a specific RDS instance.
            //The value is a DbiResourceId.
```

```
            "kms:EncryptionContext:aws:rds:db-id": "db-AAAAABBBBBCCCCDDDDDEEEEE"
        }
    }
}
```

Performance Insights 如何使用 AWS KMS 客户托管密钥

Performance Insights 使用客户托管密钥加密敏感数据。当您开启 Performance Insights 时，可以通过 API 提供 AWS KMS 密钥。Performance Insights 对此密钥创建 KMS 权限。它使用密钥并执行必要的操作来处理敏感数据。敏感数据包括用户、数据库、应用程序和 SQL 查询文本等字段。Performance Insights 可确保数据在静态和动态时都保持加密状态。

Performance Insights IAM 如何使用 AWS KMS

IAM 提供针对特定 API 的权限。Performance Insights 具有以下公有 API，您可以使用 IAM 策略对其进行限制：

- `DescribeDimensionKeys`
- `GetDimensionKeyDetails`
- `GetResourceMetadata`
- `GetResourceMetrics`
- `ListAvailableResourceDimensions`
- `ListAvailableResourceMetrics`

您可以使用以下 API 请求来获取敏感数据。

- `DescribeDimensionKeys`
- `GetDimensionKeyDetails`
- `GetResourceMetrics`

当您使用 API 获取敏感数据时，Performance Insights 会利用调用方的凭证。此检查可确保敏感数据的访问权限仅限于有权访问 KMS 密钥的用户。

调用这些 API 时，您需要通过 IAM 策略调用 API 的权限，以及通过 AWS KMS 密钥策略调用 `kms:decrypt` 操作的权限。

`GetResourceMetrics` API 可以返回敏感和非敏感数据。请求参数决定响应是否应包含敏感数据。当请求的筛选条件或分组依据参数中包含敏感维度时，API 会返回敏感数据。

有关可以与 GetResourceMetrics API 一起使用的维度的更多信息，请参阅 DimensionGroup。

Example 示例

以下示例请求 db.user 组的敏感数据：

```
POST / HTTP/1.1
Host: <Hostname>
Accept-Encoding: identity
X-Amz-Target: PerformanceInsightsv20180227.GetResourceMetrics
Content-Type: application/x-amz-json-1.1
User-Agent: <UserAgentString>
X-Amz-Date: <Date>
Authorization: AWS4-HMAC-SHA256 Credential=<Credential>, SignedHeaders=<Headers>,
 Signature=<Signature>
Content-Length: <PayloadSizeBytes>
{
  "ServiceType": "RDS",
  "Identifier": "db-ABC1DEFGHIJKL2MNOPQRSTUV3W",
  "MetricQueries": [
    {
      "Metric": "db.load.avg",
      "GroupBy": {
        "Group": "db.user",
        "Limit": 2
      }
    }
  ],
  "StartTime": 1693872000,
  "EndTime": 1694044800,
  "PeriodInSeconds": 86400
}
```

Example

以下示例请求 db.load.avg 指标的非敏感数据：

```
POST / HTTP/1.1
Host: <Hostname>
Accept-Encoding: identity
X-Amz-Target: PerformanceInsightsv20180227.GetResourceMetrics
```

```
Content-Type: application/x-amz-json-1.1
User-Agent: <UserAgentString>
X-Amz-Date: <Date>
Authorization: AWS4-HMAC-SHA256 Credential=<Credential>, SignedHeaders=<Headers>,
 Signature=<Signature>
Content-Length: <PayloadSizeBytes>
{
    "ServiceType": "RDS",
    "Identifier": "db-ABC1DEFGHIJKL2MNOPQRSTUV3W",
    "MetricQueries": [
        {
            "Metric": "db.load.avg"
        }
    ],
    "StartTime": 1693872000,
    "EndTime": 1694044800,
    "PeriodInSeconds": 86400
}
```

授予 Performance Insights 精细访问权限

精细访问控制功能提供用于控制对 Performance Insights 的访问的其他方法。此访问控制功能可以允许或拒绝对 `GetResourceMetrics`、`DescribeDimensionKeys` 和 `GetDimensionKeyDetails` Performance Insights 操作的各个维度的访问。要使用精细访问功能，请使用条件键在 IAM 策略中指定维度。访问权限的评估遵循 IAM 策略评估逻辑。有关更多信息，请参阅《 IAM 用户指南》中的策略评估逻辑。如果 IAM 策略声明未指定任何维度，该声明将控制对指定操作的所有维度的访问。有关可用维度的列表，请参阅 DimensionGroup。

要找出您的凭证有权访问的维度，请使用 `ListAvailableResourceDimensions` 中的 `AuthorizedActions` 参数并指定操作。允许的 `AuthorizedActions` 值如下所示：

- `GetResourceMetrics`
- `DescribeDimensionKeys`
- `GetDimensionKeyDetails`

例如，如果您为 `AuthorizedActions` 参数指定 `GetResourceMetrics`，`ListAvailableResourceDimensions` 将返回 `GetResourceMetrics` 操作有权访问的维度列表。如果您在 `AuthorizedActions` 参数中指定了多个操作，`ListAvailableResourceDimensions` 将返回这些操作有权访问的维度的交集。

Example

以下示例提供了对 GetResourceMetrics 和 DescribeDimensionKeys 操作的指定维度的访问权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowToDiscoverDimensions",
            "Effect": "Allow",
            "Action": [
                "pi:ListAvailableResourceDimensions"
            ],
            "Resource": [
                "arn:aws:pi:us-east-1:123456789012:metrics/rds/db-
ABC1DEFGHIJKL2MNOPQRSTUV3W"
            ]
        },
        {
            "Sid": "SingleAllow",
            "Effect": "Allow",
            "Action": [
                "pi:GetResourceMetrics",
                "pi:DescribeDimensionKeys"
            ],
            "Resource": [
                "arn:aws:pi:us-east-1:123456789012:metrics/rds/db-
ABC1DEFGHIJKL2MNOPQRSTUV3W"
            ],
            "Condition": {
                "ForAllValues:StringEquals": {
                    // only these dimensions are allowed. Dimensions not included in
                    // a policy with "Allow" effect will be denied
                    "pi:Dimensions": [
                        "db.sql_tokenized.id",
                        "db.sql_tokenized.statement"
                    ]
                }
            }
        }


    ]
```

```
}
```

以下是对所请求维度的响应：

```
 // ListAvailableResourceDimensions API
// Request
{
    "ServiceType": "RDS",
    "Identifier": "db-ABC1DEFGHIJKL2MNOPQRSTUV3W",
    "Metrics": [ "db.load" ],
    "AuthorizedActions": ["DescribeDimensionKeys"]
}

// Response
{
    "MetricDimensions": [ {
        "Metric": "db.load",
        "Groups": [
            {
                "Group": "db.sql_tokenized",
                "Dimensions": [
                    { "Identifier": "db.sql_tokenized.id" },
                  //  { "Identifier": "db.sql_tokenized.db_id" }, // not included
 because not allows in the IAM Policy
                    { "Identifier": "db.sql_tokenized.statement" }
                ]
            }

        ] }
    ]
}
```

以下示例为维度指定一个允许访问和两个拒绝访问权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
          {
            "Sid": "AllowToDiscoverDimensions",
            "Effect": "Allow",
            "Action": [
```

```
                "pi:ListAvailableResourceDimensions"
            ],
            "Resource": [
                "arn:aws:pi:us-east-1:123456789012:metrics/rds/db-
ABC1DEFGHIJKL2MNOPQRSTUV3W"
            ]
          },

          {
            "Sid": "O01AllowAllWithoutSpecifyingDimensions",
            "Effect": "Allow",
            "Action": [
                "pi:GetResourceMetrics",
                "pi:DescribeDimensionKeys"
            ],
            "Resource": [
                "arn:aws:pi:us-east-1:123456789012:metrics/rds/db-
ABC1DEFGHIJKL2MNOPQRSTUV3W"
            ]
        },

        {
            "Sid": "O01DenyAppDimensionForAll",
            "Effect": "Deny",
            "Action": [
                "pi:GetResourceMetrics",
                "pi:DescribeDimensionKeys"
            ],
            "Resource": [
                "arn:aws:pi:us-east-1:123456789012:metrics/rds/db-
ABC1DEFGHIJKL2MNOPQRSTUV3W"
            ],
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "pi:Dimensions": [
                        "db.application.name"
                    ]
                }
            }
        },

        {
            "Sid": "O01DenySQLForGetResourceMetrics",
            "Effect": "Deny",
```

```
            "Action": [
                "pi:GetResourceMetrics"
            ],
            "Resource": [
                "arn:aws:pi:us-east-1:123456789012:metrics/rds/db-
ABC1DEFGHIJKL2MNOPQRSTUV3W"
            ],
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "pi:Dimensions": [
                        "db.sql_tokenized.statement"
                    ]
                }
            }
        }
    ]
}
```

以下是对请求维度的响应：

```
   // ListAvailableResourceDimensions API
// Request
{
    "ServiceType": "RDS",
    "Identifier": "db-ABC1DEFGHIJKL2MNOPQRSTUV3W",
    "Metrics": [ "db.load" ],
    "AuthorizedActions": ["GetResourceMetrics"]
}

// Response
{
    "MetricDimensions": [ {
        "Metric": "db.load",
        "Groups": [
            {
                "Group": "db.application",
                "Dimensions": [

                  // removed from response because denied by the IAM Policy
                  //  { "Identifier": "db.application.name" }
                ]
```

```
            },
            {
                "Group": "db.sql_tokenized",
                "Dimensions": [
                    { "Identifier": "db.sql_tokenized.id" },
                    { "Identifier": "db.sql_tokenized.db_id" },

                  // removed from response because denied by the IAM Policy
                  //  { "Identifier": "db.sql_tokenized.statement" }
                ]
            },
            ...
        ] }
    ]
}
```

```
// ListAvailableResourceDimensions API
// Request
{
    "ServiceType": "RDS",
    "Identifier": "db-ABC1DEFGHIJKL2MNOPQRSTUV3W",
    "Metrics": [ "db.load" ],
    "AuthorizedActions": ["DescribeDimensionKeys"]
}

// Response
{
    "MetricDimensions": [ {
        "Metric": "db.load",
        "Groups": [
            {
                "Group": "db.application",
                "Dimensions": [
                  // removed from response because denied by the IAM Policy
                  //  { "Identifier": "db.application.name" }
                ]
            },
            {
                "Group": "db.sql_tokenized",
                "Dimensions": [
                    { "Identifier": "db.sql_tokenized.id" },
                    { "Identifier": "db.sql_tokenized.db_id" },
```

```
                // allowed for DescribeDimensionKeys because our IAM Policy
                // denies it only for GetResourceMetrics
                  { "Identifier": "db.sql_tokenized.statement" }
              ]
            },
            ...
        ] }
    ]
}
```

使用 Performance Insights 控制面板分析指标

> **Important**
>
> AWS 已宣布性能详情的生命周期终止日期为：2025 年 11 月 30 日。在此日期之后，Amazon RDS 将不再支持性能详情控制台体验、灵活的保留期（1-24 个月）及其关联定价。性能详情 API 将继续存在，定价不变。性能详情 API 的费用将与 CloudWatch 数据库洞察的费用一起显示在您的 AWS 账单中。
>
> 我们建议您在 2025 年 11 月 30 日之前，将使用性能详情付费套餐的所有数据库集群升级到数据库洞察的高级模式。有关升级到数据库洞察的高级模式的信息，请参阅开启适用于 Amazon Aurora 的数据库洞察的高级模式。
>
> 如果您不采取任何行动，则使用性能详情的数据库集群将默认为使用数据库洞察的标准模式。在数据库洞察的标准模式下，您可能会在 7 天后无法访问性能数据历史记录，并且可能无法在 Amazon RDS 控制台中使用执行计划和按需分析功能。在 2025 年 11 月 30 日之后，只有数据库洞察的高级模式将支持执行计划和按需分析。
>
> 借助 CloudWatch 数据库洞察，您可以监控数据库实例集的数据库负载，并对性能进行大规模分析和故障排除。有关数据库洞察的更多信息，请参阅使用 CloudWatch 数据库洞察监控 Amazon Aurora 数据库。有关定价信息，请参阅 Amazon CloudWatch 定价。

Performance Insights 控制面板包含帮助您分析和排查性能问题的数据库性能信息。在主控制面板页面上，可以查看有关数据库负载的信息。您可以按维度（例如等待事件或 SQL）对数据库负载进行“切片”。

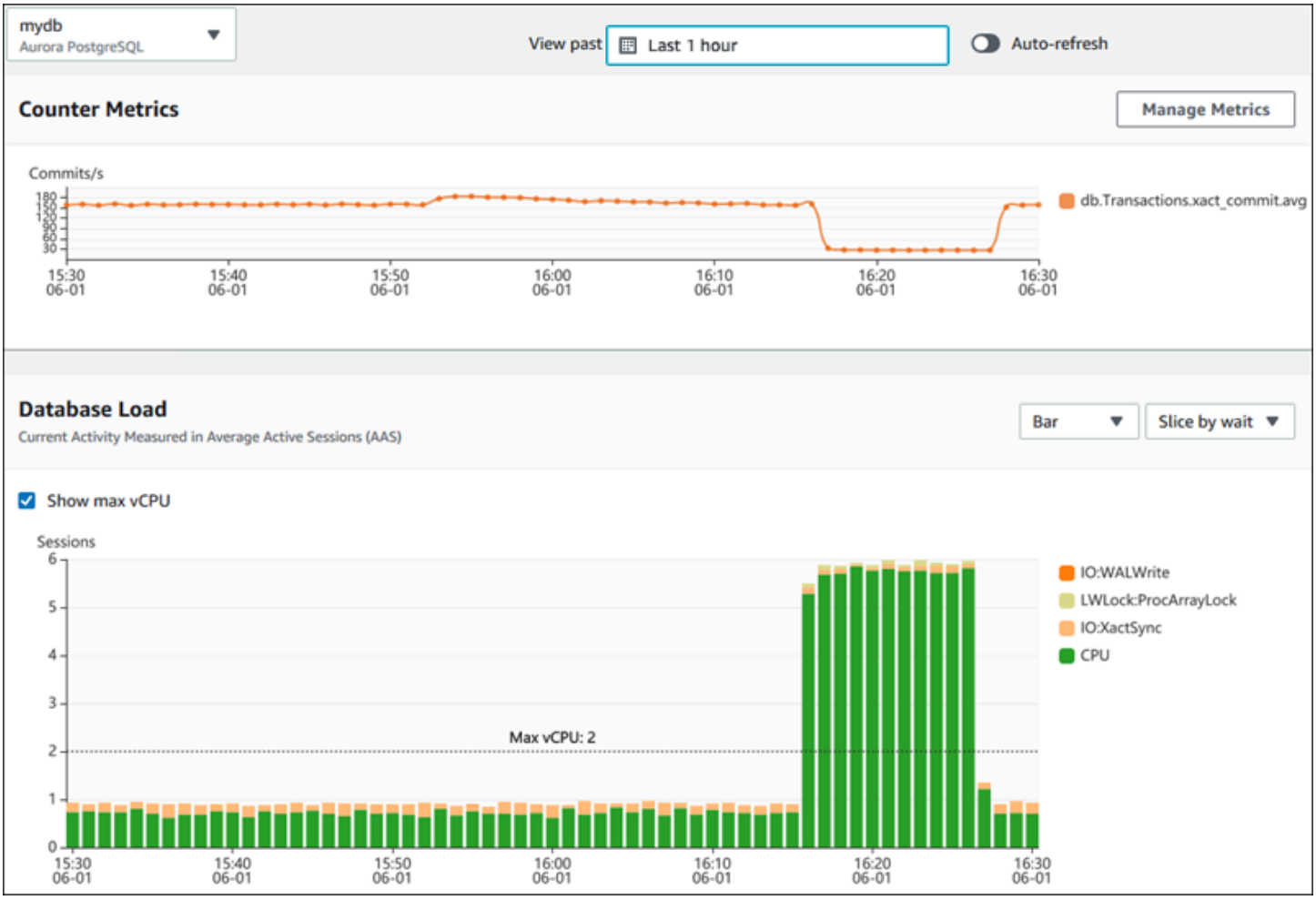
性能详情控制面板

- 性能详情控制面板概览
- 访问性能详情控制面板

- 按等待事件分析数据库负载
- 分析一段时间内的数据库性能
- 使用性能详情中的“主要 SQL”选项卡分析查询

性能详情控制面板概览

与性能详情进行交互的最简单方式即为控制面板。以下示例显示了 PostgreSQL 数据库实例的控制面板。



主题

- 时间范围筛选器
- 计数器指标图表
- 数据库负载图表
- 主要维度表

时间范围筛选器

默认情况下，Performance Insights 控制面板将显示最近一小时的数据库负载。您可以将此范围调整为短至 5 分钟或长达 2 年。您也可以选择自定义相对范围。

您可以选择一个具有开始和结束日期和时间的绝对范围。以下示例显示的时间范围从 2024 年 9 月 25 日午夜开始，到 2024 年 9 月 28 日晚上 11:59 结束。

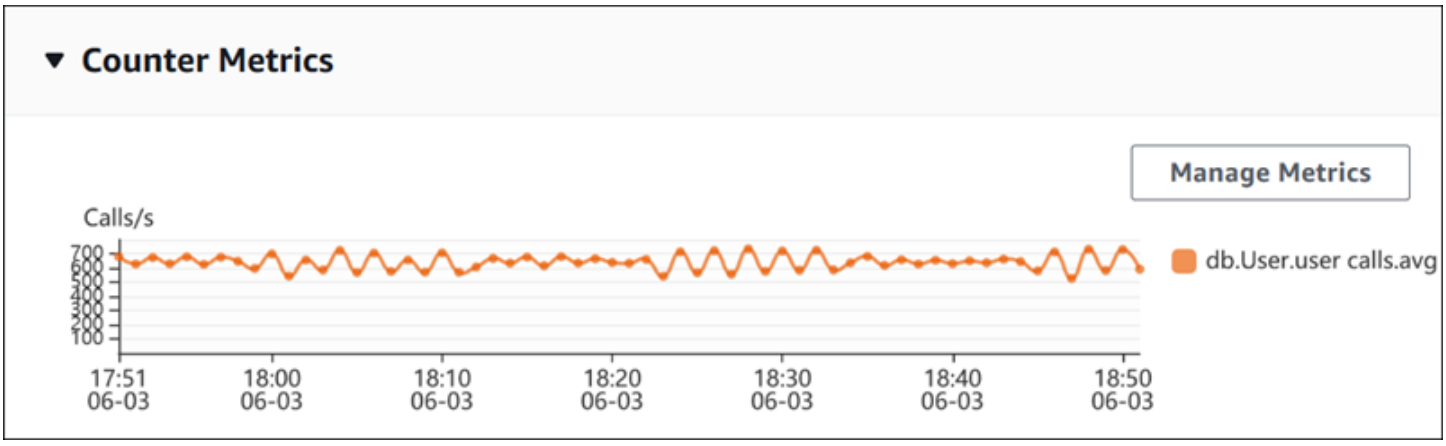
默认情况下，性能详情控制面板的时区为协调世界时（UTC）。也可以选择本地时区。

计数器指标图表

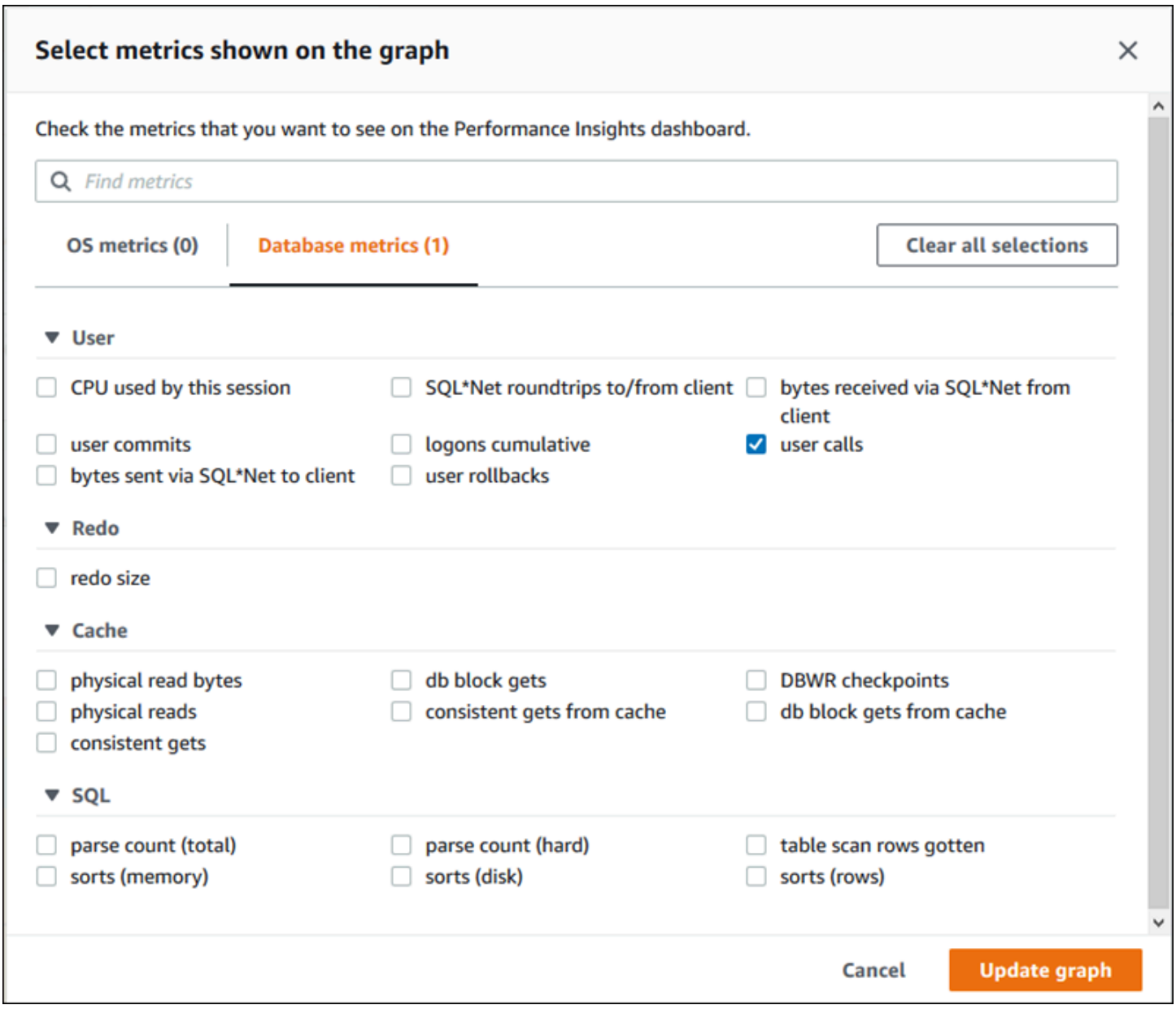
使用计数器指标，您可以自定义 Performance Insights 控制面板来包括最多 10 个其他图表。这些图表显示了所选的数十个操作系统和数据库性能指标。您可将此信息与数据库负载相关联，以帮助识别和分析性能问题。

计数器指标图表显示了性能计数器的数据。原定设置指标取决于数据库引擎：

- Aurora MySQL – `db.SQL.Innodb_rows_read.avg`
- Aurora PostgreSQL – `db.Transactions.xact_commit.avg`



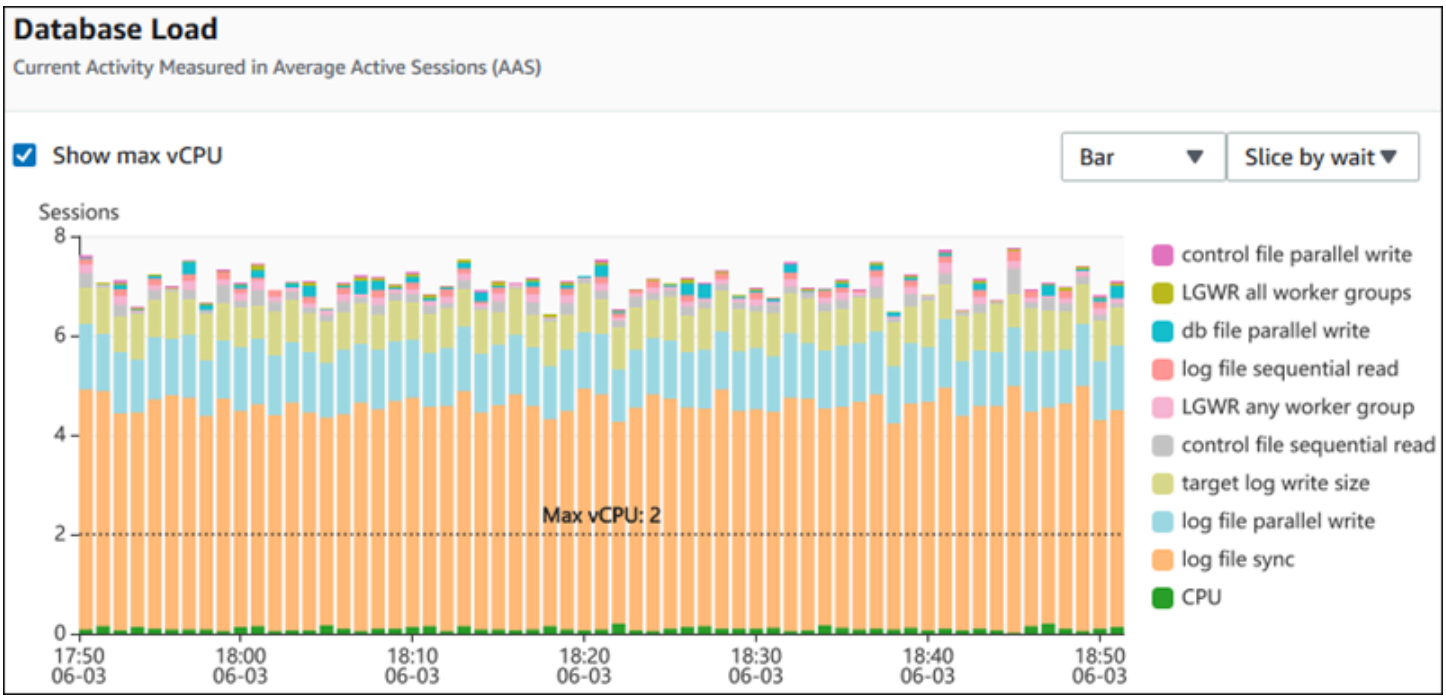
要更改性能计数器，请选择 Manage Metrics（管理指标）。您可以选择多个操作系统指标或数据库指标，如以下屏幕截图所示。要查看任何指标的详细信息，请将鼠标悬停在相应指标名称上。



有关可以为每个数据库引擎添加的计数器指标的说明，请参阅 性能详情计数器指标。

数据库负载图表

数据库负载图表显示数据库负载与最大 vCPU 线表示的数据库实例容量的比较情况。预设情况下，堆叠折线图将以每单位时间的平均活动会话数表示数据库负载。数据库负载按等待状态进行切片（分组）。

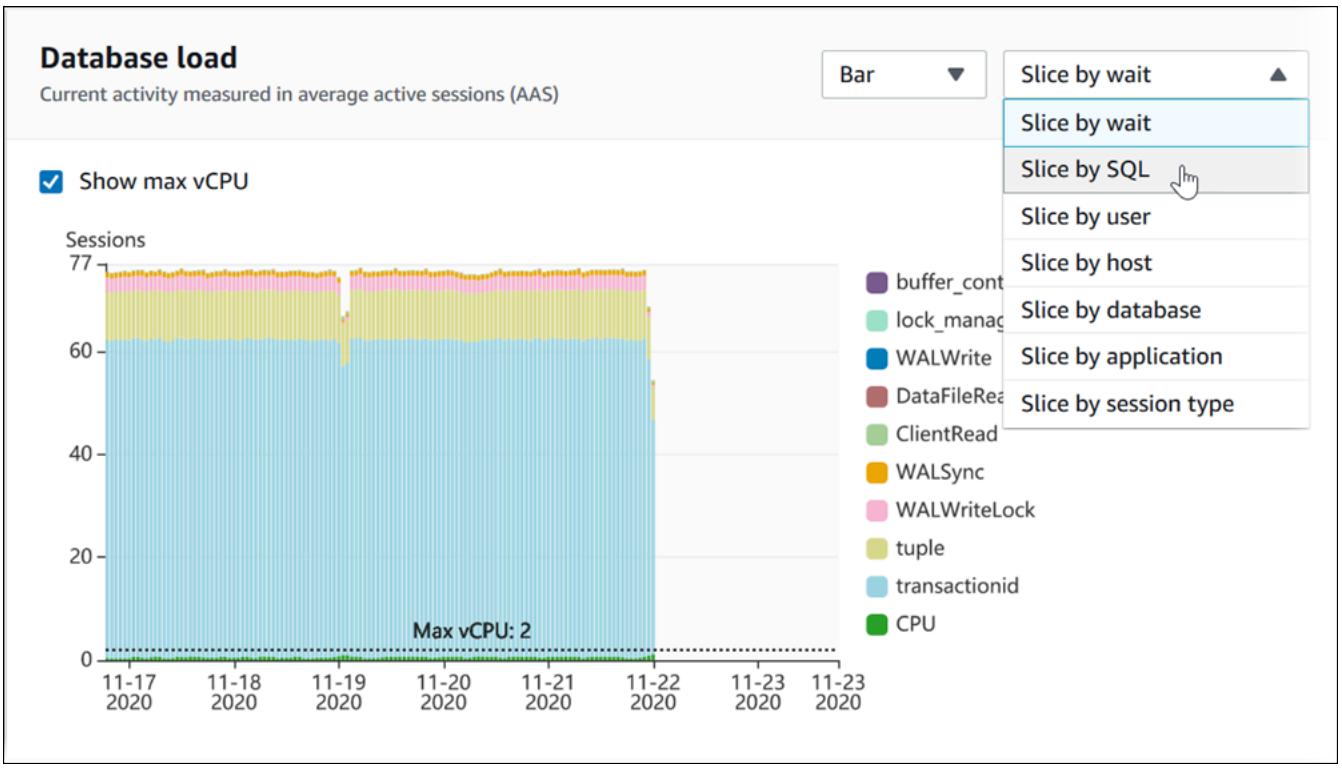


按维度切片的数据库负载

您可以选择按任何受支持维度分组的活动会话显示负载。下表显示了不同引擎支持的维度。

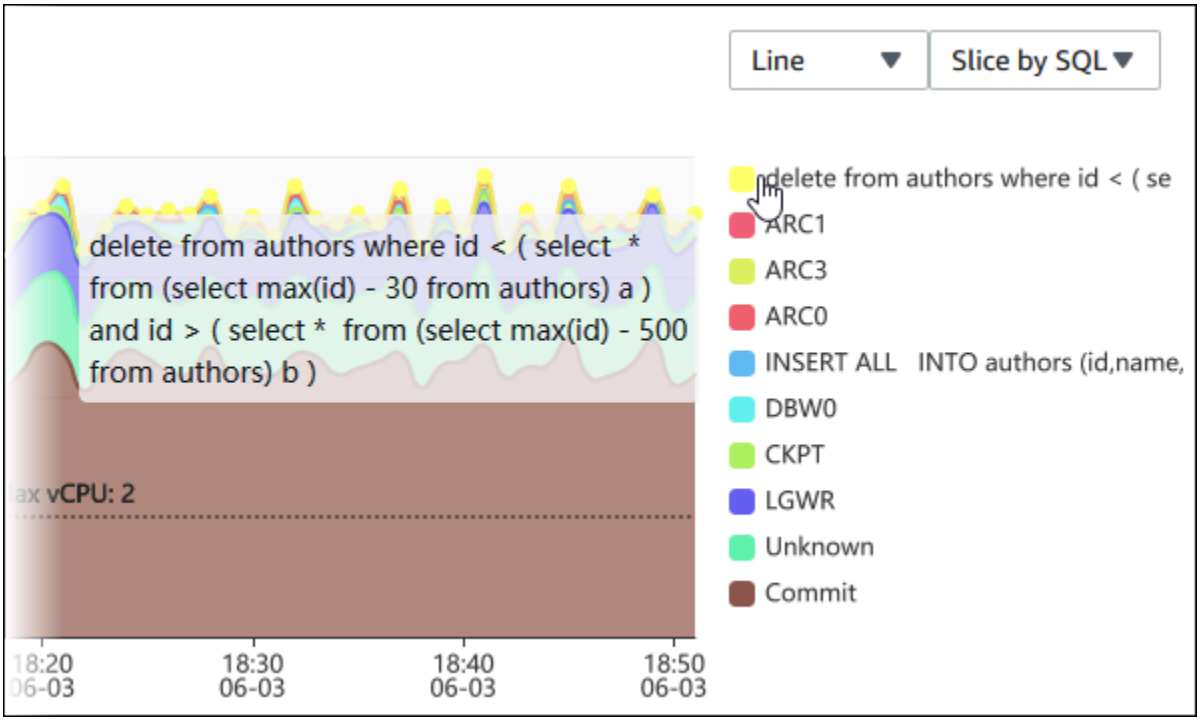
维度	Aurora PostgreSQL	Aurora MySQL
Host	是	是
SQL	是	是
User	是	是
等待	是	是
应用程序	是	否
数据库	是	是
会话类型	是	否

下图显示了 PostgreSQL 数据库实例的维度。

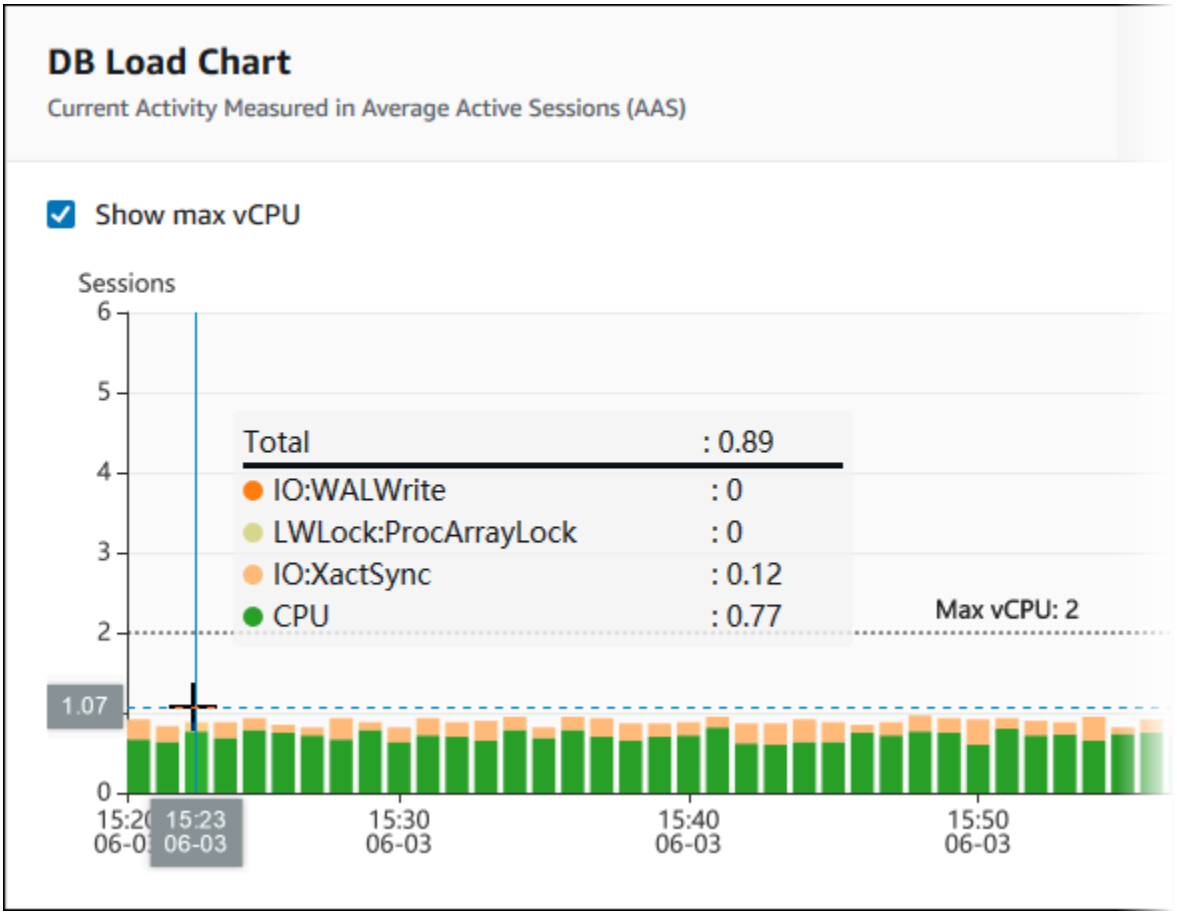


维度项目的数据库负载详细信息

要查看维度中数据库负载项目的详细信息，请将光标悬停在相应项目名称上。下图显示了 SQL 语句的详细信息。

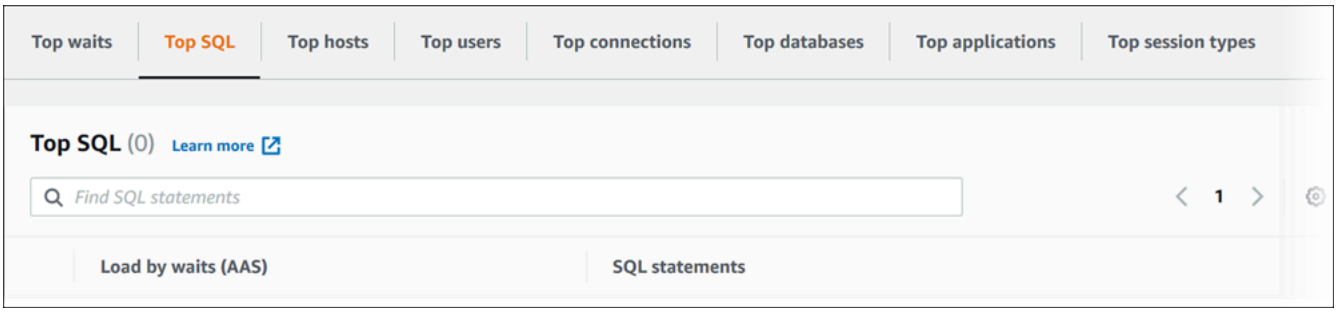


要在图例中查看任何项目在选定时间段内的详细信息，请将鼠标悬停在相应项目上。



主要维度表

主要维度表将按不同的维度切割数据库负载。维度是数据库负载不同特征的类别或“切片依据”。如果维度为 SQL，则 Top SQL（主要 SQL）显示了对数据库负载影响最大的 SQL 语句。



请选择以下任何一个维度选项卡。

选项卡	描述	支持的引擎
主要 SQL	当前正在运行的 SQL 语句	全部
主要等待	数据库后端正在等待的事件	全部
主要主机	所连接客户端的主机名	全部
主要用户	登录到数据库的用户	全部
主要应用程序	连接到数据库的应用程序的名称	仅限 Aurora PostgreSQL
主要会话类型	当前会话的类型	仅限 Aurora PostgreSQL

要了解如何使用 Top SQL（主要 SQL）选项卡分析查询，请参阅 “Top SQL”（主要 SQL）选项卡概览。

访问性能详情控制面板

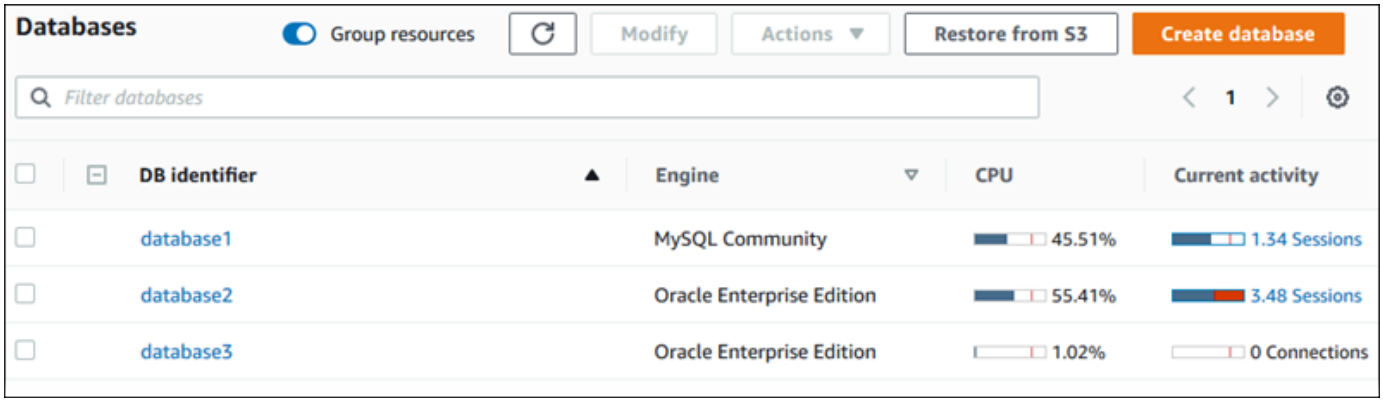
Amazon RDS 在性能详情控制面板中提供性能详情和 CloudWatch 指标的合并视图。

要访问性能详情控制面板，请使用以下过程。

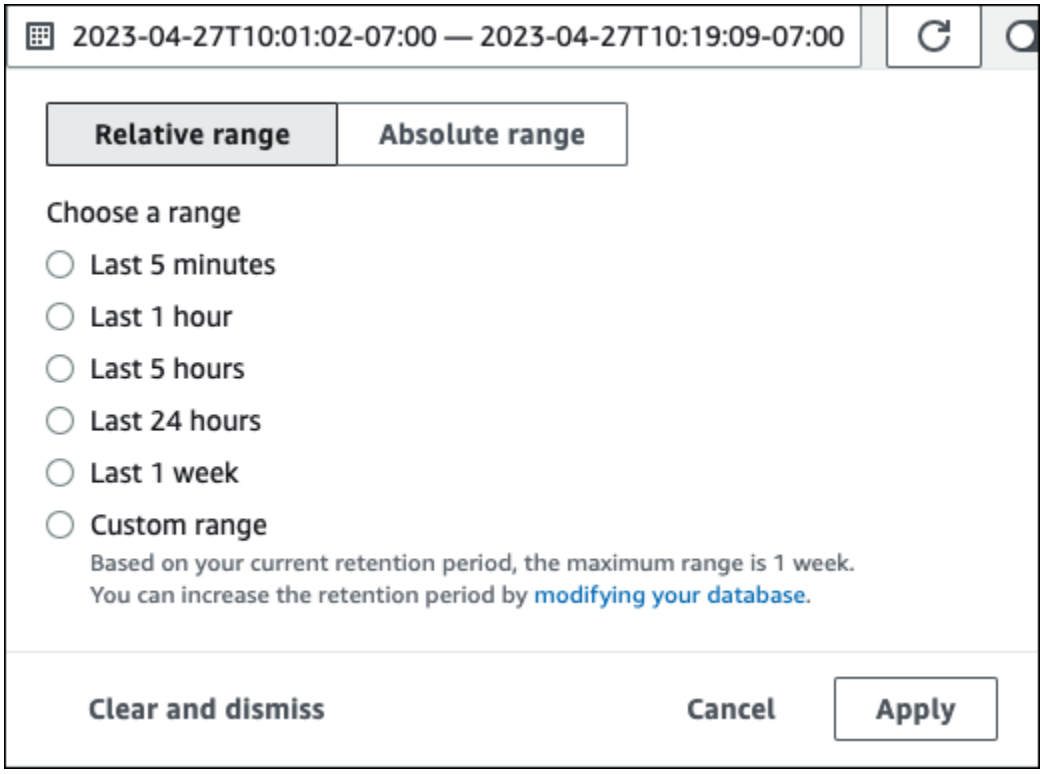
在 AWS 管理控制台中查看性能详情控制面板

1. 通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在左侧导航窗格中，选择性能详情。
3. 选择一个数据库实例。

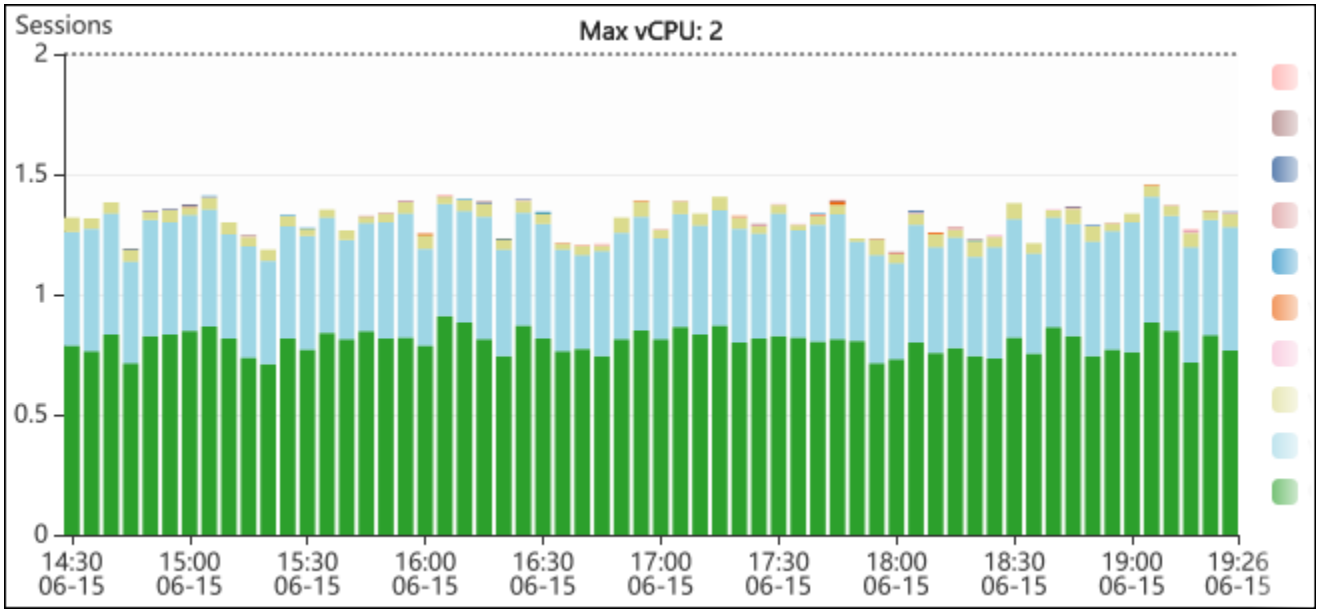
   对于已开启性能详情的数据库实例，还可以通过选择数据库实例列表中的会话项目来访问控制面板。在当前活动下，会话项目显示在过去五分钟内平均活跃会话中的数据库负载。条形图显示负载量。当条形图为空时，数据库实例处于空闲状态。随着负载的增加，条形图会以蓝色填充。当负载超过数据库实例类上的虚拟 CPU (vCPU) 数量时，条形图变为红色，表示可能出现瓶颈。



4. （可选）在右上角选择日期或时间范围，并指定不同的相对或绝对时间间隔。现在，您可以指定时间段，并生成数据库性能分析报告。该报告提供了已发现的见解和建议。有关更多信息，请参阅在性能详情中创建性能分析报告。

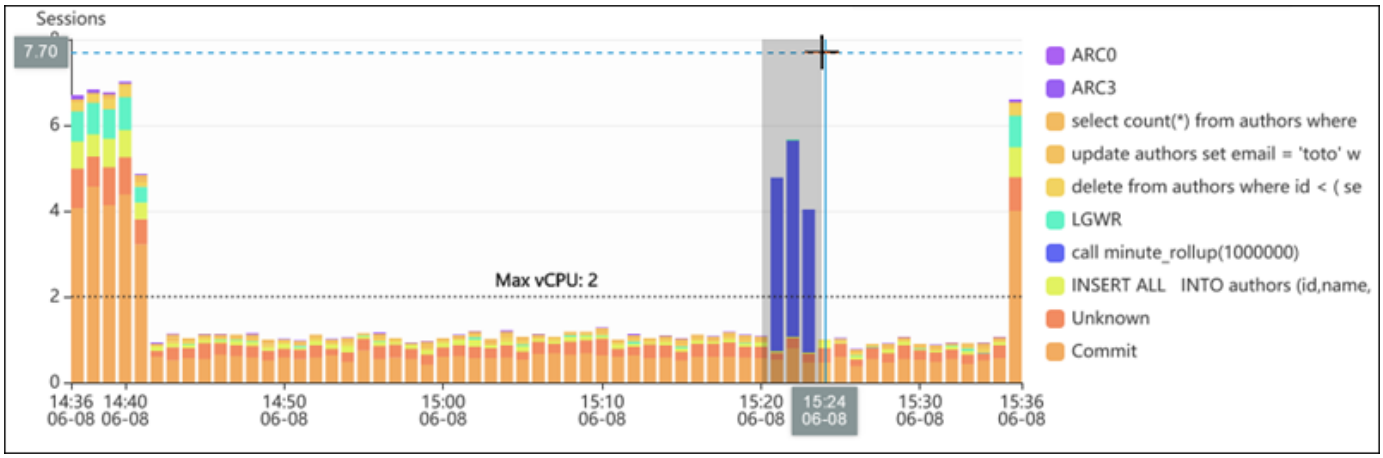


在以下屏幕截图中，数据库负载间隔为 5 小时。

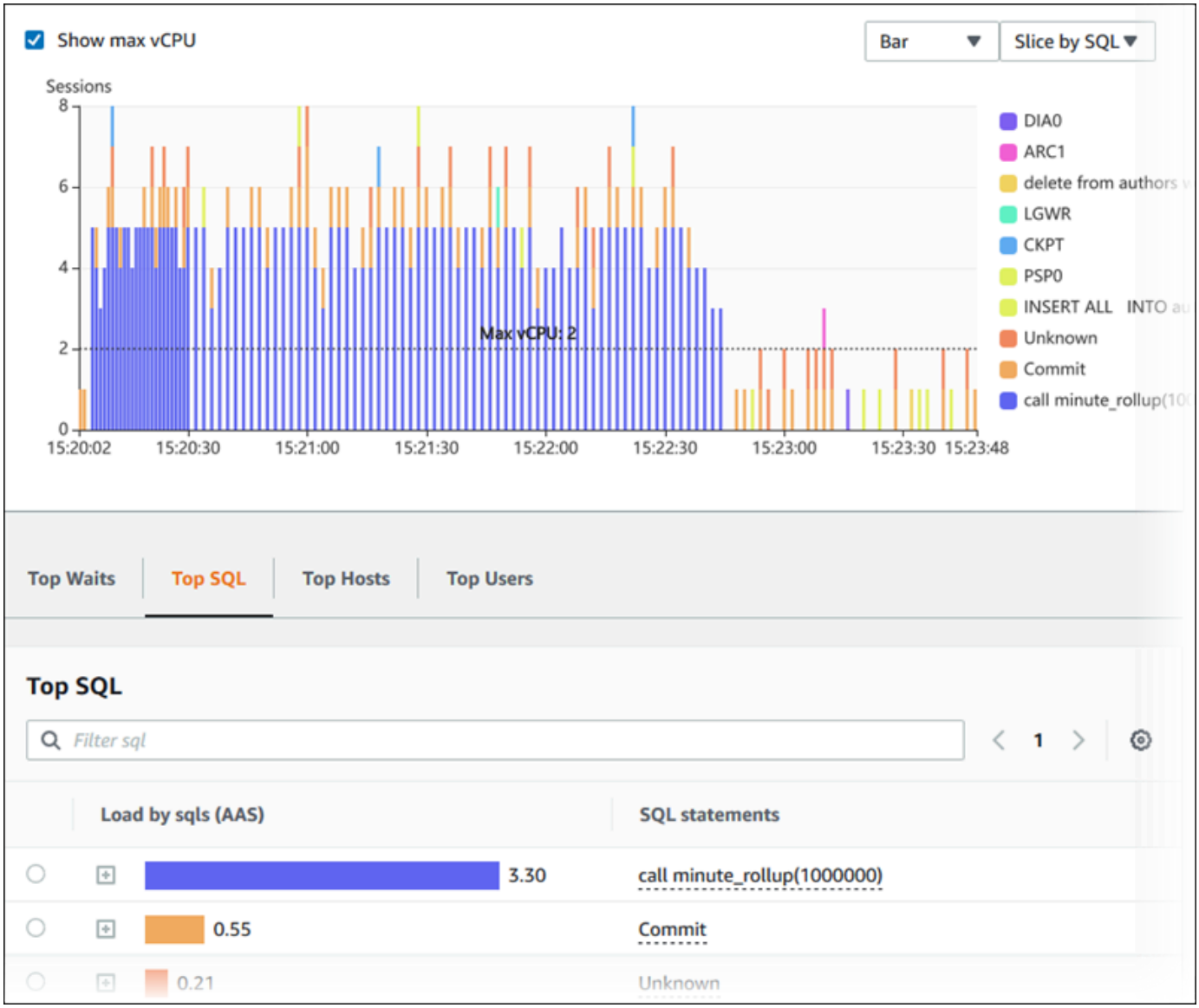


5. （可选）要放大数据库负载图表的一部分，请选择开始时间并拖动到所需时间段的结尾。

   所选区域将在数据库负载图表中突出显示。



   释放鼠标时，数据库负载图表上的所选 AWS将放大，并重新计算 Top dimensions（热门维度）表。



6. （可选）要自动刷新数据，请选择自动刷新。



性能详情控制面板自动用新数据进行刷新。刷新速率取决于所显示的数据量：

- 5 分钟则每 10 秒刷新一次。
- 1 小时则每 5 分钟刷新一次。
- 5 小时则每 5 分钟刷新一次。
- 24 小时则每 30 分钟刷新一次。
- 1 周则每天刷新一次。

- 1 个月则每天刷新一次。

按等待事件分析数据库负载

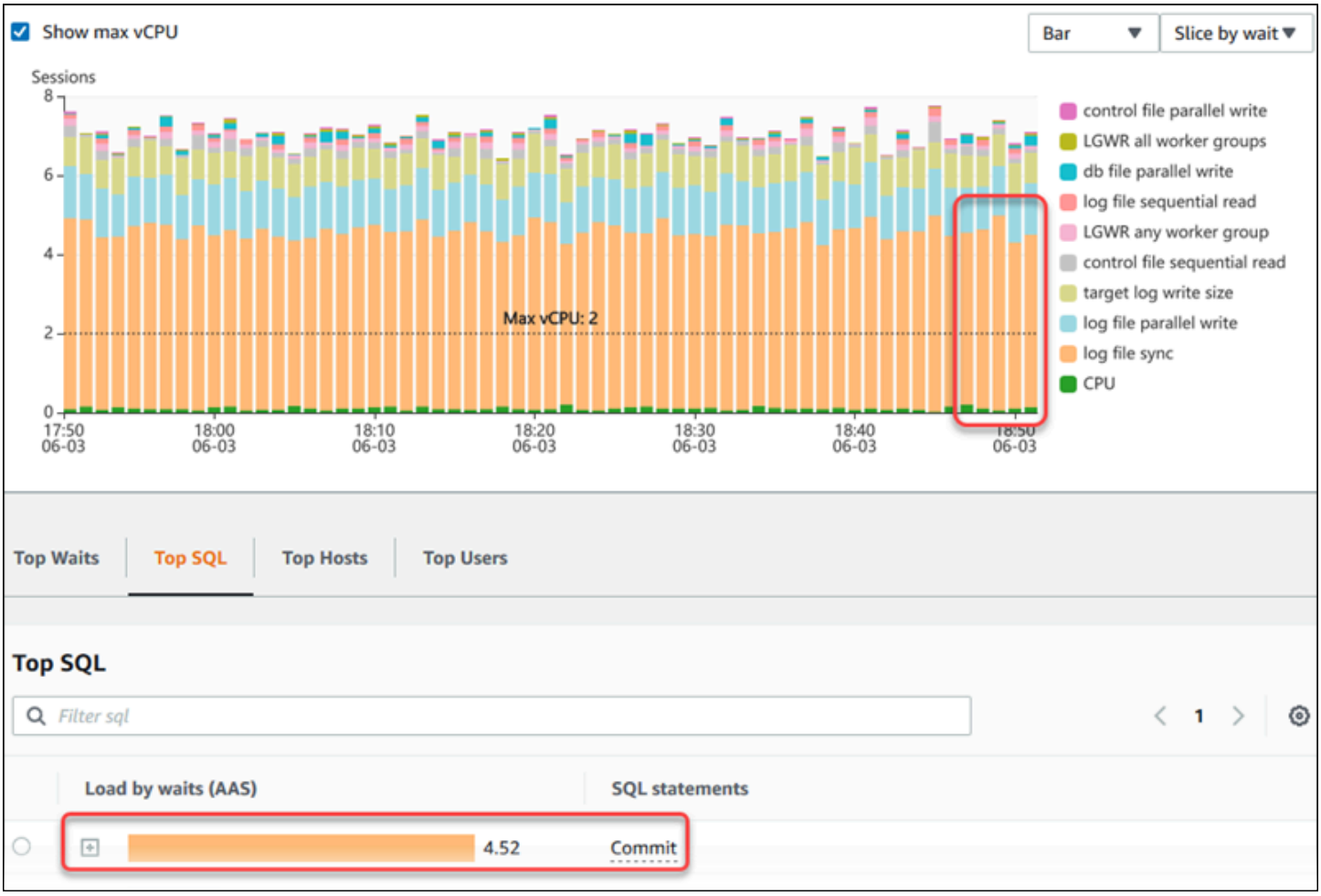
如果数据库负载图表显示了一个瓶颈，您可以找出负载的来源。为此，请查看数据库负载图表下方的主要负载项目。选择特定项目 (如 SQL 查询或用户) 以深入了解该项目并查看有关该项目的详细信息。

按等待状态和主要 SQL 查询分组的数据库负载是默认 Performance Insights 控制面板视图，此组合通常提供了最多的性能问题见解。按等待状态分组的数据库负载显示了数据库中是否存在任何资源瓶颈或并发瓶颈。在这种情况下，“Top Load Items”表的 SQL 选项卡显示了增大该负载的查询。

诊断性能问题的典型工作流程如下：

1. 查看数据库负载图表并了解是否存在数据库负载的事件越过了 Max CPU 线。
2. 如果有，请查看数据库负载图表并确定负主要责任的等待状态。
3. 通过以下方式确定导致负载的摘要查询：查看“Top Load Items”表上的 SQL 选项卡中的哪个查询对于导致这些等待状态所起的作用最大。可通过 DB Load by Wait (按等待状态排列的数据库负载) 列加以识别。
4. 在 SQL 选项卡中选择这些摘要查询之一以展开它并查看它包含的子查询。

例如，在下面的控制面板中，日志文件同步等待状态占大部分数据库负载。LGWR 所有工作线程组等待状态也很高。主要 SQL 图表显示导致日志文件同步等待状态的内容：频繁的 `COMMIT` 语句。在这种情况下，降低提交频率将会减少数据库负载。



分析一段时间内的数据库性能

通过创建一段时间的性能分析报告，可以通过按需分析来分析数据库性能。可以查看性能分析报告，发现任何性能问题，例如资源瓶颈或数据库实例中的查询更改。Performance Insights 控制面板支持您选择时间段并创建性能分析报告。您还可以向报告添加一个或多个标签。

要使用此功能，您必须使用付费套餐保留期。有关更多信息，请参阅 性能详情的定价和数据留存

该报告可供在性能分析报告 - 新增选项卡中进行选择和查看。该报告包含见解、相关指标和解决性能问题的建议。在 Performance Insights 保留期内，该报告可供查看。

如果报告分析时段的开始时间在保留期之外，则报告将被删除。您也可以在保留期结束之前删除报告。

要检测性能问题并为数据库实例生成分析报告，必须开启 Performance Insights。有关开启 Performance Insights 的更多信息，请参阅为 Aurora 打开和关闭性能详情。

有关此功能的区域、数据库引擎和实例类支持信息，请参阅支持性能详情功能的 Amazon Aurora 数据库引擎、区域和实例类

在以下各节中，您可以创建、查看、删除性能分析报告和向其中添加标签。

主题

- 在性能详情中创建性能分析报告
- 在性能详情中查看性能分析报告
- 在性能详情中向性能分析报告添加标签
- 在性能详情中删除性能分析报告

在性能详情中创建性能分析报告

您可以在 Performance Insights 控制面板中创建特定时段的性能分析报告。您可以选择一个时间段并将一个或多个标签添加到分析报告中。

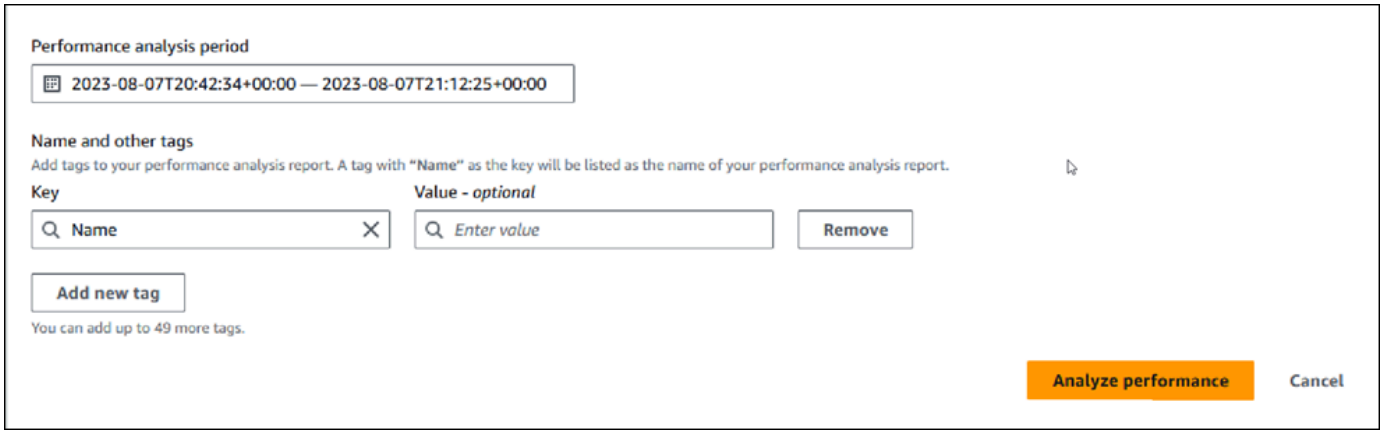
分析时段从 5 分钟到 6 天不等。在分析开始之前，必须有至少 24 小时的性能数据。

有关此功能的区域、数据库引擎和实例类支持信息，请参阅支持性能详情功能的 Amazon Aurora 数据库引擎、区域和实例类

创建一段时间内的性能分析报告

1. 通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在左侧导航窗格中，选择性能详情。
3. 选择一个数据库实例。
4. 在性能详情控制面板上的数据库负载部分中选择分析性能。

   将显示用于设置时间段和向性能分析报告中添加一个或多个标签的字段。



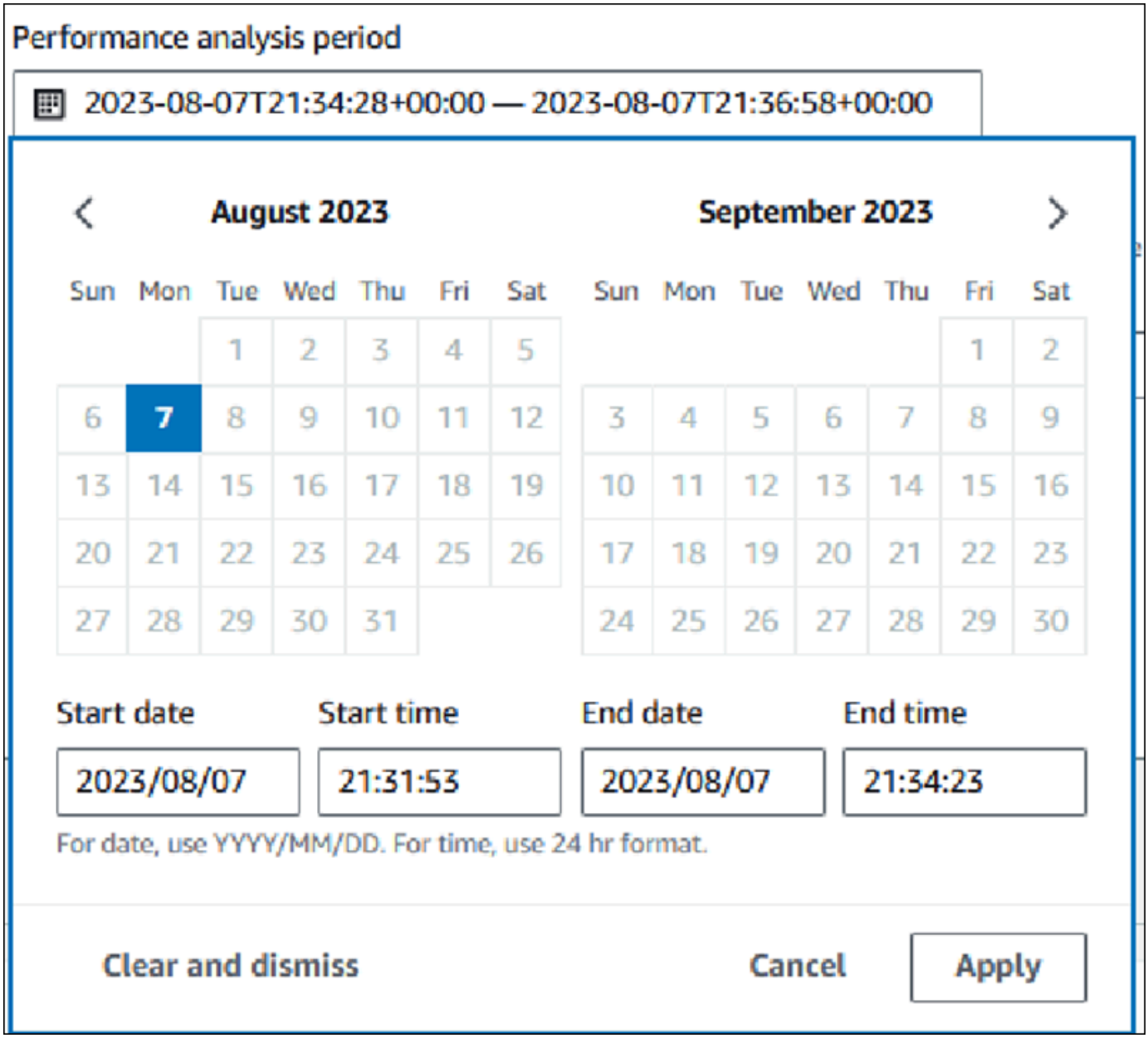
5. 选择时间段。如果在右上角的相对范围或绝对范围中设置时间段，您只能输入或选择该时间段内的分析报告日期和时间。如果您选择该时间段之外的分析时段，则会显示一条错误消息。

要设置时间段，您可以执行以下任一操作：

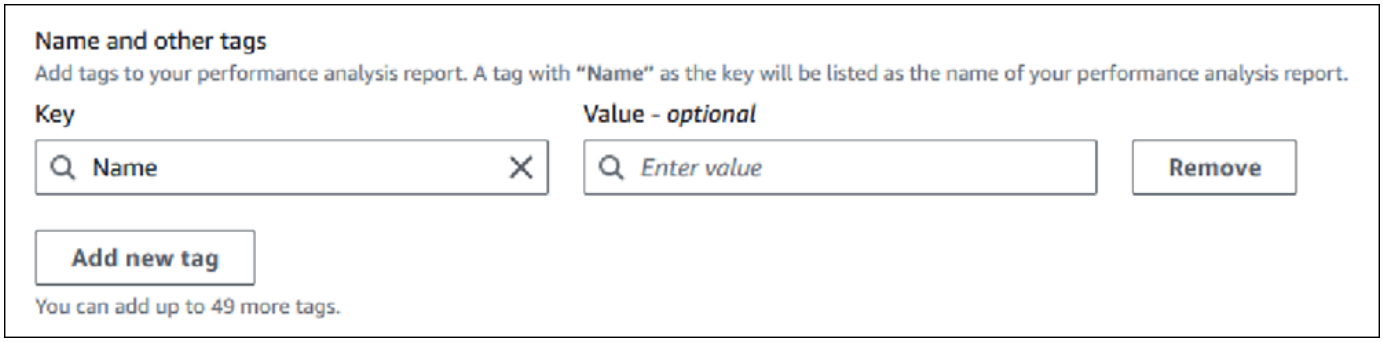
- 按下并拖动数据库负载图表上的任何滑块。

  性能分析时段框显示选定的时间段，数据库负载图表突出显示选定的时间段。

- 在性能分析时段框中，选择开始日期、开始时间、结束日期以及结束时间。



6. （可选）输入键和值 - 可选，以便为报告添加标签。



7. 选择分析性能。

   无论报告生成是成功还是失败，横幅都会显示一条消息。该消息还提供了查看报告的链接。

   以下示例显示了带有报告创建成功消息的横幅。



   该报告可供在性能分析报告 - 新增选项卡中查看。

您可以使用 AWS CLI 创建性能分析报告。有关如何使用 AWS CLI 创建报告的示例，请参阅创建一段时间的性能分析报告。

在性能详情中查看性能分析报告

性能分析报告 - 新增选项卡列出了为数据库实例创建的所有报告。对于每个报告，显示以下内容：

- ID：报告的唯一标识符。
- 名称：添加到报告中的标签键。
- 报告创建时间：您创建报告的时间。
- 分析开始时间：报告中分析的开始时间。
- 分析结束时间：报告中分析的结束时间。

查看性能分析报告

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在左侧导航窗格中，选择 Performance Insights。
3. 选择要查看其分析报告的数据库实例。

4. 在性能详情控制面板中，向下滚动并选择性能分析报告 - 新增选项卡。

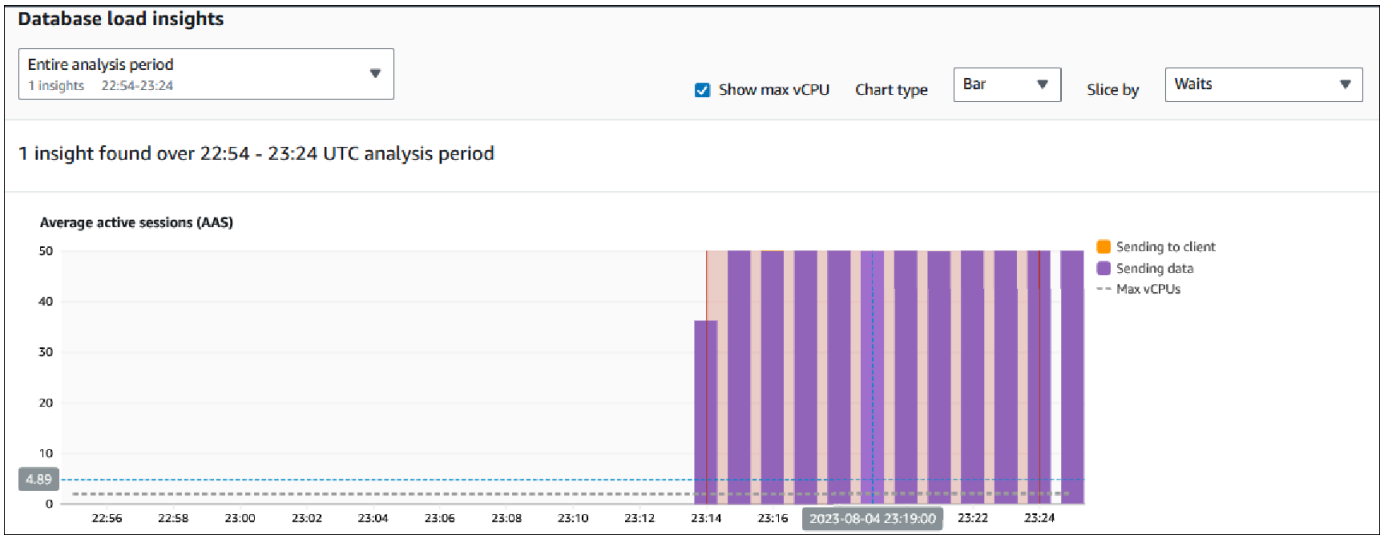
   将显示不同时间段的所有分析报告。

5. 选择您要查看的报告的 ID。

   如果确定了多个见解，则原定设置情况下，数据库负载图表会显示整个分析时段。如果报告确定了一个见解，则原定设置情况下，数据库负载图表会显示该见解。

   控制面板还在标签部分中列出了报告的标签。

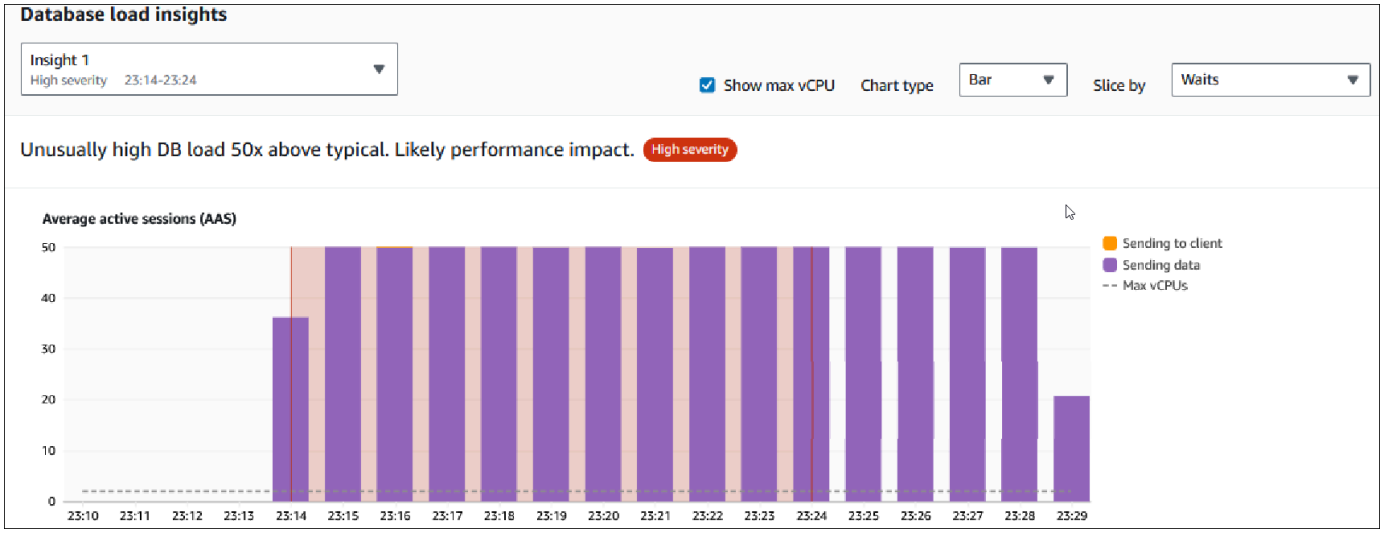
   以下示例显示了报告的整个分析时段。

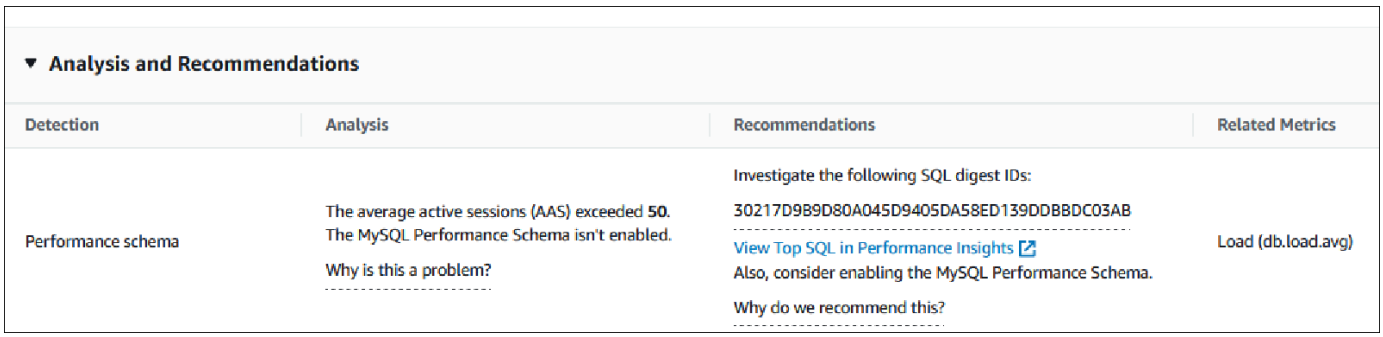


6. 如果在报告中发现多个见解，请在数据库负载见解列表中选择要查看的见解。

   控制面板显示见解消息、数据库负载图表（突出显示见解的时间段、分析和建议）以及报告标签列表。

   以下示例显示了报告中的数据库负载见解。





在性能详情中向性能分析报告添加标签

您可以在创建或查看报告时添加标签。您最多可以为一个报告添加 50 个标签。

您需要权限才能添加标签。有关 Performance Insights 的访问策略的更多信息，请参阅为 Performance Insights 配置访问策略。

要在创建报告时添加一个或多个标签，请参阅过程在性能详情中创建性能分析报告中的步骤 6。

在查看报告时添加一个或多个标签

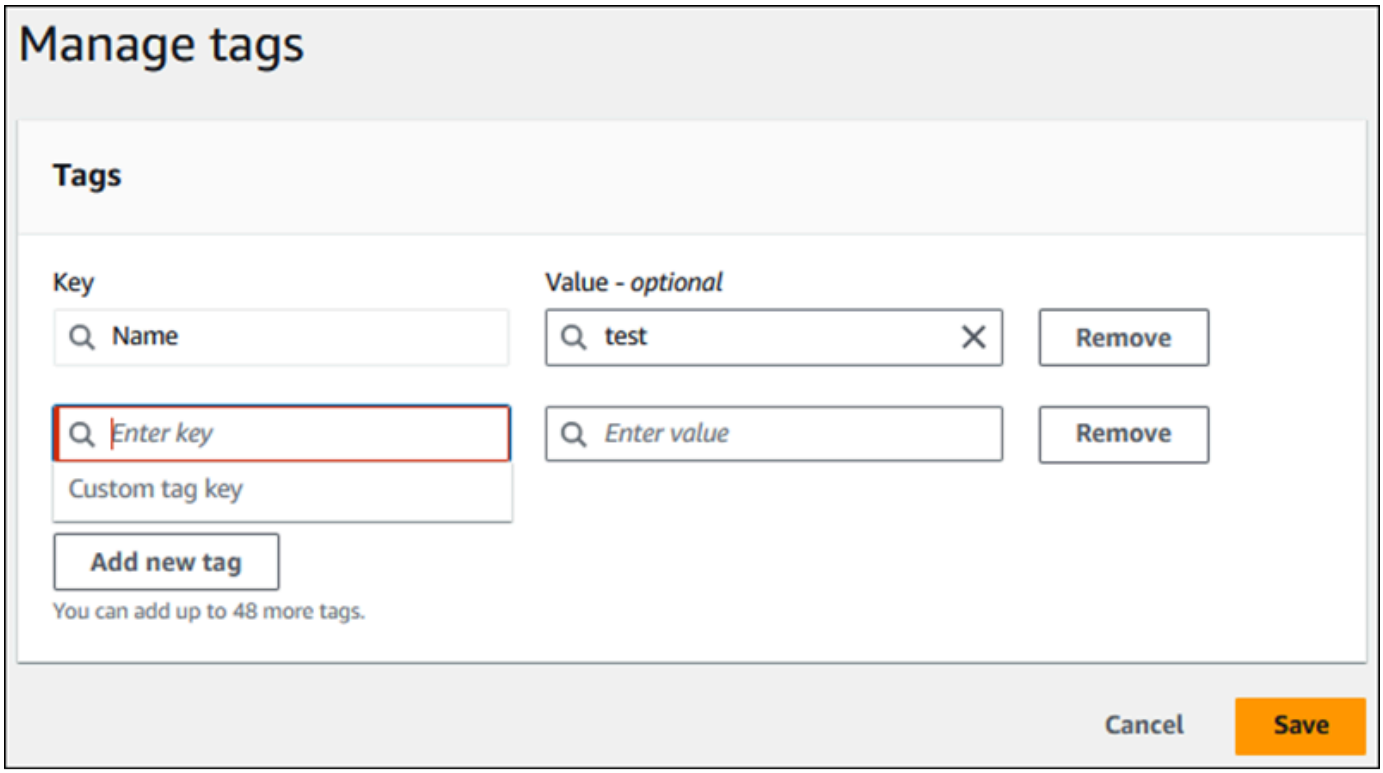
1. 通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在左侧导航窗格中，选择 Performance Insights。
3. 选择一个数据库实例。

   将显示该数据库实例的 Performance Insights 控制面板。
4. 向下滚动并选择性能分析报告 - 新增选项卡。
5. 选择要为其添加标签的报告。

控制面板显示报告。

6. 向下滚动到标签，然后选择管理标签。
7. 选择添加新标签。
8. 输入键和值 - 可选，然后选择添加新标签。

以下示例提供了为所选报告添加新标签的选项。



为报告创建了一个新标签。

报告的标签列表显示在控制面板上的标签部分。如果您想从报告中删除标签，请选择标签旁边的移除。

在性能详情中删除性能分析报告

您可以从性能分析报告选项卡中显示的报告列表中删除报告，也可以在查看报告时删除报告。

删除报告

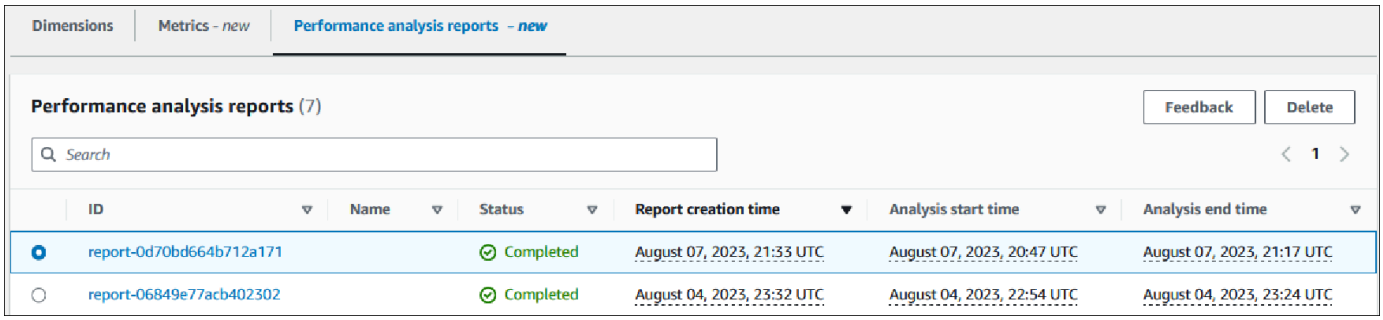
1. 通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在左侧导航窗格中，选择 Performance Insights。

3. 选择一个数据库实例。

   将显示该数据库实例的 Performance Insights 控制面板。

4. 向下滚动并选择性能分析报告 - 新增选项卡。

5. 选择要删除的报告，然后选择右上角的删除。



   将显示一个确认窗口。选择确认后，将删除报告。

6. （可选）选择您要删除的报告的 ID。

   在报告页面中，选择右上角的删除。

   将显示一个确认窗口。选择确认后，将删除报告。

使用性能详情中的“主要 SQL”选项卡分析查询

在 Amazon RDS Performance Insights 控制面板中，您可以在 Top dimensions（主要维度）表中的 Top SQL（主要 SQL）选项卡下找到有关运行中查询和最近查询的信息。您可以使用此信息来优化查询。

主题

- “Top SQL”（主要 SQL）选项卡概览
- 在 Performance Insights 控制面板中访问更多 SQL 文本
- 在 Performance Insights 控制面板中查看 SQL 统计数据

“Top SQL”（主要 SQL）选项卡概览

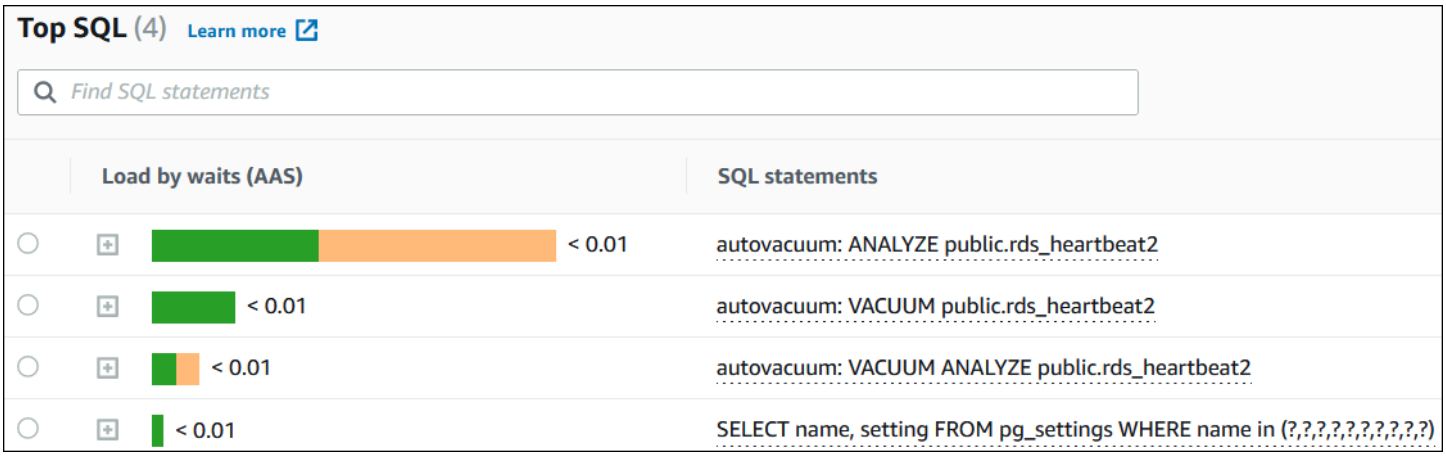
原定设置情况下，Top SQL（主要 SQL）选项卡将显示对数据库负载影响最大的 25 个 SQL 查询。为了帮助优化查询，您可以分析查询文本和 SQL 统计数据等信息。您还可以选择想要显示在 Top SQL（主要 SQL）选项卡中的统计数据。

主题

- SQL 文本
- SQL 统计数据
- 按等待状态排列的负载 (AAS)
- 查看 SQL 信息
- 选择统计数据首选项

SQL 文本

原定设置情况下，Top SQL（主要 SQL）表中的每行为每条语句显示 500 字节的文本。

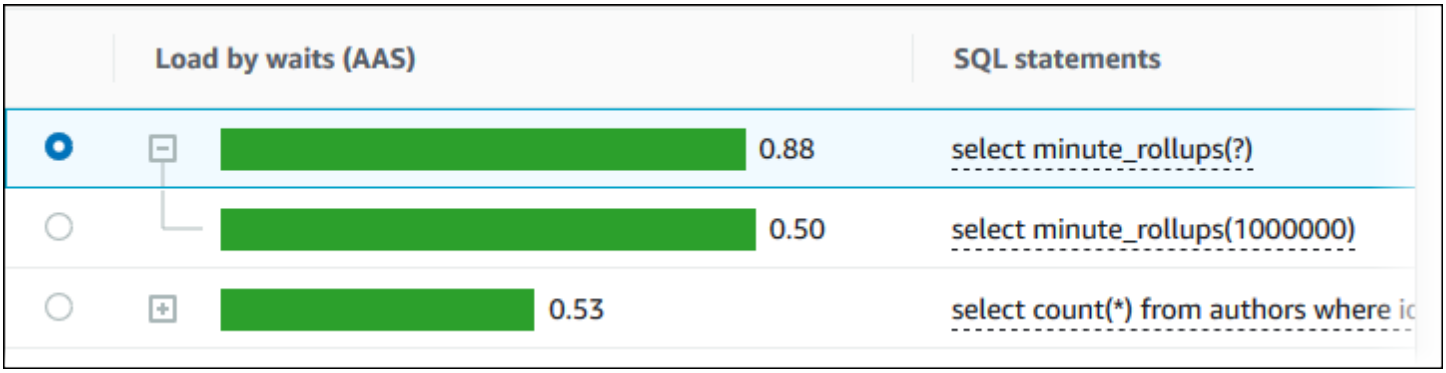


要了解如何查看超过默认 500 字节的 SQL 文本，请参阅在 Performance Insights 控制面板中访问更多 SQL 文本。

SQL 摘要是多个结构上相似但可能具有不同文本值的实际查询的组合。摘要用问号替换硬编码值。例如，摘要可能为 `SELECT * FROM emp WHERE lname= ?`。此摘要可能包含以下子查询：

```
SELECT * FROM emp WHERE lname = 'Sanchez'
SELECT * FROM emp WHERE lname = 'Olagappan'
SELECT * FROM emp WHERE lname = 'Wu'
```

要查看摘要中的文字 SQL 语句，请选择查询，然后选择加号 (+)。在以下示例中，所选查询为摘要。

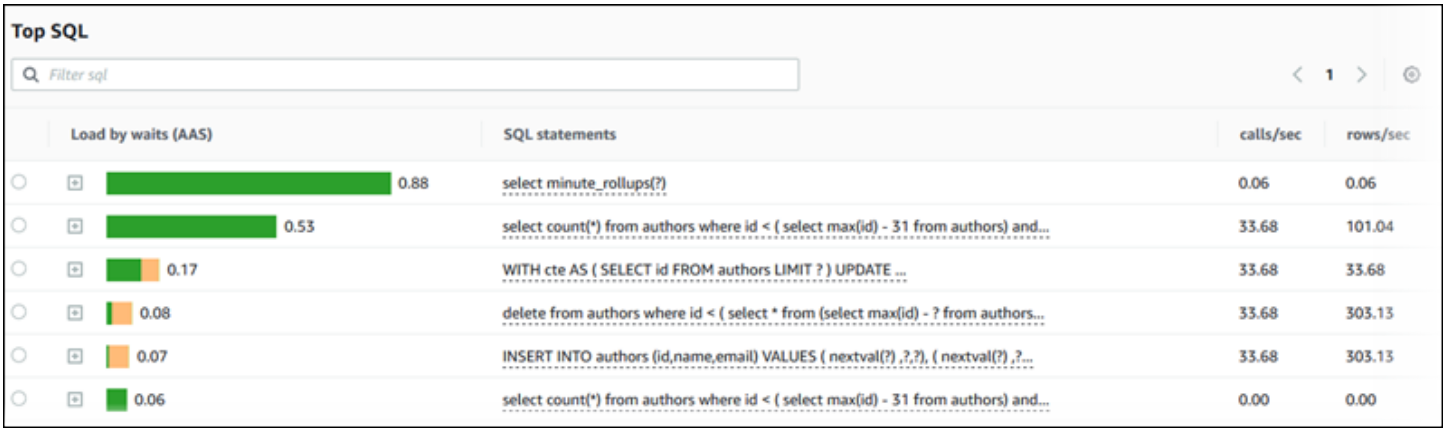


> **Note**
>
> SQL 摘要将相似的 SQL 语句进行分组，但不会编辑敏感信息。

SQL 统计数据

SQL 统计数据是关于 SQL 查询的性能相关指标。例如，性能详情可能会显示每秒执行数或每秒处理的行数。性能详情仅收集最常见查询的统计数据。通常，它们与 Performance Insights 控制面板中显示的按负载列出的主要查询匹配。

Top SQL（主要 SQL）表中的每一行显示了 SQL 语句或摘要的相关统计数据，如以下示例所示。



Performance Insights 可以报告 SQL 统计数据的 `0.00` 和 `-`（未知）。这种情况在以下条件下发生：

- 只存在一个样本。例如，Performance Insights 根据 `pg_stat_statements` 视图的多个样本，计算 Aurora PostgreSQL 的更改速率。当工作负载运行时间较短时，Performance Insights 可能只收集一个样本，这意味着它无法计算更改速率。未知值用短划线 (`-`) 表示。
- 两个样本具有相同的值。由于未发生更改，Performance Insights 无法计算更改速率，因此它将速率报告为 `0.00`。

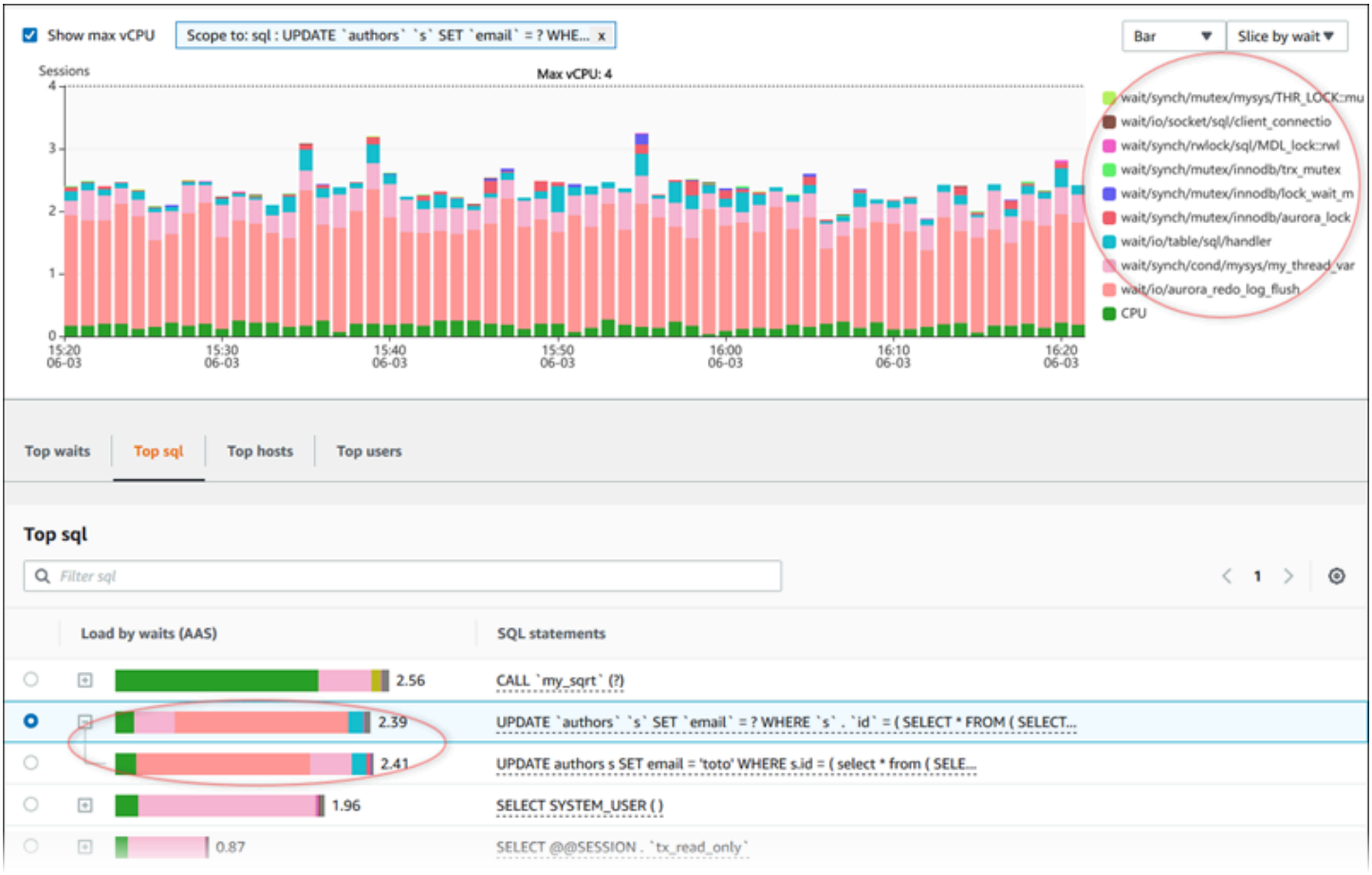
- Aurora PostgreSQL 语句缺乏有效的标识符。PostgreSQL 仅在解析和分析之后才会为语句创建标识符。因此，语句可以存在于 PostgreSQL 内部内存结构中，而无需标识符。由于 Performance Insights 每秒对内部内存结构进行一次采样，因此低延迟查询可能只出现在单个样本中。如果查询标识符对该样本不可用，则 Performance Insights 无法将此语句与其统计数据进行关联。未知值用短划线 (-) 表示。

有关 Aurora 引擎的 SQL 统计数据的说明，请参阅 Performance Insights 的 SQL 统计数据。

按等待状态排列的负载 (AAS)

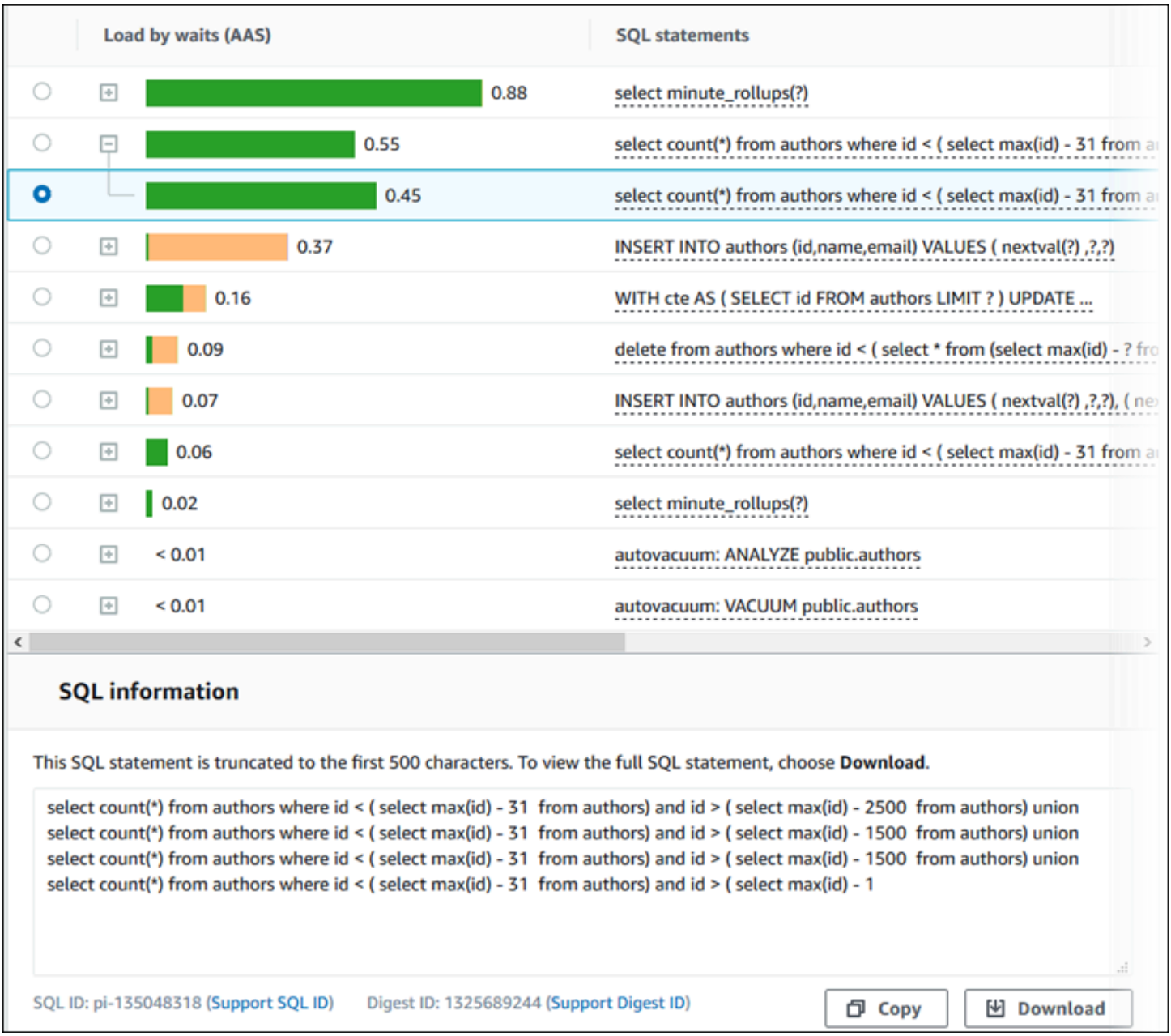
在 Top SQL（主要 SQL）中，按等待状态排列的负载 (AAS) 列说明了与每个主要负载项目关联的数据库负载的百分比。此列按当前在数据库负载图表中选择的分组方式反映该项目的负载。有关平均活动会话数（AAS）的更多信息，请参阅平均活动会话数。

例如，您可以按等待状态对数据库负载图表进行分组。您可以检查主要负载项目表中的 SQL 查询。在这种情况下，系统将对 DB Load by Waits (按等待状态排列的数据库负载) 栏进行大小调整、分段和颜色编码，以显示该查询在导致给定等待状态方面所起的作用大小，它还会显示哪些等待状态正在影响选定的查询。



查看 SQL 信息

在 Top SQL（主要 SQL）表中，您可以打开一条语句以查看其信息。信息将显示在底部窗格中。

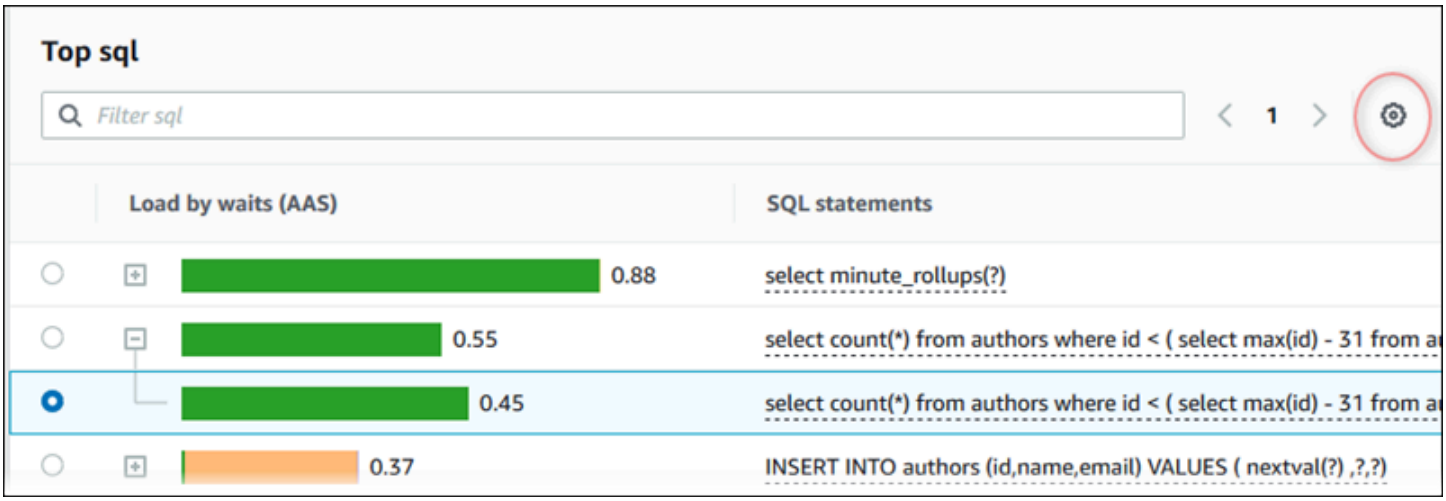


与 SQL 语句关联的以下类型的标识符 (ID)：

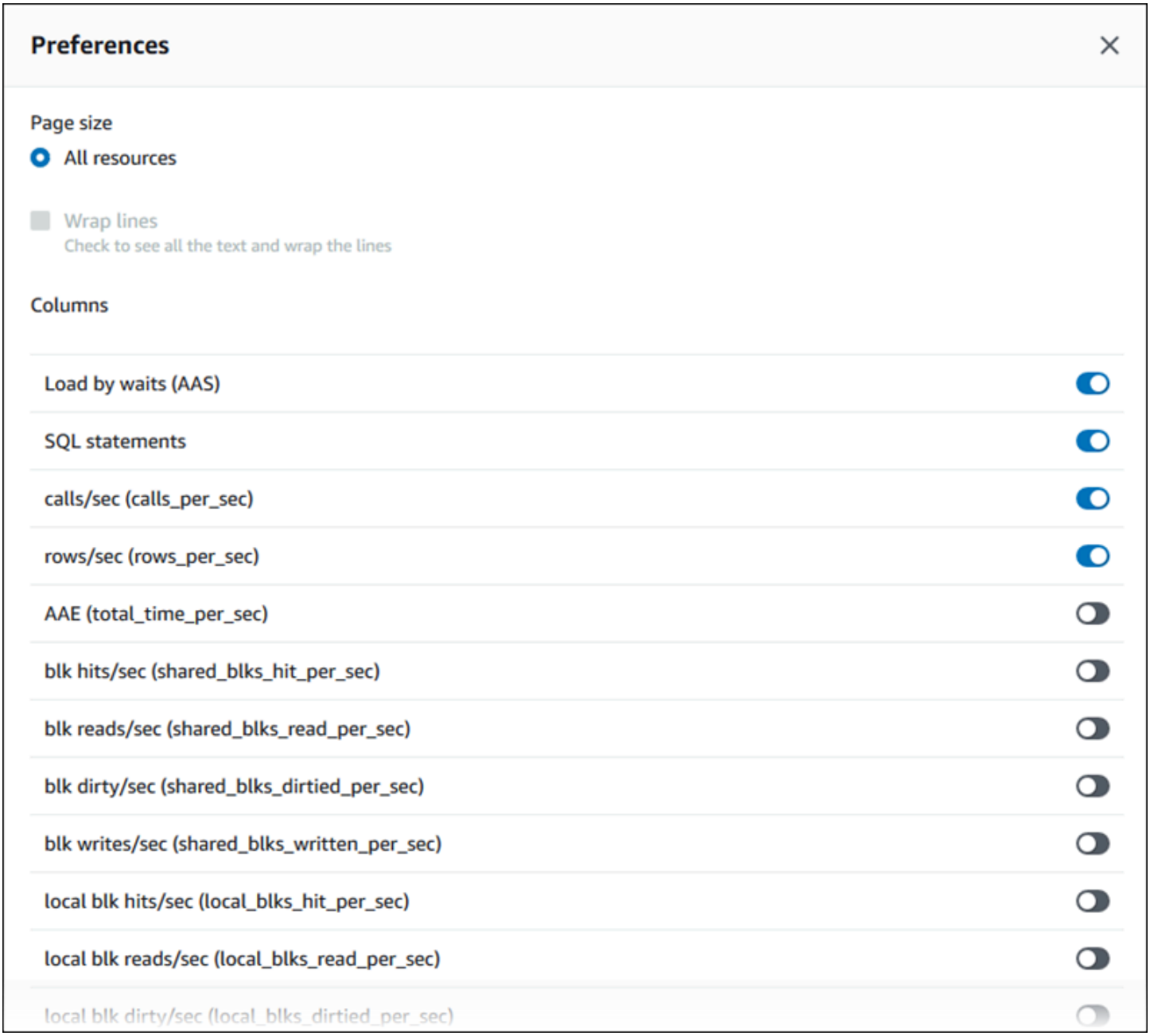
- 支持 SQL ID – SQL ID 的哈希值。此值仅用于当您处理 AWS Support 时引用 SQL ID。AWSSupport 无法访问您实际的 SQL ID 和 SQL 文本。
- 支持摘要 ID – 摘要 ID 的哈希值。此值仅用于当您处理 AWS Support 时引用摘要 ID。AWSSupport 无法访问您实际的摘要 ID 和 SQL 文本。

选择统计数据首选项

您可以通过选择 Preferences（首选项）图标来控制 Top SQL（主要 SQL）选项卡中显示的统计数据。



当您选择 Preferences (首选项) 图标时，Preferences (首选项) 窗口将打开。以下屏幕截图是 Preferences（首选项）窗口的示例。



要启用您希望在 Top SQL（主要 SQL）选项卡中显示的统计数据，请使用鼠标滚动到窗口底部，然后选择 Continue（继续）。

有关 Aurora 引擎每秒或每次调用统计数据的更多信息，请参阅Performance Insights 的 SQL 统计数据中引擎特定的 SQL 统计数据部分

在 Performance Insights 控制面板中访问更多 SQL 文本

预设情况下，Top SQL（主要 SQL）表中的每行为每个 SQL 语句显示 500 字节的 SQL 文本。



当 SQL 语句超过 500 字节时，您可以在 Top SQL（主要 SQL）表下的 SQL text（SQL 文本）部分中查看更多文本。在这种情况下，SQL text（SQL 文本）中显示的文本的最大长度为 4KB。此限制由控制台引入，并受数据库引擎设置的限制的约束。要保存 SQL text（SQL 文本）中显示的文本，请选择 Download（下载）。

主题

- Aurora MySQL 的文本大小限制
- 为 Aurora PostgreSQL 数据库实例设置 SQL 文本限制
- 在 Performance Insights 控制面板中查看和下载 SQL 文本

Aurora MySQL 的文本大小限制

下载 SQL 文本时，数据库引擎将确定文本的最大长度。您可以下载最多为以下每个引擎限制的 SQL 文本。

数据库引擎	已下载文本的最大长度
Aurora MySQL	长度固定为 4096 字节。

性能详情控制台的 SQL text（SQL 文本）部分最多显示引擎返回的最大长度的文本。例如，如果 Aurora MySQL 最多返回 1 KB 到 Performance Insights，则只能收集并显示 1 KB，即使原始查询较大。因此，当您在 SQL text（SQL 文本）中查看查询或下载查询时，性能详情将返回相同的字节数。

如果您使用 AWS CLI 或 API，则性能详情没有控制台强制实施的 4KB 限制。`DescribeDimensionKeys` 和 `GetResourceMetrics` 返回最多 500 字节。

> **Note**
>
> `GetDimensionKeyDetails` 将返回完整的查询，但大小受引擎限制约束。

为 Aurora PostgreSQL 数据库实例设置 SQL 文本限制

Aurora PostgreSQL 以不同的方式处理文本。您可以使用数据库实例参数 `track_activity_query_size` 设置文本大小限制。该参数具有以下特征：

默认文本大小

在 Aurora PostgreSQL 版本 9.6 中，`track_activity_query_size` 参数的默认设置为 1024 字节。在 Aurora PostgreSQL 版本 10 或更高版本中，默认值为 4096 字节。

最大文本大小

对于 Aurora PostgreSQL 版本 12 及更低版本，`track_activity_query_size` 的限制为 102400 字节。版本 13 及更高版本的最大值为 1 MB。

如果引擎返回 1 MB 至 Performance Insights，则控制台仅显示前 4 KB。如果您下载查询，您将得到完整的 1 MB。在这种情况下，查看和下载返回不同的字节数。有关 `track_activity_query_size` 数据库实例参数的更多信息，请参阅 PostgreSQL 文档中的运行时统计数据。

要增加 SQL 文本大小，请增加 `track_activity_query_size` 限制。要修改参数，请更改与 Aurora PostgreSQL 数据库实例关联的参数组中的参数设置。

在实例使用默认参数组时更改设置

1. 为相应数据库引擎和数据库引擎版本创建新的数据库实例参数组。
2. 在新参数组中设置参数。
3. 将新参数组与数据库实例相关联。

有关设置数据库实例参数的信息，请参阅 在 Amazon Aurora 中修改数据库参数组中的参数。

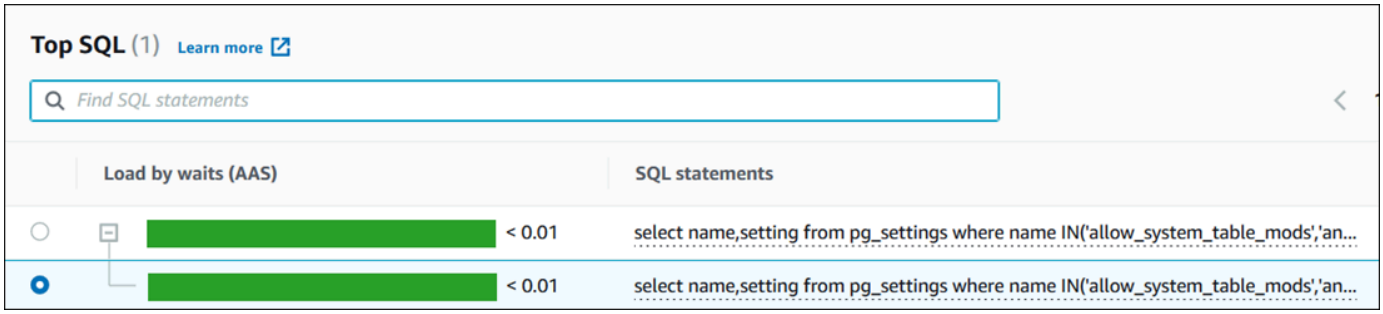
在 Performance Insights 控制面板中查看和下载 SQL 文本

在 Performance Insights 控制面板中，您可以查看或下载 SQL 文本。

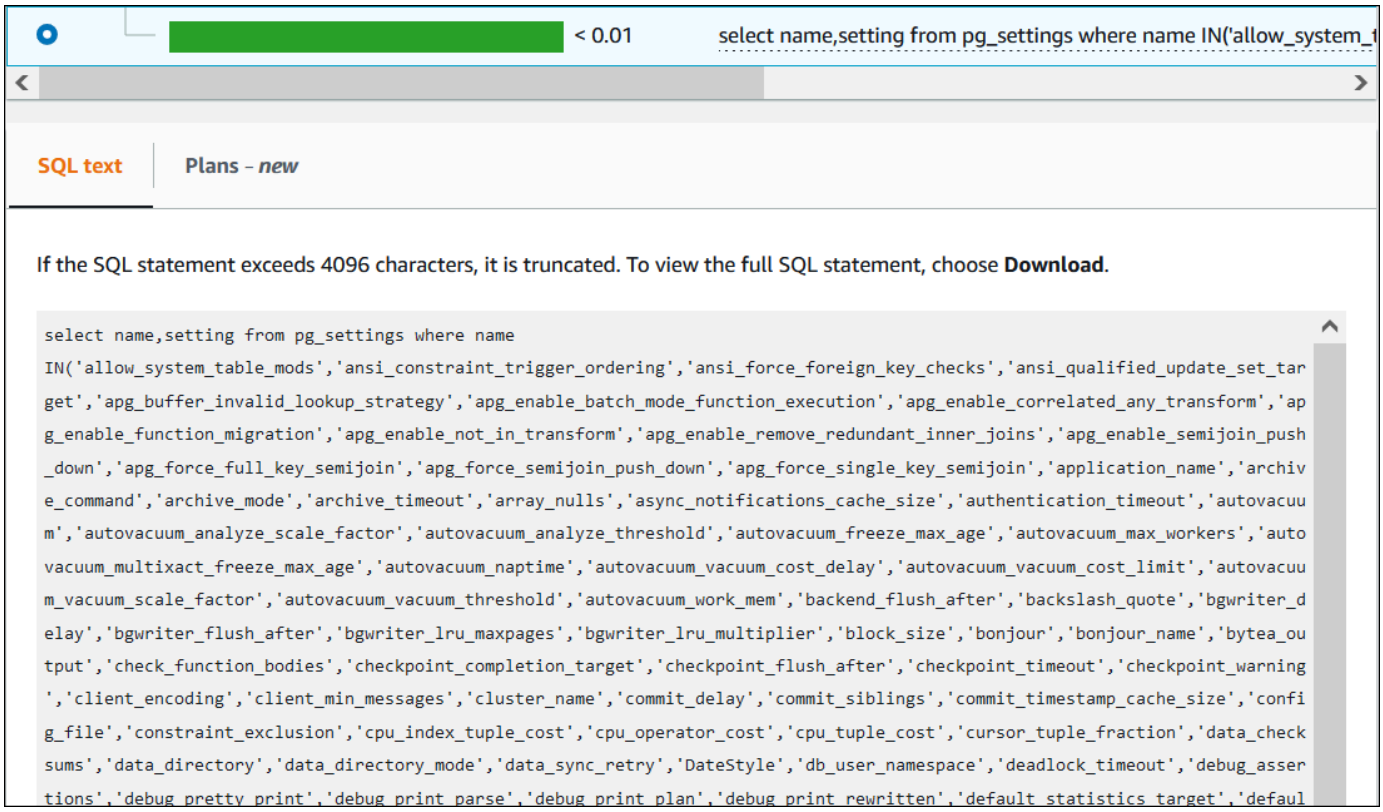
在 Performance Insights 控制面板中查看更多 SQL 文本

1. 通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Performance Insights。
3. 选择一个数据库实例。
4. 在性能详情控制面板中向下滚动到主要 SQL 选项卡。
5. 选择加号来展开 SQL 摘要，然后选择该摘要的子查询之一。

   具有大于 500 字节的文本的 SQL 语句如下图所示。



6. 向下滚动至 SQL text（SQL 文本）选项卡。



Performance Insights 控制面板可以为每个 SQL 语句最多显示 4096 字节。

7. （可选）选择复制来复制所显示的 SQL 语句，或选择下载来下载 SQL 语句以查看不超过数据库引擎限制的 SQL 文本。

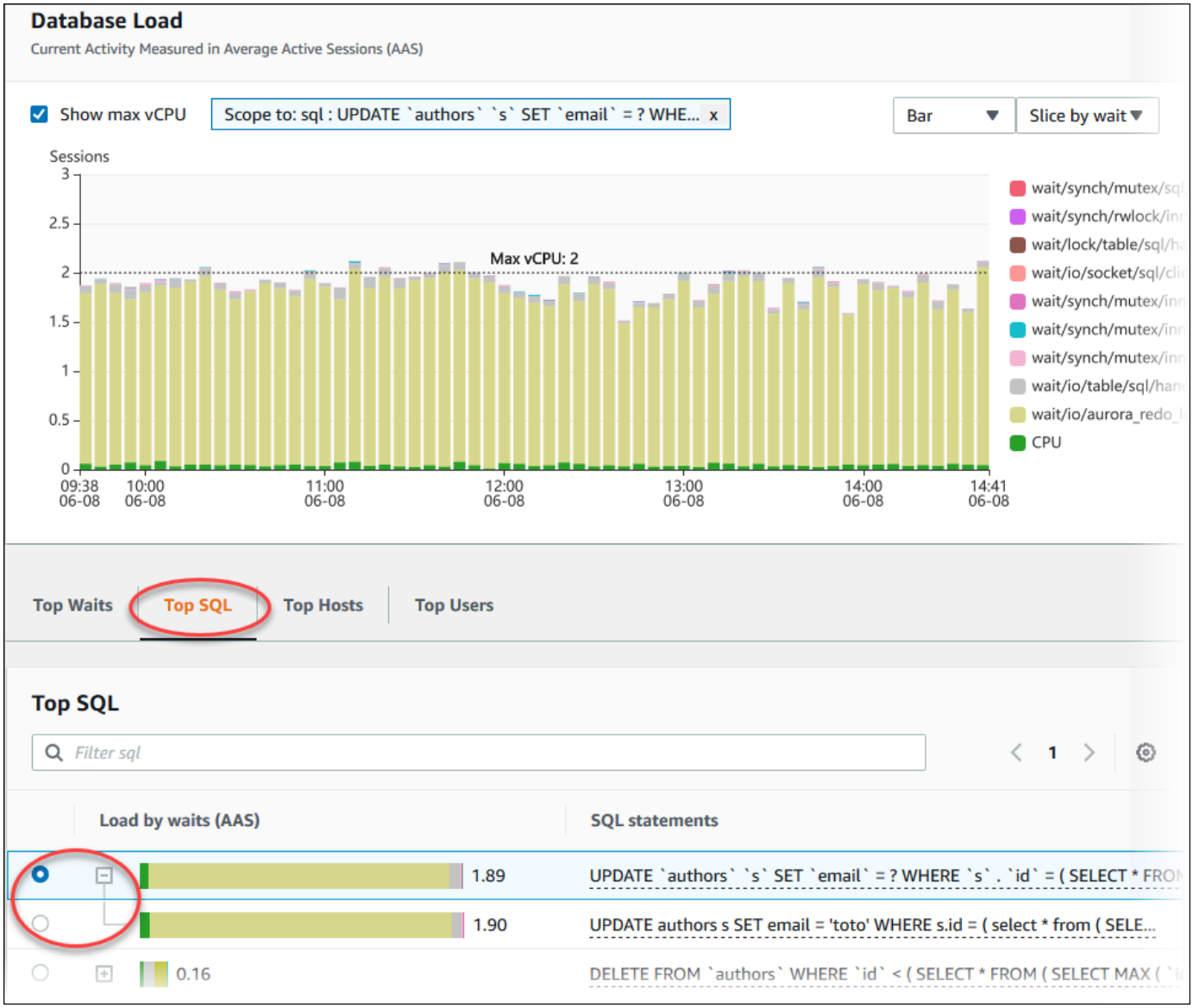
> **Note**
>
> 要复制或下载 SQL 语句，请禁用弹出窗口阻止程序。

在 Performance Insights 控制面板中查看 SQL 统计数据

在 Performance Insights 控制面板中，SQL 统计数据可在 Database load（数据库负载）图表的 Top SQL（主要 SQL）选项卡中找到。

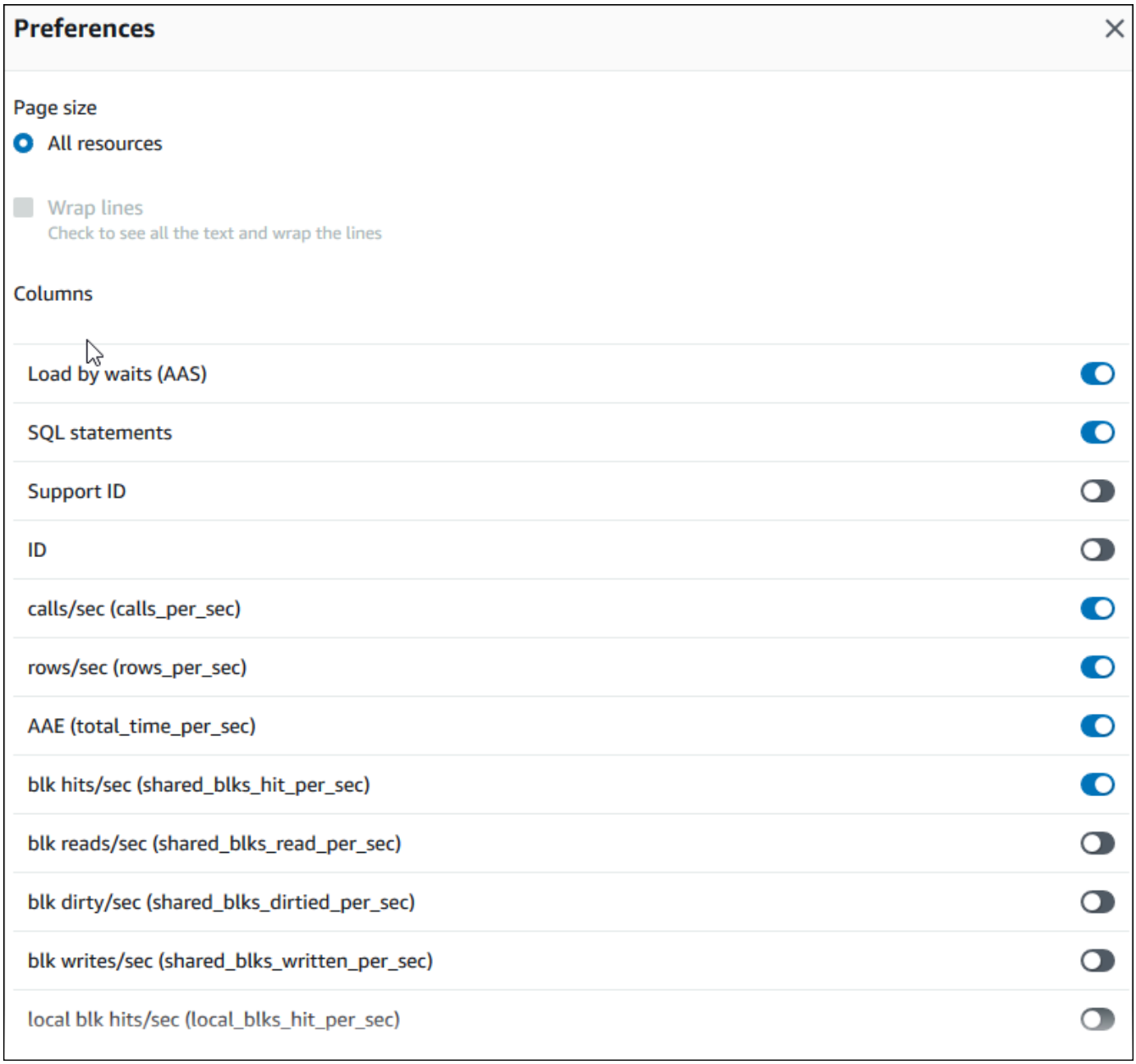
查看 SQL 统计数据

1. 通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在左侧导航窗格中，选择 Performance Insights。
3. 在页面顶部，选择要查看 SQL 统计数据的数据库。
4. 滚动到页面底部并选择 Top SQL（主要 SQL）选项卡。
5. 选择单独的语句（仅限 Aurora MySQL）或摘要查询。

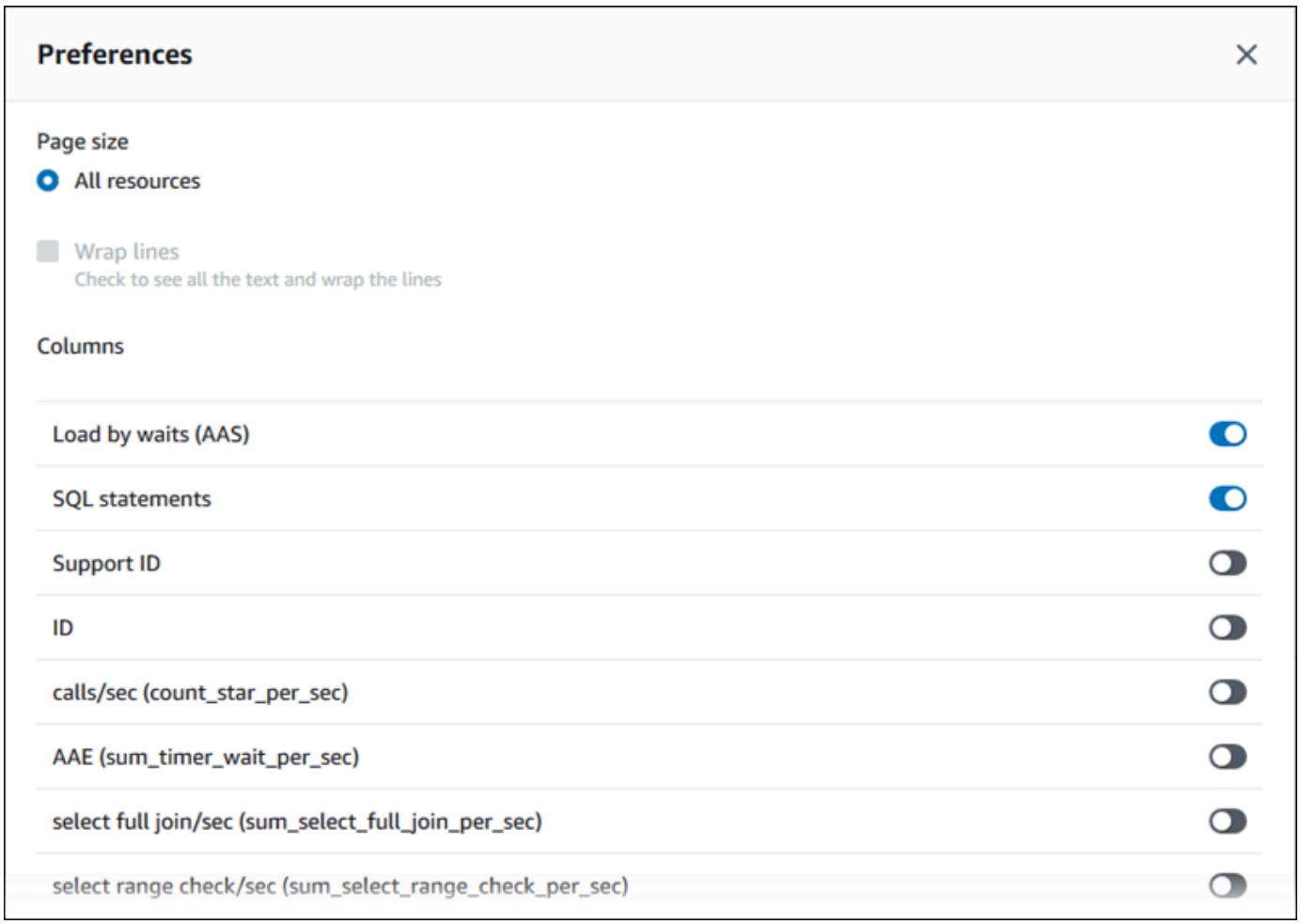


6. 通过选择图表右上角的齿轮图标来选择要显示的统计数据。有关 Amazon RDSAurora 引擎的 SQL 统计数据的说明，请参阅 Performance Insights 的 SQL 统计数据。

   以下示例显示 Aurora PostgreSQL 的首选项。



   以下示例显示 Aurora MySQL 数据库实例的首选项。



7. 选择“Save（保存）”以保存首选项。

   Top SQL（主要 SQL）表将刷新。

查看性能详情主动建议

Amazon RDS 性能详情可监控特定指标，并自动通过分析指定资源可能存在问题的级别来创建阈值。当新的指标值在给定时间段内超过预定义的阈值时，性能详情会生成主动建议。此建议有助于防止数据库性能将来受到影响。要获得这些主动建议，您必须开启性能详情，并设置付费套餐保留期。

有关开启 Performance Insights 的更多信息，请参阅为 Aurora 打开和关闭性能详情。有关性能详情的定价和数据留存的更多信息，请参阅性能详情的定价和数据留存。

要了解主动建议支持的区域、数据库引擎和实例类别，请参阅支持性能详情功能的 Amazon Aurora 数据库引擎、区域和实例类。

您可以在建议详细信息页面中查看主动建议的详细分析和建议的调查。

有关建议的更多信息，请参阅 来自 Amazon Aurora 的建议。

查看主动建议的详细分析

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，执行以下任何一项操作：

   - 选择建议。

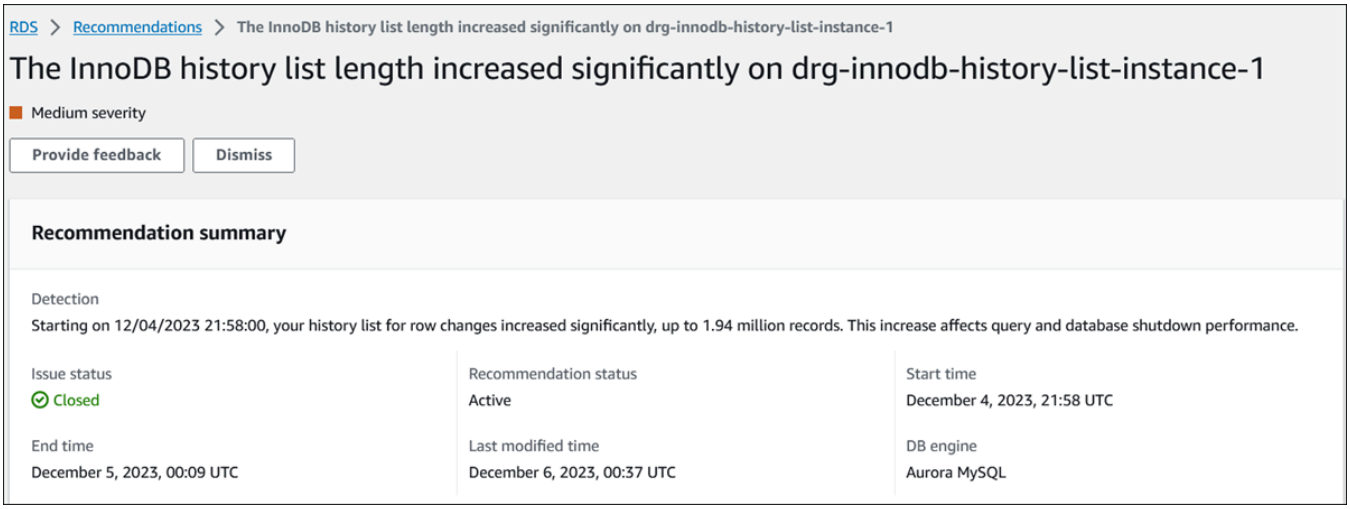
     建议页面显示按您账户中所有资源的严重性排序的建议列表。
   - 选择数据库，然后在数据库页中为资源选择建议。

     建议选项卡显示所选资源的建议及其详细信息。
3. 查找主动建议，然后选择查看详细信息。

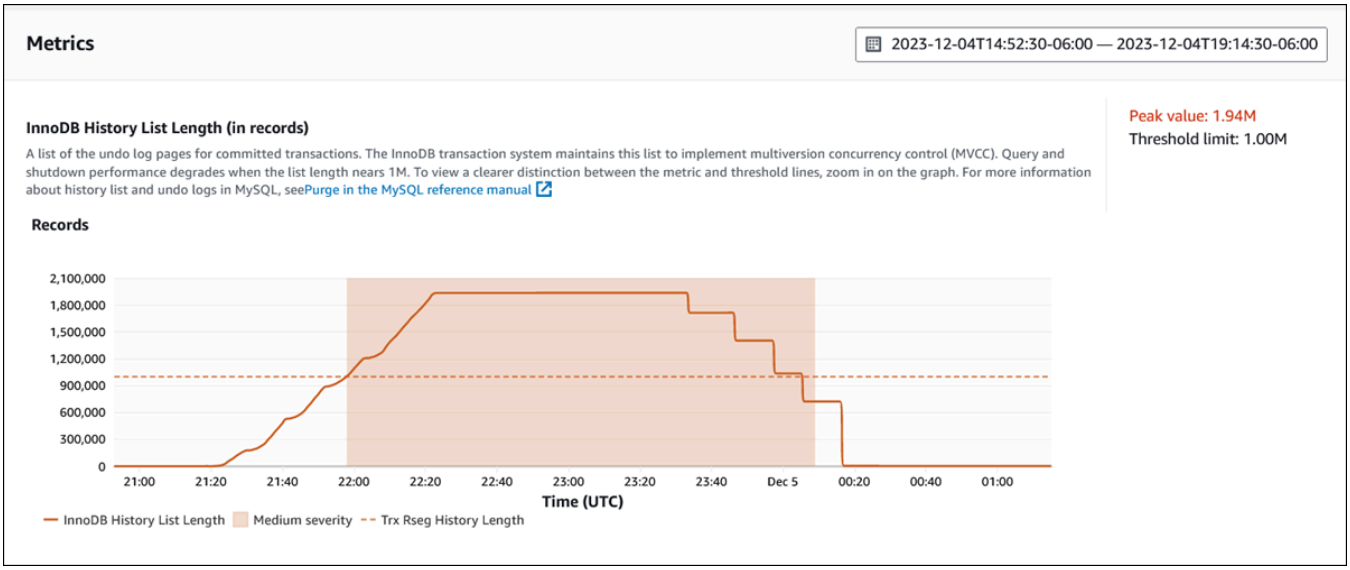
   此时会显示建议详细信息页面。标题提供了受影响资源的名称以及检测到的问题和严重性。

   以下是建议详细信息页面上的组件：

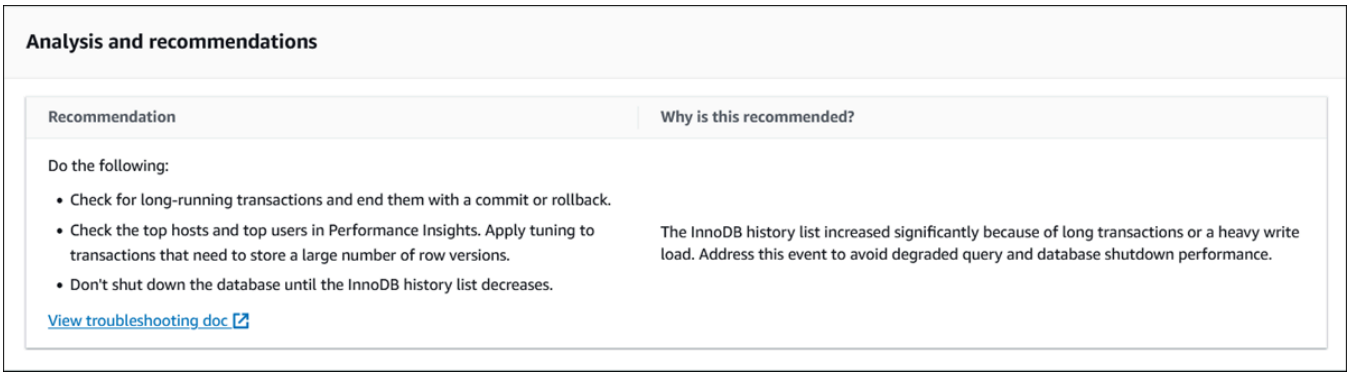
   - 建议摘要 - 检测到的问题、建议和问题状态、问题开始和结束时间、建议修改时间以及引擎类型。

   - 指标 - 检测到的问题的图表。每个图表都显示由资源的基线行为确定的阈值，以及从问题开始时间报告的指标数据。



- 分析和建议 - 建议和所提建议的理由。



您可以查看问题的原因，然后执行建议的操作来解决问题，或者选择右上角的忽略来忽略建议。

使用 Performance Insights API for Aurora 检索指标

开启性能详情后，API 将提供实例性能的可见性。Amazon CloudWatch Logs 日志为AWS服务提供售卖监控指标的权威源。

Performance Insights 提供了按平均活动会话 (AAS) 衡量的数据库负载的特定于域的视图。对 API 使用者而言，此指标看起来像是二维时间序列数据集。数据的时间维度提供所查询时间范围的每个时间点的数据库负载数据。每个时间点将分解与所请求维度相关的整体负载，如相应时间点测量的 SQL、Wait-event、User 或 Host。

Amazon RDS Performance Insights 用于监控您的 Amazon Aurora集群，使您可以分析数据库性能和排查数据库性能问题。查看 Performance Insights 数据的一种方法是在 AWS Management Console

中。Performance Insights 还提供公有 API，以便您可以查询自己的数据。您可以使用 API 来执行以下操作：

- 将数据卸载到数据库中
- 将 Performance Insights 数据添加到现有监控控制面板
- 构建监控工具

要使用 Performance Insights API，请在您的 Amazon RDS 数据库实例之一上启用 Performance Insights。有关启用 Performance Insights 的信息，请参阅 为 Aurora 打开和关闭性能详情。有关 Performance Insights API 的更多信息，请参阅 Amazon RDS Performance Insights API 参考。

Performance Insights API 提供以下操作。

Performance Insights 操作	AWS CLI command	描述
CreatePerformanceAnalysisReport	aws pi create-performance-analysis-report	为数据库实例创建特定时间段的性能分析报告。结果为 AnalysisReportId ，这是报告的唯一标识符。
DeletePerformanceAnalysisReport	aws pi delete-performance-analysis-report	删除性能分析报告。
DescribeDimensionKeys	aws pi describe-dimension-keys	对于特定的时间段，检索指标的前 N 个维度键。
GetDimensionKeyDetails	aws pi get-dimension-key-details	检索数据库实例或数据源的指定维度组的属性。例如，如果您指定了 SQL ID，并且有维度详细信息，则 GetDimensionKeyDetails 将检索与此 ID 关联的维度 db.sql.statement 的全文。此操作很有用，因为 GetResourceMetrics 和 DescribeD

Performance Insights 操作	AWS CLI command	描述
		imensionKeys 不支持检索大型 SQL 语句文本。
GetPerformanceAnalysisReport	aws pi get-performance-analysis-report	检索报告，包括报告的见解。结果包括报告状态、报告 ID、报告时间详情、见解和建议。
GetResourceMetadata	aws pi get-resource-metadata	检索不同功能的元数据。例如，元数据可以表明特定数据库实例上的某个功能已打开或关闭。
GetResourceMetrics	aws pi get-resource-metrics	检索一组数据来源在一段时间内的 Performance Insights 指标。您可以提供特定维度组和维度，并为每个组提供聚合和筛选条件。
ListAvailableResourceDimensions	aws pi list-available-resource-dimensions	检索特定实例上每个特定指标类型可查询的维度。
ListAvailableResourceMetrics	aws pi list-available-resource-metrics	检索指定指标类型的所有可用指标，指定数据库实例可用该指标进行查询。
ListPerformanceAnalysisReports	aws pi list-performance-analysis-reports	检索数据库实例的所有可用分析报告。这些报告根据每个报告的开始时间列出。
ListTagsForResource	aws pi list-tags-for-resource	列出添加到资源的所有元数据标签。该列表包含标签的名称和值。
TagResource	aws pi tag-resource	将元数据标签添加到 Amazon RDS 资源中。该标签包含名称和值。

Performance Insights 操作	AWS CLI command	描述
UntagResource	aws pi untag-resource	从资源中删除元数据标签。

有关检索性能详情的时间序列指标和 AWS CLI 示例的更多信息，请参阅以下主题。

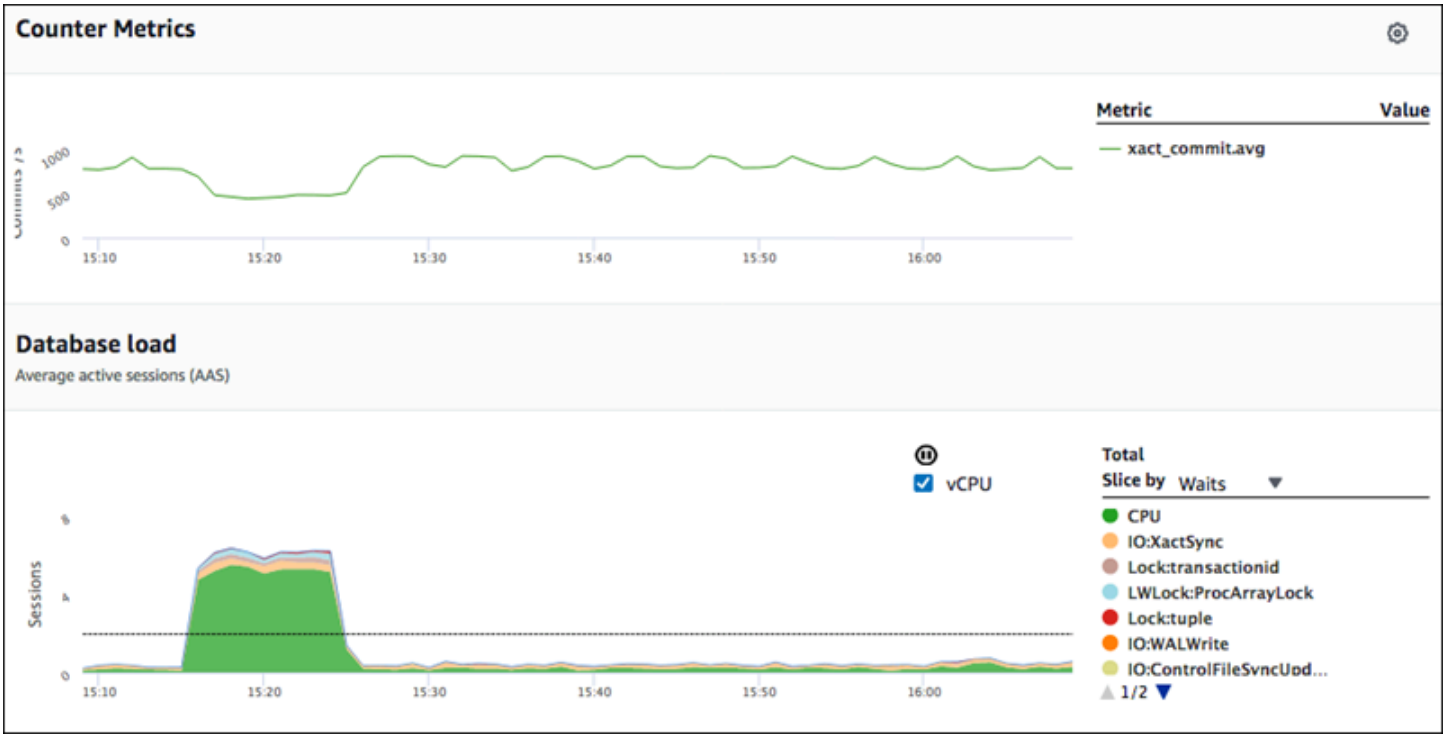
主题

- 检索性能详情的时间序列指标
- Performance Insights 的 AWS CLI 示例

检索性能详情的时间序列指标

`GetResourceMetrics` 操作从 Performance Insights 数据中检索一个或多个时间序列指标。`GetResourceMetrics` 需要指标和时间段，并返回包含数据点列表的响应。

例如，AWS Management Console使用 `GetResourceMetrics` 来填充 Counter Metrics (计数器指标) 图表和 Database Load (数据库负载) 图表，如下图所示。



GetResourceMetrics 返回的所有指标都是标准的时间序列指标，但 db.load 除外。此指标显示在 Database Load (数据库负载) 图表中。db.load 指标不同于其他时间序列指标，因为您可以将它分为称为维度的子组件。在上图中，按组成 db.load 的等待状态对 db.load 进行细分和分组。

> **Note**
>
> GetResourceMetrics 也可以返回 db.sampleload 指标，但 db.load 指标在大多数情况下是合适的。

有关 GetResourceMetrics 返回的计数器指标的信息，请参阅性能详情计数器指标。

指标支持以下计算：

- 平均值 – 指标在一段时间内的平均值。在指标名称后面附加 .avg。
- 最小值 – 指标在一段时间内的最小值。在指标名称后面附加 .min。
- 最大值 – 指标在一段时间内的最大值。在指标名称后面附加 .max。
- 总计 – 指标值在一段时间内的总计。在指标名称后面附加 .sum。
- 样本数 – 在一段时间内收集指标的次数。在指标名称后面附加 .sample_count。

例如，假定在 300 秒（5 分钟）时段内收集指标，并且每分钟收集一次指标。各分钟的值为 1、2、3、4 和 5。在本例中，返回以下计算：

- 平均值 – 3
- 最小值 – 1
- 最大值 – 5
- 总计 – 15
- 样本数 – 5

有关使用 get-resource-metrics AWS CLI 命令的信息，请参阅 get-resource-metrics。

对于 --metric-queries 选项，请指定一个或多个要获取其结果的查询。每个查询包括必需的 Metric 和可选的 GroupBy 和 Filter 参数。以下是 --metric-queries 选项规范的示例。

```
{
   "Metric": "string",
```

```
  "GroupBy": {
    "Group": "string",
    "Dimensions": ["string", ...],
    "Limit": integer
  },
  "Filter": {"string": "string"
    ...}
```

Performance Insights 的 AWS CLI 示例

在以下各节中，详细了解性能详情的 AWS Command Line Interface（AWS CLI）并使用 AWS CLI 示例。

主题

- 性能详情的 AWS CLI 的内置帮助
- 检索计数器指标
- 检索首要等待事件的数据库负载平均值
- 检索首要 SQL 的数据库负载平均值
- 检索按 SQL 筛选的数据库负载平均值
- 检索 SQL 语句的全文
- 创建一段时间的性能分析报告
- 检索性能分析报告
- 列出数据库实例的所有性能分析报告
- 删除性能分析报告
- 向性能分析报告中添加标签
- 列出性能分析报告的所有标签
- 从性能分析报告中删除标签

性能详情的 AWS CLI 的内置帮助

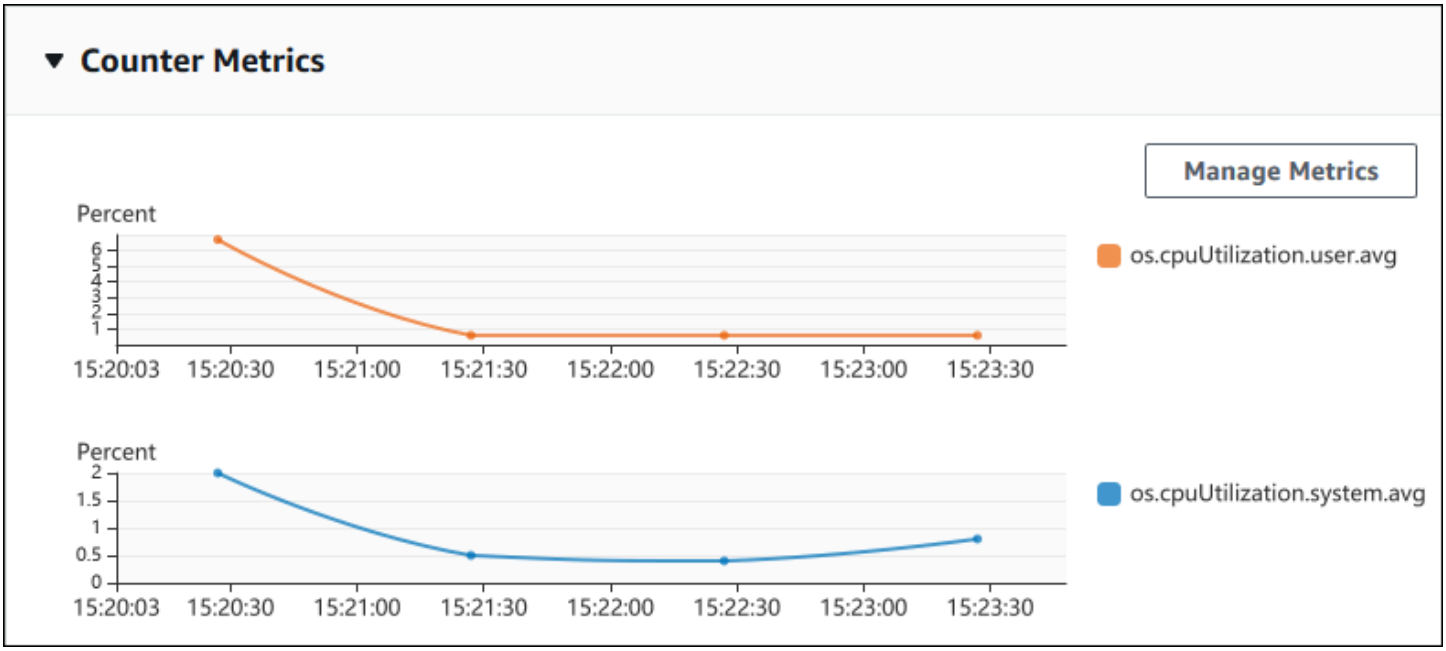
您可以使用 AWS CLI 查看 Performance Insights 数据。可以通过在命令行上输入以下内容来查看 Performance Insights 的 AWS CLI 命令的帮助。

```
aws pi help
```

如果尚未安装 AWS CLI，请参阅《AWS CLI 用户指南》中的安装 AWS CLI 来了解安装信息。

检索计数器指标

以下屏幕截图显示 AWS Management Console 中的两个计数器指标图表。



以下示例显示如何收集 AWS Management Console 用于生成两个计数器指标图表的相同数据。

对于 Linux、macOS 或 Unix：

```
aws pi get-resource-metrics \
   --service-type RDS \
   --identifier db-ID \
   --start-time 2018-10-30T00:00:00Z \
   --end-time   2018-10-30T01:00:00Z \
   --period-in-seconds 60 \
   --metric-queries '[{"Metric": "os.cpuUtilization.user.avg"  },
                      {"Metric": "os.cpuUtilization.idle.avg"}]'
```

对于 Windows：

```
aws pi get-resource-metrics ^
   --service-type RDS ^
   --identifier db-ID ^
   --start-time 2018-10-30T00:00:00Z ^
   --end-time   2018-10-30T01:00:00Z ^
```

```
   --period-in-seconds 60 ^
   --metric-queries '[{"Metric": "os.cpuUtilization.user.avg"  },
                      {"Metric": "os.cpuUtilization.idle.avg"}]'
```

还可以通过为 `--metrics-query` 选项指定文件来使命令更易于读取。以下示例为该选项使用名为 query.json 的文件。此文件具有以下内容。

```
[
    {
        "Metric": "os.cpuUtilization.user.avg"
    },
    {
        "Metric": "os.cpuUtilization.idle.avg"
    }
]
```

运行以下命令来使用此文件。

对于 Linux、macOS 或 Unix：

```
aws pi get-resource-metrics \
   --service-type RDS \
   --identifier db-ID \
   --start-time 2018-10-30T00:00:00Z \
   --end-time   2018-10-30T01:00:00Z \
   --period-in-seconds 60 \
   --metric-queries file://query.json
```

对于 Windows：

```
aws pi get-resource-metrics ^
   --service-type RDS ^
   --identifier db-ID ^
   --start-time 2018-10-30T00:00:00Z ^
   --end-time   2018-10-30T01:00:00Z ^
   --period-in-seconds 60 ^
   --metric-queries file://query.json
```

上一个示例为各选项指定了以下值：

- `--service-type` – 对于 Amazon RDS 来说为 `RDS`

- `--identifier` – 数据库实例的资源 ID
- `--start-time` 和 `--end-time` – 要查询的期间的 ISO 8601 `DateTime` 值，支持多种格式

它查询一小时时间范围：

- `--period-in-seconds` – 对于每分钟查询来说为 `60`
- `--metric-queries` – 两个查询的数组，每个查询只用于一个指标。

  指标名称使用点在有用的类别中分类指标，最后一个元素是函数。在示例中，对于每个查询来说，此函数是 `avg`。与 Amazon CloudWatch 一样，支持的函数为 `min`、`max`、`total` 和 `avg`。

响应类似于以下内容。

```
{
    "Identifier": "db-XXX",
    "AlignedStartTime": 1540857600.0,
    "AlignedEndTime": 1540861200.0,
    "MetricList": [
        { //A list of key/datapoints
            "Key": {
                "Metric": "os.cpuUtilization.user.avg" //Metric1
            },
            "DataPoints": [
                //Each list of datapoints has the same timestamps and same number of
 items
                {
                    "Timestamp": 1540857660.0, //Minute1
                    "Value": 4.0
                },
                {
                    "Timestamp": 1540857720.0, //Minute2
                    "Value": 4.0
                },
                {
                    "Timestamp": 1540857780.0, //Minute 3
                    "Value": 10.0
                }
                //... 60 datapoints for the os.cpuUtilization.user.avg metric
            ]
        },
        {
```

```
            "Key": {
                "Metric": "os.cpuUtilization.idle.avg" //Metric2
            },
            "DataPoints": [
                {
                    "Timestamp": 1540857660.0, //Minute1
                    "Value": 12.0
                },
                {
                    "Timestamp": 1540857720.0, //Minute2
                    "Value": 13.5
                },
                //... 60 datapoints for the os.cpuUtilization.idle.avg metric
            ]
        }
    ] //end of MetricList
} //end of response
```

响应具有 Identifier、AlignedStartTime 和 AlignedEndTime。但 --period-in-seconds 值为 60，开始和结束时间已与分钟对齐。如果 --period-in-seconds 为 3600，则开始和结束时间已与小时对齐。

响应中的 MetricList 具有许多条目，每个条目具有 Key 和 DataPoints 条目。每个 DataPoint 具有 Timestamp 和 Value。每个 Datapoints 列表具有 60 个数据点，因为查询针对一小时内的每分钟数据，具有 Timestamp1/Minute1、Timestamp2/Minute2 等，一直到 Timestamp60/Minute60。

因为查询用于两个不同的计数器指标，响应 MetricList 中有两个元素。

检索首要等待事件的数据库负载平均值

以下示例是 AWS Management Console用于生成堆栈区域线图的相同查询。此示例检索按前七个等待事件划分负载的最后一个小时的 db.load.avg。命令与 检索计数器指标 中的命令相同。不过，query.json 文件具有以下内容。

```
[
    {
        "Metric": "db.load.avg",
        "GroupBy": { "Group": "db.wait_event", "Limit": 7 }
    }
]
```

运行以下命令。

对于 Linux、macOS 或 Unix：

```
aws pi get-resource-metrics \
   --service-type RDS \
   --identifier db-ID \
   --start-time 2018-10-30T00:00:00Z \
   --end-time   2018-10-30T01:00:00Z \
   --period-in-seconds 60 \
   --metric-queries file://query.json
```

对于 Windows：

```
aws pi get-resource-metrics ^
   --service-type RDS ^
   --identifier db-ID ^
   --start-time 2018-10-30T00:00:00Z ^
   --end-time   2018-10-30T01:00:00Z ^
   --period-in-seconds 60 ^
   --metric-queries file://query.json
```

此示例指定指标 db.load.avg 和前七个等待事件的 GroupBy。有关此示例的有效值的详细信息，请参阅 Performance Insights API 参考 中的 DimensionGroup。

响应类似于以下内容。

```
{
    "Identifier": "db-XXX",
    "AlignedStartTime": 1540857600.0,
    "AlignedEndTime": 1540861200.0,
    "MetricList": [
        { //A list of key/datapoints
            "Key": {
                //A Metric with no dimensions. This is the total db.load.avg
                "Metric": "db.load.avg"
            },
            "DataPoints": [
                //Each list of datapoints has the same timestamps and same number of
 items
                {
                    "Timestamp": 1540857660.0, //Minute1
```

```
                    "Value": 0.5166666666666667
                },
                {
                    "Timestamp": 1540857720.0, //Minute2
                    "Value": 0.38333333333333336
                },
                {
                    "Timestamp": 1540857780.0, //Minute 3
                    "Value": 0.26666666666666666
                }
                //... 60 datapoints for the total db.load.avg key
            ]
        },
        {
            "Key": {
                //Another key. This is db.load.avg broken down by CPU
                "Metric": "db.load.avg",
                "Dimensions": {
                    "db.wait_event.name": "CPU",
                    "db.wait_event.type": "CPU"
                }
            },
            "DataPoints": [
                {
                    "Timestamp": 1540857660.0, //Minute1
                    "Value": 0.35
                },
                {
                    "Timestamp": 1540857720.0, //Minute2
                    "Value": 0.15
                },
                //... 60 datapoints for the CPU key
            ]
        },
        //... In total we have 8 key/datapoints entries, 1) total, 2-8) Top Wait Events
    ] //end of MetricList
} //end of response
```

在此响应中，MetricList 中有八个条目。有一个有关总 db.load.avg 的条目，还有七个条目，其中每个条目关于按前七个等待事件之一划分的 db.load.avg。与第一个示例不同，因为具有分组维度，所以必须具有一个用于每个指标分组的键。不能像在基本计数器指标使用案例中那样每个指标只有一个键。

检索首要 SQL 的数据库负载平均值

以下示例按前 10 个 SQL 语句对 db.wait_events 进行分组。有两个不同的 SQL 语句组：

- db.sql – 完整的 SQL 语句，例如 select * from customers where customer_id = 123
- db.sql_tokenized – 令牌化的 SQL 语句，例如 select * from customers where customer_id = ?

在分析数据库性能时，将仅参数不同的 SQL 语句视为一个逻辑项目很有用。因此，您在查询时可以使用 db.sql_tokenized。不过，尤其在您对说明计划感兴趣时，查看带参数的完整 SQL 语句和按 db.sql 分组查询有时会更有用。令牌化和完整 SQL 之间存在父-子关系，多个完整 SQL（子级）分组在同一令牌化 SQL（父级）下。

此示例中的命令类似于 检索首要等待事件的数据库负载平均值 中的命令。不过，query.json 文件具有以下内容。

```
[
    {
        "Metric": "db.load.avg",
        "GroupBy": { "Group": "db.sql_tokenized", "Limit": 10 }
    }
]
```

下面的示例使用了 db.sql_tokenized。

对于 Linux、macOS 或 Unix：

```
aws pi get-resource-metrics \
   --service-type RDS \
   --identifier db-ID \
   --start-time 2018-10-29T00:00:00Z \
   --end-time   2018-10-30T00:00:00Z \
   --period-in-seconds 3600 \
   --metric-queries file://query.json
```

对于 Windows：

```
aws pi get-resource-metrics ^
   --service-type RDS ^
   --identifier db-ID ^
   --start-time 2018-10-29T00:00:00Z ^
```

```
    --end-time   2018-10-30T00:00:00Z  ^
    --period-in-seconds 3600 ^
    --metric-queries file://query.json
```

此示例查询 24 小时，以秒为单位的期间为一小时。

此示例指定指标 db.load.avg 和前七个等待事件的 GroupBy。有关此示例的有效值的详细信息，请参阅 Performance Insights API 参考 中的 DimensionGroup。

响应类似于以下内容。

```
{
    "AlignedStartTime": 1540771200.0,
    "AlignedEndTime": 1540857600.0,
    "Identifier": "db-XXX",

    "MetricList": [ //11 entries in the MetricList
        {
            "Key": { //First key is total
                "Metric": "db.load.avg"
            }
            "DataPoints": [ //Each DataPoints list has 24 per-hour Timestamps and a
 value
                {
                    "Value": 1.6964980544747081,
                    "Timestamp": 1540774800.0
                },
                //... 24 datapoints
            ]
        },
        {
            "Key": { //Next key is the top tokenized SQL
                "Dimensions": {
                    "db.sql_tokenized.statement": "INSERT INTO authors (id,name,email)
 VALUES\n( nextval(?)  ,?,?)",
                    "db.sql_tokenized.db_id": "pi-2372568224",
                    "db.sql_tokenized.id": "AKIAIOSFODNN7EXAMPLE"
                },
                "Metric": "db.load.avg"
            },
            "DataPoints": [ //... 24 datapoints
            ]
        },
```

```
        // In total 11 entries, 10 Keys of top tokenized SQL, 1 total key
    ] //End of MetricList
} //End of response
```

此响应的 MetricList 中具有 11 个条目（1 个总计，10 个首要令牌化 SQL），其中每个条目具有 24 个每小时 DataPoints。

对于令牌化 SQL，每个维度列表中具有三个条目：

- db.sql_tokenized.statement – 令牌化 SQL 语句。
- db.sql_tokenized.db_id – 用于引用 SQL 的本机数据库 ID，或 Performance Insights 为您生成的合成 ID（如果本机数据库 ID 不可用）。此示例返回 pi-2372568224 合成 ID。
- db.sql_tokenized.id – Performance Insights 中的查询的 ID。

  在 AWS Management Console中，此 ID 称为支持 ID。它如此命名是因为 ID 是 AWS 支持服务可检查以帮助您解决数据库问题的数据。AWS 极其重视您的数据的安全性和隐私，几乎所有数据都使用您的 AWS KMS 密钥进行加密存储。因此，AWS 中的任何人都无法查看这些数据。在上一个示例中，tokenized.statement 和 tokenized.db_id 都进行了加密存储。如果您的数据库出现问题，AWS Support 可以通过引用支持 ID 来帮助您。

在查询时，在 Group 中指定 GroupBy 可能很方便。不过，要更精细地控制返回的数据，请指定维度列表。例如，如果所需的所有内容是 db.sql_tokenized.statement，则可将 Dimensions 属性添加到 query.json 文件中。

```
[
    {
        "Metric": "db.load.avg",
        "GroupBy": {
            "Group": "db.sql_tokenized",
            "Dimensions":["db.sql_tokenized.statement"],
            "Limit": 10
        }
    }
]
```

检索按 SQL 筛选的数据库负载平均值



上图显示选择了特定查询，并且首要平均活动会话堆栈区域线图限定为该查询。虽然查询仍针对前七个总体等待事件，已筛选响应的值。筛选条件使它只考虑与特定筛选条件匹配的会话。

此示例中的相应 API 查询类似于 检索首要 SQL 的数据库负载平均值 中的命令。不过，query.json 文件具有以下内容。

```
[
 {
        "Metric": "db.load.avg",
        "GroupBy": { "Group": "db.wait_event", "Limit": 5  },
        "Filter": { "db.sql_tokenized.id": "AKIAIOSFODNN7EXAMPLE" }
    }
]
```

对于 Linux、macOS 或 Unix：

```
aws pi get-resource-metrics \
   --service-type RDS \
   --identifier db-ID \
   --start-time 2018-10-30T00:00:00Z \
   --end-time   2018-10-30T01:00:00Z \
```

```
    --period-in-seconds 60 \
    --metric-queries file://query.json
```

对于 Windows：

```
aws pi get-resource-metrics ^
    --service-type RDS ^
    --identifier db-ID ^
    --start-time 2018-10-30T00:00:00Z ^
    --end-time   2018-10-30T01:00:00Z ^
    --period-in-seconds 60 ^
    --metric-queries file://query.json
```

响应类似于以下内容。

```
{
    "Identifier": "db-XXX",
    "AlignedStartTime": 1556215200.0,
    "MetricList": [
        {
            "Key": {
                "Metric": "db.load.avg"
            },
            "DataPoints": [
                {
                    "Timestamp": 1556218800.0,
                    "Value": 1.4878117913832196
                },
                {
                    "Timestamp": 1556222400.0,
                    "Value": 1.192823803967328
                }
            ]
        },
        {
            "Key": {
                "Metric": "db.load.avg",
                "Dimensions": {
                    "db.wait_event.type": "io",
                    "db.wait_event.name": "wait/io/aurora_redo_log_flush"
                }
            },
            "DataPoints": [
```

```
                {
                    "Timestamp": 1556218800.0,
                    "Value": 1.1360544217687074
                },
                {
                    "Timestamp": 1556222400.0,
                    "Value": 1.058051341890315
                }
            ]
        },
        {
            "Key": {
                "Metric": "db.load.avg",
                "Dimensions": {
                    "db.wait_event.type": "io",
                    "db.wait_event.name": "wait/io/table/sql/handler"
                }
            },
            "DataPoints": [
                {
                    "Timestamp": 1556218800.0,
                    "Value": 0.16241496598639457
                },
                {
                    "Timestamp": 1556222400.0,
                    "Value": 0.05163360560093349
                }
            ]
        },
        {
            "Key": {
                "Metric": "db.load.avg",
                "Dimensions": {
                    "db.wait_event.type": "synch",
                    "db.wait_event.name": "wait/synch/mutex/innodb/
aurora_lock_thread_slot_futex"
                }
            },
            "DataPoints": [
                {
                    "Timestamp": 1556218800.0,
                    "Value": 0.11479591836734694
                },
                {
```

```
                    "Timestamp": 1556222400.0,
                    "Value": 0.013127187864644107
                }
            ]
        },
        {
            "Key": {
                "Metric": "db.load.avg",
                "Dimensions": {
                    "db.wait_event.type": "CPU",
                    "db.wait_event.name": "CPU"
                }
            },
            "DataPoints": [
                {
                    "Timestamp": 1556218800.0,
                    "Value": 0.05215419501133787
                },
                {
                    "Timestamp": 1556222400.0,
                    "Value": 0.05805134189031505
                }
            ]
        },
        {
            "Key": {
                "Metric": "db.load.avg",
                "Dimensions": {
                    "db.wait_event.type": "synch",
                    "db.wait_event.name": "wait/synch/mutex/innodb/lock_wait_mutex"
                }
            },
            "DataPoints": [
                {
                    "Timestamp": 1556218800.0,
                    "Value": 0.017573696145124718
                },
                {
                    "Timestamp": 1556222400.0,
                    "Value": 0.002333722287047841
                }
            ]
        }
    ],
```

```
    "AlignedEndTime": 1556222400.0
} //end of response
```

在此响应中，根据 query.json 文件中指定的令牌化 SQL AKIAIOSFODNN7EXAMPLE 的贡献筛选所有值。键还可能遵循与没有筛选条件的查询不同的顺序，因为前五个等待事件影响了筛选的 SQL。

检索 SQL 语句的全文

以下示例检索数据库实例 db-10BCD2EFGHIJ3KL4M5NO6PQRS5 的 SQL 语句的全文。--group 是 db.sql，--group-identifier 是 db.sql.id。在此示例中，*my-sql-id* 表示通过调用 pi get-resource-metrics 或 pi describe-dimension-keys 检索的 SQL ID。

运行以下命令。

对于 Linux、macOS 或 Unix：

```
aws pi get-dimension-key-details \
   --service-type RDS \
   --identifier db-10BCD2EFGHIJ3KL4M5NO6PQRS5 \
   --group db.sql \
   --group-identifier my-sql-id \
   --requested-dimensions statement
```

对于 Windows：

```
aws pi get-dimension-key-details ^
   --service-type RDS ^
   --identifier db-10BCD2EFGHIJ3KL4M5NO6PQRS5 ^
   --group db.sql ^
   --group-identifier my-sql-id ^
   --requested-dimensions statement
```

在此示例中，有维度详细信息。因此，Performance Insights 将检索 SQL 语句的全文，而不会将其截断。

```
{
    "Dimensions":[
    {
        "Value": "SELECT e.last_name, d.department_name FROM employees e, departments d
 WHERE e.department_id=d.department_id",
        "Dimension": "db.sql.statement",
        "Status": "AVAILABLE"
```

```
    },
    ...
    ]
}
```

创建一段时间的性能分析报告

以下示例为 db-loadtest-0 数据库创建了一个开始时间为 1682969503、结束时间为 1682979503 的性能分析报告。

```
aws pi create-performance-analysis-report \
        --service-type RDS \
        --identifier db-loadtest-0 \
        --start-time 1682969503 \
        --end-time 1682979503 \
        --region us-west-2
```

响应是报告的唯一标识符 report-0234d3ed98e28fb17。

```
{
   "AnalysisReportId": "report-0234d3ed98e28fb17"
}
```

检索性能分析报告

以下示例检索 report-0d99cc91c4422ee61 报告的分析报告详细信息。

```
aws pi get-performance-analysis-report \
--service-type RDS \
--identifier db-loadtest-0 \
--analysis-report-id report-0d99cc91c4422ee61 \
--region us-west-2
```

响应提供报告状态、ID、时间详细信息和见解。

```

        {
    "AnalysisReport": {
        "Status": "Succeeded",
        "ServiceType": "RDS",
        "Identifier": "db-loadtest-0",
        "StartTime": 1680583486.584,
```

```
        "AnalysisReportId": "report-0d99cc91c4422ee61",
        "EndTime": 1680587086.584,
        "CreateTime": 1680587087.139,
        "Insights": [
           ... (Condensed for space)
        ]
    }
}
```

列出数据库实例的所有性能分析报告

以下示例列出了 db-loadtest-0 数据库的所有可用的性能分析报告。

```
aws pi list-performance-analysis-reports \
--service-type RDS \
--identifier db-loadtest-0 \
--region us-west-2
```

响应列出了所有带有报告 ID、状态和时间段详细信息的报告。

```
{
    "AnalysisReports": [
        {
            "Status": "Succeeded",
            "EndTime": 1680587086.584,
            "CreationTime": 1680587087.139,
            "StartTime": 1680583486.584,
            "AnalysisReportId": "report-0d99cc91c4422ee61"
        },
        {
            "Status": "Succeeded",
            "EndTime": 1681491137.914,
            "CreationTime": 1681491145.973,
            "StartTime": 1681487537.914,
            "AnalysisReportId": "report-002633115cc002233"
        },
        {
            "Status": "Succeeded",
            "EndTime": 1681493499.849,
            "CreationTime": 1681493507.762,
            "StartTime": 1681489899.849,
            "AnalysisReportId": "report-043b1e006b47246f9"
        },
```

```
        {
            "Status": "InProgress",
            "EndTime": 1682979503.0,
            "CreationTime": 1682979618.994,
            "StartTime": 1682969503.0,
            "AnalysisReportId": "report-01ad15f9b88bcbd56"
        }
    ]
}
```

删除性能分析报告

以下示例删除 db-loadtest-0 数据库的分析报告。

```
aws pi delete-performance-analysis-report \
--service-type RDS \
--identifier db-loadtest-0 \
--analysis-report-id report-0d99cc91c4422ee61 \
--region us-west-2
```

向性能分析报告中添加标签

以下示例向 report-01ad15f9b88bcbd56 报告中添加了一个键为 name、值为 test-tag 的标签。

```
aws pi tag-resource \
--service-type RDS \
--resource-arn arn:aws:pi:us-west-2:356798100956:perf-reports/RDS/db-loadtest-0/
report-01ad15f9b88bcbd56 \
--tags Key=name,Value=test-tag \
--region us-west-2
```

列出性能分析报告的所有标签

以下示例列出了 report-01ad15f9b88bcbd56 报告的所有标签。

```
aws pi list-tags-for-resource \
--service-type RDS \
--resource-arn arn:aws:pi:us-west-2:356798100956:perf-reports/RDS/db-loadtest-0/
report-01ad15f9b88bcbd56 \
--region us-west-2
```

响应列出了添加到报告中的所有标签的值和键：

```
{
    "Tags": [
        {
            "Value": "test-tag",
            "Key": "name"
        }
    ]
}
```

从性能分析报告中删除标签

以下示例从 `report-01ad15f9b88bcbd56` 报告中删除 `name` 标签。

```
aws pi untag-resource \
--service-type RDS \
--resource-arn arn:aws:pi:us-west-2:356798100956:perf-reports/RDS/db-loadtest-0/
report-01ad15f9b88bcbd56 \
--tag-keys name \
--region us-west-2
```

删除标签后，调用 `list-tags-for-resource` API 不会列出此标签。

使用 AWS CloudTrail 记录 Performance Insights 调用

Performance Insights 与 AWS CloudTrail 一起运行，后者是一项服务，在 Performance Insights 中提供由用户、角色或 AWS 服务所执行的操作的记录。CloudTrail 将 Performance Insights 的所有 API 调用作为事件捕获。此捕获包括的调用来自 Amazon RDS 控制台以及来自对性能详情 API 操作的代码调用。

如果您创建跟踪，可以使 CloudTrail 事件持续传送到 Amazon S3 存储桶（包括 Performance Insights 的事件）。如果您不配置跟踪，则仍可在 CloudTrail 控制台中的事件历史记录中查看最新事件。通过使用 CloudTrail 收集的数据，您可以确定特定的信息。此信息包括向 Performance Insights 发出的请求、从中发出请求的 IP 地址、何人发出的请求以及请求的发出时间。它还包括其他详细信息。

要了解有关 CloudTrail 的更多信息，请参阅 AWS CloudTrail 用户指南。

在 CloudTrail 中使用性能详情信息

在您创建 AWS 账户时，将在该账户上启用 CloudTrail。当 Performance Insights 中发生活动时，该活动将记录在 CloudTrail 事件中，并与其他 AWS 服务事件一同保存在 CloudTrail 控制台的 Event

history (事件历史记录) 中。您可以在 AWS 账户中查看、搜索和下载最新事件。有关更多信息，请参阅 AWS CloudTrail 用户指南中的使用 CloudTrail 事件历史记录查看事件。

要持续记录 AWS 账户中的事件（包括 Performance Insights 的事件），请创建跟踪。通过跟踪，CloudTrail 可将日志文件传送至 Amazon S3 存储桶。预设情况下，在控制台中创建跟踪时，此跟踪应用于所有 AWS 区域。此跟踪记录来自 AWS 分区中的所有 AWS 区域的事件，并将日志文件传送至您指定的 Amazon S3 存储桶。此外，您可以配置其他 AWS 服务，进一步分析在 CloudTrail 日志中收集的事件数据并采取行动。有关更多信息，请参阅 AWS CloudTrail 用户指南中的以下主题：

- 创建跟踪概述
- CloudTrail 支持的服务和集成
- 为 CloudTrail 配置 Amazon SNS 通知
- 从多个区域接收 CloudTrail 日志文件和从多个账户接收 CloudTrail 日志文件

所有 Performance Insights 操作均由 CloudTrail 进行日志记录，您可以在 Performance Insights API 参考中找到这些操作的信息。例如，对 `DescribeDimensionKeys` 和 `GetResourceMetrics` 操作的调用将在 CloudTrail 日志文件中生成条目。

每个事件或日志条目都包含有关生成请求的人员信息。身份信息可帮助您确定以下内容：

- 请求是使用根用户凭证还是 IAM 用户凭证发出的。
- 请求是使用角色还是联合用户的临时安全凭证发出的。
- 请求是否由其他 AWS 服务发出。

有关更多信息，请参阅 CloudTrail userIdentity 元素。

性能详情日志文件条目

跟踪是一种配置，可用于将事件作为日志文件传送到您指定的 Amazon S3 存储桶。CloudTrail 日志文件包含一个或多个日志条目。事件 表示来自任何源的单个请求。每个事件包括有关所请求操作的信息、操作的日期和时间和请求参数等。CloudTrail 日志文件不是公用 API 调用的有序堆栈跟踪，因此它们不会按任何特定顺序显示。

下面的示例显示了一个 CloudTrail 日志条目，该条目说明了 `GetResourceMetrics` 操作。

```
{
    "eventVersion": "1.05",
```

```
    "userIdentity": {
        "type": "IAMUser",
         "principalId": "AKIAIOSFODNN7EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/johndoe",
        "accountId": "123456789012",
        "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
        "userName": "johndoe"
    },
    "eventTime": "2019-12-18T19:28:46Z",
    "eventSource": "pi.amazonaws.com",
    "eventName": "GetResourceMetrics",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "72.21.198.67",
    "userAgent": "aws-cli/1.16.240 Python/3.7.4 Darwin/18.7.0 botocore/1.12.230",
    "requestParameters": {
        "identifier": "db-YTDU5J5V66X7CXSCVDFD2V3SZM",
        "metricQueries": [
            {
                "metric": "os.cpuUtilization.user.avg"
            },
            {
                "metric": "os.cpuUtilization.idle.avg"
            }
        ],
        "startTime": "Dec 18, 2019 5:28:46 PM",
        "periodInSeconds": 60,
        "endTime": "Dec 18, 2019 7:28:46 PM",
        "serviceType": "RDS"
    },
    "responseElements": null,
    "requestID": "9ffbe15c-96b5-4fe6-bed9-9fccff1a0525",
    "eventID": "08908de0-2431-4e2e-ba7b-f5424f908433",
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```


性能详情 API 和接口 VPC 端点（AWS PrivateLink）

可以使用 AWS PrivateLink 在 VPC 和 Amazon RDS 性能详情之间创建私有连接。可以像在 VPC 中一样访问性能详情，而无需使用互联网网关、NAT 设备、VPN 连接或 AWS Direct Connect 连接。VPC 中的实例不需要公有 IP 地址即可访问性能详情。

您可以通过创建由 AWS PrivateLink 提供支持的接口端点来建立此私有连接。我们将在您为接口端点启用的每个子网中创建一个端点网络接口。这些是请求者托管的网络接口，用作发往性能详情的流量的入口点。

有关更多信息，请参阅《AWS PrivateLink 指南》中的通过 AWS PrivateLink 访问 AWS 服务。

性能详情的注意事项

在为性能详情设置接口端点之前，请首先查看《AWS PrivateLink 指南》中的 Considerations。

性能详情支持通过接口端点调用其所有 API 操作。

默认情况下，支持通过接口端点对性能详情进行完全访问。要控制通过接口端点流向性能详情的流量，请将安全组与端点网络接口关联。

可用性

性能详情 API 目前在支持性能详情的 AWS 区域中支持 VPC 端点。有关性能详情可用性的信息，请参阅支持 Performance Insights 的区域和 Aurora 数据库引擎。

为性能详情创建接口端点

可以使用 Amazon VPC 控制台或 AWS Command Line Interface（AWS CLI）为性能详情创建接口端点。有关更多信息，请参阅《AWS PrivateLink 指南》中的创建接口端点。

使用以下服务名称为性能详情创建接口端点：

如果为接口端点启用私有 DNS，则可使用其默认区域 DNS 名称向性能详情发出 API 请求。例如，`pi.us-east-1.amazonaws.com`。

为性能详情 API 创建 VPC 端点策略

端点策略是一种 IAM 资源，您可以将其附加到接口端点。默认端点策略支持通过接口端点完全访问性能详情。要控制支持从 VPC 访问性能详情的权限，请将自定义端点策略附加到接口端点。

端点策略指定以下信息：

- 可执行操作的主体（AWS 账户、IAM 用户和 IAM 角色）。
- 可执行的操作。
- 可对其执行操作的资源。

有关更多信息，请参阅《AWS PrivateLink 指南》中的使用端点策略控制对服务的访问权限。

示例：性能详情操作的 VPC 端点策略

以下是自定义端点策略的示例。将此策略附加到接口端点时，该策略会向所有主体授予对所有资源执行所列出的性能详情操作的访问权限。

```
{
   "Statement":[
      {
         "Principal":"*",
         "Effect":"Allow",
         "Action":[
            "rds:CreatePerformanceAnalysisReport",
            "rds:DeletePerformanceAnalysisReport",
            "rds:GetPerformanceAnalysisReport"
         ],
         "Resource":"*"
      }
   ]
}
```

示例：拒绝来自指定 AWS 账户的所有访问的 VPC 终端节点策略

以下 VPC 端点策略会拒绝 AWS 账户 123456789012 所有使用端点访问资源的权限。此策略允许来自其他账户的所有操作。

```
{
  "Statement": [
    {
      "Action": "*",
      "Effect": "Allow",
      "Resource": "*",
      "Principal": "*"
    },
    {
      "Action": "*",
      "Effect": "Deny",
      "Resource": "*",
      "Principal": { "AWS": [ "123456789012" ] }
     }
   ]
```

```
}
```

性能详情的 IP 寻址

IP 地址使 VPC 中的资源能够相互通信以及与 Internet 上的资源进行通信。性能详情同时支持 IPv4 和 IPv6 寻址协议。默认情况下，性能详情和 Amazon VPC 使用 IPv4 寻址协议。您无法关闭这种行为。创建 VPC 时，请确保指定 IPv4 CIDR 块 (一系列私有 IPv4 地址)。

可以选择将 IPv6 CIDR 块分配给 VPC 和子网，并将来自该块的 IPv6 地址分配给子网中的 RDS 资源。对 IPv6 协议的支持扩展了支持的 IP 地址数量。通过使用 IPv6 协议，您可以确保有足够的可用地址来应对 Internet 的未来发展。新的和现有 RDS 资源可以在 VPC 内使用 IPv4 和 IPv6 地址。在应用程序不同部分使用的两个协议之间配置、保护和转换网络流量可能会产生运营开销。您可以对 Amazon RDS 资源的 IPv6 协议进行标准化，以简化网络配置。有关服务端点和配额的更多信息，请参阅 Amazon Relational Database Service endpoints and quotas。

有关 Aurora IP 寻址的更多信息，请参阅 Aurora IP 寻址。

使用 Amazon DevOps Guru for Amazon RDS 分析 Aurora 性能异常

Amazon DevOps Guru 是一项完全托管式运营服务，可帮助开发人员和运营商提高应用程序的性能和可用性。DevOps Guru 卸下了与识别运营问题相关的任务，以便您可以快速实施改进应用程序的建议。有关更多信息，请参阅《Amazon DevOps Guru 用户指南》中的什么是 Amazon DevOps Guru？。

DevOps Guru 可检测、分析所有 Amazon RDS 数据库引擎的现有操作问题并提出建议。DevOps Guru for RDS 通过将机器学习应用于 Amazon Aurora 数据库的性能详情指标，扩展了此功能。这些监控功能使得适用于 RDS 的 DevOps Guru 可以检测和诊断性能瓶颈，并建议具体的纠正措施。DevOps Guru for RDS 还可以在您的 Aurora 数据库中出现问题状况之前检测出这些状况。

现在，您可以在 RDS 控制台中查看这些建议。有关更多信息，请参阅 来自 Amazon Aurora 的建议。

> **Important**
>
> Amazon DevOps Guru 不适用于 Aurora Serverless 数据库集群。

以下视频是关于 DevOps Guru for RDS 的概览。

有关此主题的深入分析，请参阅 Amazon DevOps Guru for RDS 深入剖析。

主题

- DevOps Guru for RDS 的优势
- 适用于 RDS 的 DevOps Guru 的工作原理
- 设置适用于 RDS 的 DevOps Guru

DevOps Guru for RDS 的优势

如果是您负责 Amazon Aurora 数据库，则您可能不知道正在发生影响该数据库的事件或回归。当您了解这个问题时，您可能不知道为什么会发生这个问题，也不知道该怎么处理它。您可以遵循 DevOps Guru for RDS 的建议，而不是向数据库管理员 (DBA) 寻求帮助或依赖第三方工具。

从 DevOps Guru for RDS 的详细分析中，您可以获得以下优势：

快速诊断

DevOps Guru for RDS 会持续监控和分析数据库遥测。性能详情、增强监控和 Amazon CloudWatch 会收集数据库集群的遥测数据。DevOps Guru for RDS 会使用统计和机器学习技术来挖掘这些数据并检测异常情况。要了解有关遥测数据的更多信息，请参阅《Amazon Aurora 用户指南》中的在 Amazon Aurora 上使用性能详情监控数据库负载和使用增强监控来监控操作系统指标

快速解决方案

每个异常情况都会识别性能问题，并建议调查或纠正措施的途径。例如，DevOps Guru for RDS 可能会建议您调查特定的等待事件。或者，它可能建议您优化应用程序池设置以限制数据库连接的数量。根据这些建议，您可以比手动进行故障排除更快地解决性能问题。

主动见解

DevOps Guru for RDS 使用资源中的指标来检测潜在的问题行为，以免其成为更大的问题。例如，此服务可以检测数据库何时使用越来越多的磁盘临时表，这可能会开始影响性能。然后，DevOps Guru 会提供建议以帮助解决问题，从而防止问题变成更大的问题。

深入了解 Amazon 工程师和机器学习

为了检测性能问题并帮助您解决瓶颈，DevOps Guru for RDS 依赖于机器学习（ML）和高级数据公式。Amazon 数据库工程师为 DevOps Guru for RDS 结果的开发做出了贡献，该服务封装了多年来管理成千上万个数据库的经验。通过利用这些集体知识，适用于 RDS 的 DevOps Guru 可以教您最佳实践。

适用于 RDS 的 DevOps Guru 的工作原理

DevOps Guru for RDS 从 Amazon RDS 性能详情中收集有关 Aurora 数据库的数据。最重要的指标是 `DBLoad`。适用于 RDS 的 DevOps Guru 使用性能详情指标，使用机器学习对其进行分析，并将洞察发布到控制面板。

洞察是 DevOps Guru 检测到的相关异常的集合。

在 DevOps Guru for RDS 中，异常是一种与 Amazon Aurora 数据库的正常性能有所偏差的模式。

主动见解

主动见解可以让您在问题发生之前了解问题行为。它包含异常情况以及建议和相关指标，可以帮助您解决 Amazon Aurora 数据库中的问题，以免问题变得更严重。这些见解发布在 DevOps Guru 控制面板上。

例如，DevOps Guru 可能会检测到 Aurora PostgreSQL 数据库正在创建许多磁盘上的临时表。如果不加以解决，这种趋势可能会导致性能问题。每项主动见解都包含有关纠正行为的建议以及指向使用 Amazon DevOps Guru 主动见解优化 Aurora MySQL或使用 Amazon DevOps Guru 主动见解优化 Aurora PostgreSQL中相关主题的链接。有关更多信息，请参阅《Amazon DevOps Guru 用户指南》中的在 DevOps Guru 中使用见解。

被动见解

被动见解可在异常行为发生时识别此类行为。如果 DevOps Guru for RDS 发现您的 Amazon Aurora 数据库实例中的性能问题，它会在 DevOps Guru 控制面板中发布被动见解。有关更多信息，请参阅《Amazon DevOps Guru 用户指南》中的在 DevOps Guru 中使用见解。

因果异常

因果异常是被动见解内的一项顶级异常。数据库加载（数据库加载）是适用于 RDS 的 DevOps Guru 的因果异常。

异常情况通过分配严重性级别 High（高）、Medium（中）或 Low（低）来衡量性能影响。要了解详情，请参阅《Amazon DevOps Guru 用户指南》中的适用于 RDS 的 DevOps Guru 的关键概念。

如果 DevOps Guru 检测到数据库实例上的当前异常情况，则会在 RDS 控制台的 Databases（数据库）页面中收到提示。控制台还会提示您注意过去 24 小时内发生的异常。要从 RDS 控制台转到异常页面，请选择警报消息中的链接。RDS 控制台还会在 Amazon Aurora 数据库集群的页面中向您发出提示。

上下文异常

上下文异常是数据库负载（DB 负载）内与被动见解相关的一项调查结果。每个上下文异常都描述了需要调查的特定 Amazon Aurora 性能问题。例如，适用于 RDS 的 DevOps Guru 可能会建议您考虑增加 CPU 容量或调查导致数据库负载的等待事件。

> **Important**
>
> 我们建议您在修改生产实例之前在测试实例上测试所有更改。通过这种方式，您可以了解更改的影响。

要了解详情，请参阅《Amazon DevOps Guru 用户指南》中的分析 Amazon RDS 中的异常。

设置适用于 RDS 的 DevOps Guru

要允许 DevOps Guru for Amazon RDS 发布 Amazon Aurora 数据库的见解，请完成以下任务。

主题

- 配置适用于 RDS 的 DevOps Guru 的 IAM 访问策略
- 为您的 Aurora 数据库实例开启性能详情
- 打开 DevOps Guru 并指定资源覆盖范围

配置适用于 RDS 的 DevOps Guru 的 IAM 访问策略

要在 RDS 控制台中查看来自 DevOps Guru 的提示，您的 AWS Identity and Access Management (IAM) 用户或角色必须具有以下任一策略：

- AWS 托管策略 `AmazonDevOpsGuruConsoleFullAccess`
- AWS 托管式策略 `AmazonDevOpsGuruConsoleReadOnlyAccess` 以及以下任一策略：
  - AWS 托管策略 `AmazonRDSFullAccess`
  - 客户托管策略包括 `pi:GetResourceMetrics` 和 `pi:DescribeDimensionKeys`

有关更多信息，请参阅 为 Performance Insights 配置访问策略。

为您的 Aurora 数据库实例开启性能详情

适用于 RDS 的 DevOps Guru 依靠性能详情来获取其数据。如果没有性能详情，DevOps Guru 会发布异常情况，但不包括详细的分析和建议。

创建 Aurora 数据库集群或修改集群实例时，可以开启性能详情。有关更多信息，请参阅 为 Aurora 打开和关闭性能详情。

打开 DevOps Guru 并指定资源覆盖范围

您可以开启 DevOps Guru，以让它通过以下任一方式监控您的 Amazon Aurora 数据库。

主题

- 在 RDS 控制台中开启 DevOps Guru
- 在 DevOps Guru 控制台中添加 Aurora 资源

- 使用 AWS CloudFormation 添加 Aurora 资源

在 RDS 控制台中开启 DevOps Guru

您可以在 Amazon RDS 控制台中通过多个途径来开启 DevOps Guru。

主题

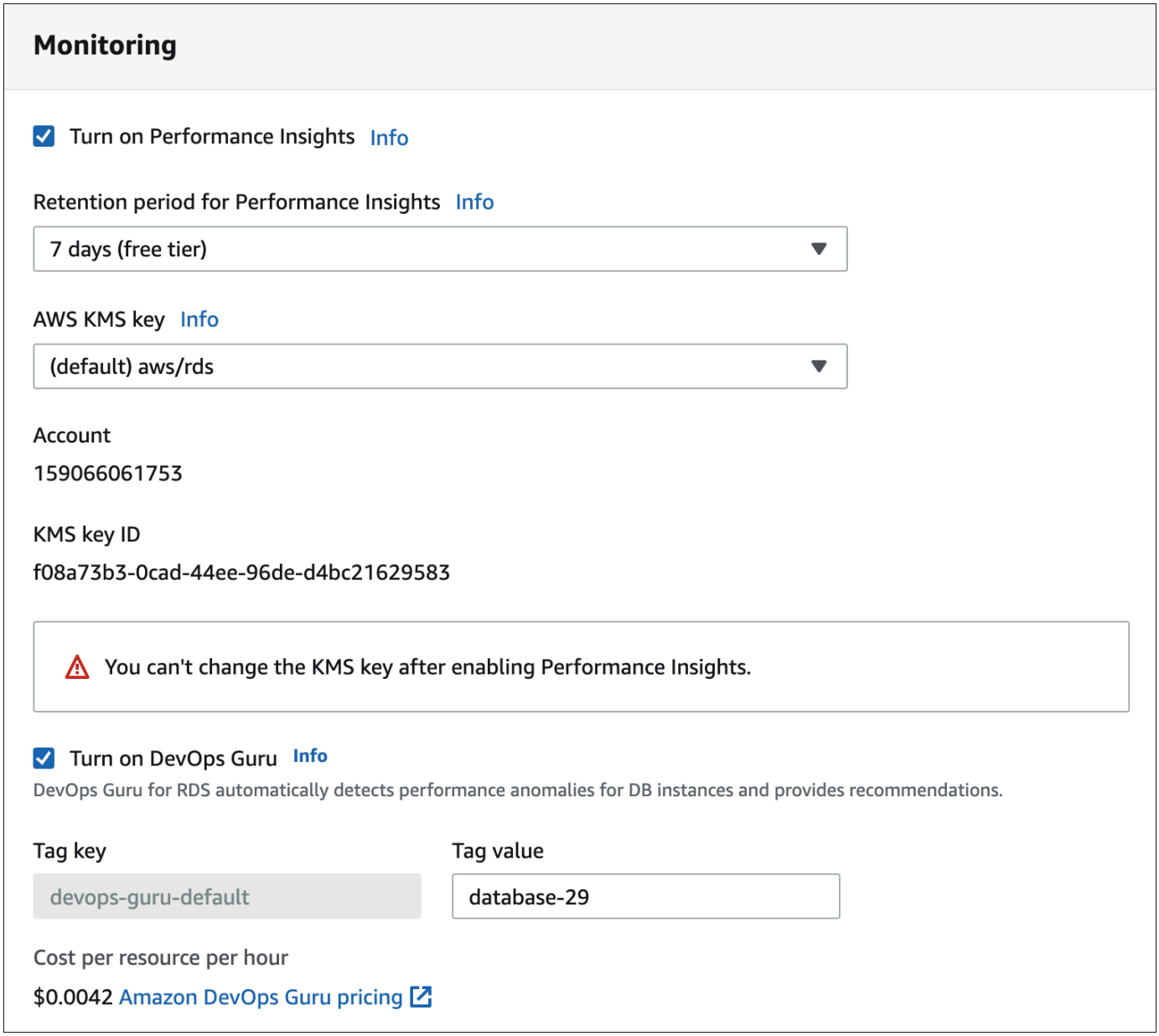
- 创建 Aurora 数据库时开启 DevOps Guru
- 从通知横幅开启 DevOps Guru
- 响应开启 DevOps Guru 时的权限错误

创建 Aurora 数据库时开启 DevOps Guru

创建工作流包括一个设置，该设置可为数据库开启 DevOps Guru 覆盖范围。原定设置情况下，当您选择 Production（生产）模板时，此设置处于开启状态。

创建 Aurora 数据库时开启 DevOps Guru

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 按创建数据库集群中的步骤操作，直至（但不包括）您选择监控设置的步骤。
3. 在 Monitoring（监控）中，选择 Turn on Performance Insights（开启性能详情）。为让 DevOps Guru for RDS 提供对性能异常情况的详细分析，必须开启性能详情。
4. 选择 Turn on DevOps Guru（开启 DevOps Guru）。



5. 为数据库创建一个标签，以便 DevOps Guru 可以对其进行监控。执行以下操作：

   - 在 Tag key（标签键）的文本字段中，输入以 **`Devops-Guru-`** 开头的名称。
   - 在 Tag key（标签键）的文本字段中，输入任意值。例如，如果输入 **`rds-database-1`** 作为 Aurora 数据库的名称，还可以输入 **`rds-database-1`** 作为标签值。

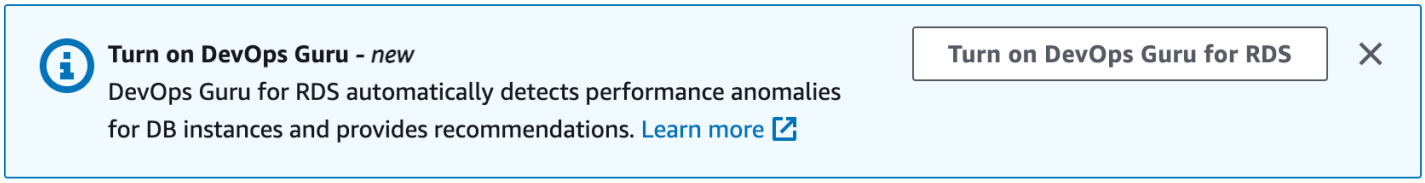
   有关标签的更多信息，请参阅《Amazon DevOps Guru 用户指南》中的“使用标签识别 DevOps Guru 应用程序中的资源”。

6. 完成创建数据库集群 中的其余步骤。

从通知横幅开启 DevOps Guru

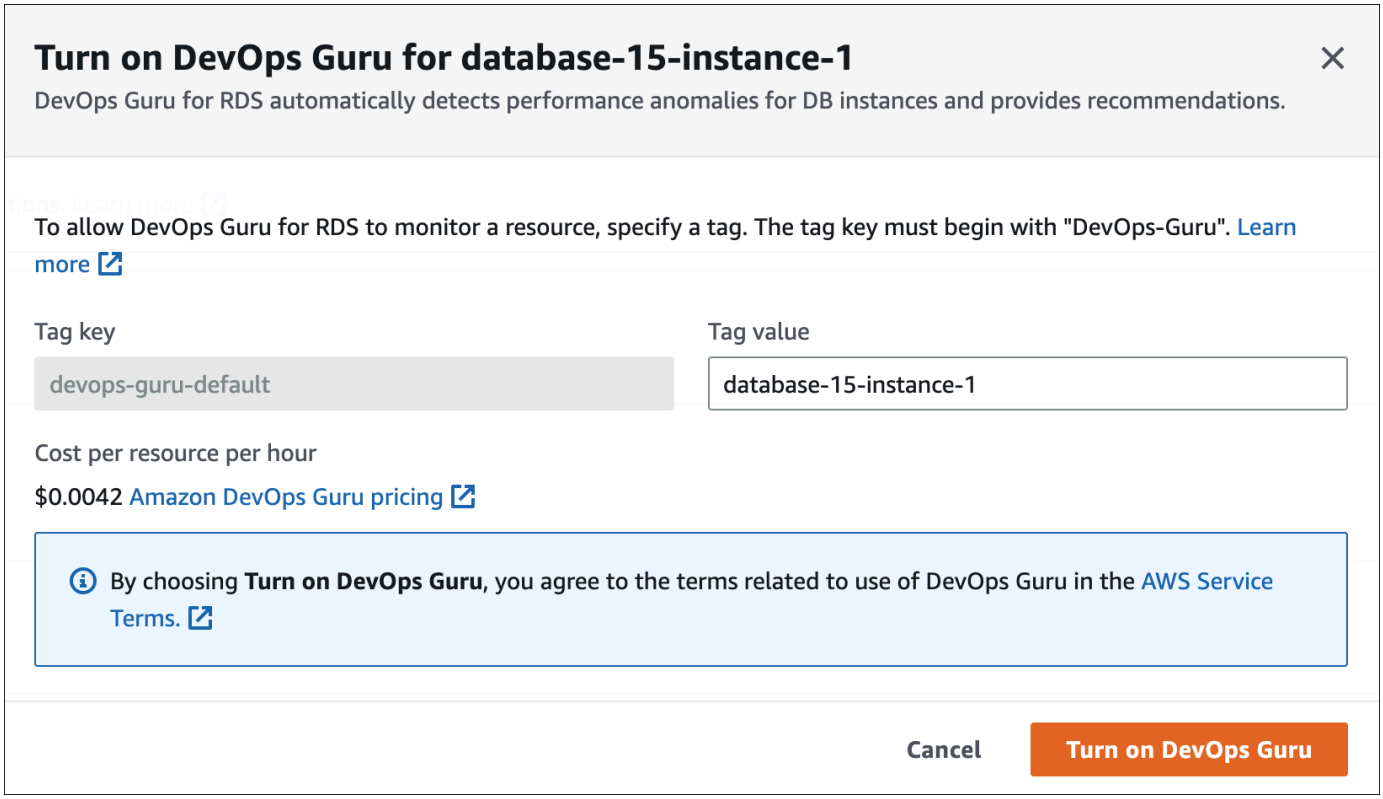
如果 DevOps Guru 未涵盖您的资源，Amazon RDS 会在以下位置通过横幅通知您：

- 数据库集群实例的 Monitoring（监控）选项卡
- 性能详情控制面板



为 Aurora 数据库开启 DevOps Guru

1. 在横幅中，选择 Turn on DevOps Guru for RDS（开启适用于 RDS 的 DevOps Guru）。
2. 输入标签键名称和值。有关标签的更多信息，请参阅《Amazon DevOps Guru 用户指南》中的“使用标签识别 DevOps Guru 应用程序中的资源”。



3. 选择 Turn on DevOps Guru（开启 DevOps Guru）。

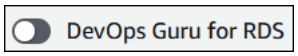
响应开启 DevOps Guru 时的权限错误

如果您在创建数据库时从 RDS 控制台开启 DevOps Guru，RDS 可能会显示以下关于缺少权限的横幅。

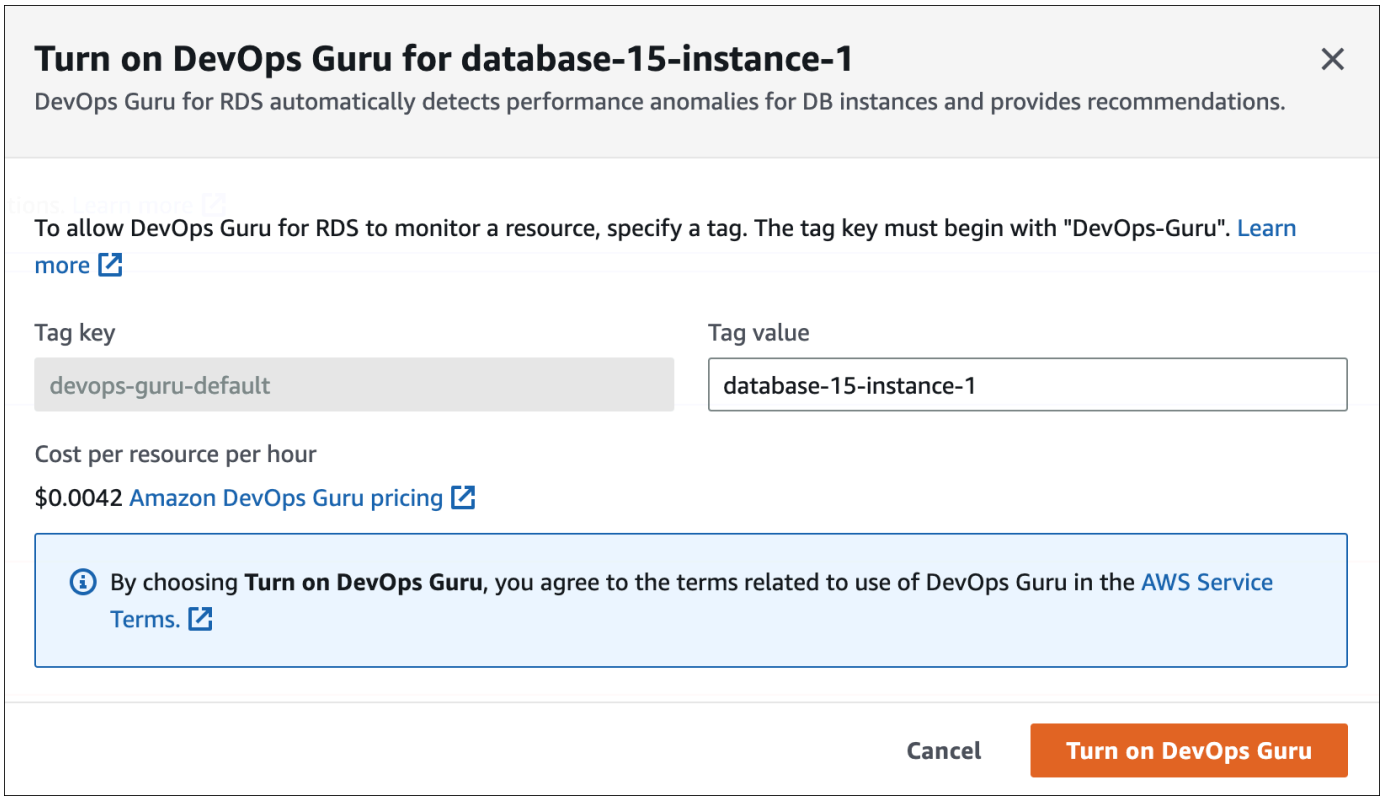


响应权限错误

1. 向您的 IAM 用户或角色授予用户托管式角色 `AmazonDevOpsGuruConsoleFullAccess`。有关更多信息，请参阅 配置适用于 RDS 的 DevOps Guru 的 IAM 访问策略。
2. 打开 RDS 控制台。
3. 在导航窗格中，选择 Performance Insights。
4. 在刚创建的集群中选择一个数据库实例。
5. 选择此开关以开启 DevOps Guru for RDS。

6. 选择标签值。有关更多信息，请参阅《Amazon DevOps Guru 用户指南》中的“使用标签识别 DevOps Guru 应用程序中的资源”。



7. 选择 Turn on DevOps Guru（开启 DevOps Guru）。

在 DevOps Guru 控制台中添加 Aurora 资源

您可以在 DevOps Guru 控制台上指定 DevOps Guru 资源覆盖范围。按照《Amazon DevOps Guru 用户指南》中的指定 DevOps Guru 资源覆盖范围中描述的步骤操作。编辑所分析的资源时，请选择以下选项之一：

- 选择所有账户资源以分析所有受支持的资源，包括您的 AWS 账户和区域中的 Aurora 数据库。
- 选择 CloudFormation 堆栈来分析您选择的堆栈中的 Aurora 数据库。有关更多信息，请参阅《Amazon DevOps Guru 用户指南》中的使用 AWS CloudFormation 堆栈识别 DevOps Guru 应用程序中的资源。
- 选择标签来分析您已标记的 Aurora 数据库。有关更多信息，请参阅《Amazon DevOps Guru 用户指南》中的使用标签识别 DevOps Guru 应用程序中的资源。

有关更多信息，请参阅《Amazon DevOps Guru 用户指南》中的启用 DevOps Guru。

使用 AWS CloudFormation 添加 Aurora 资源

可以使用标签向您的 CloudFormation 模板添加 Aurora 资源的覆盖范围。以下过程假设您具有同时适用于 Aurora 数据库实例和 DevOps Guru 堆栈的 CloudFormation 模板。

使用 CloudFormation 标签指定 Aurora 数据库实例

1. 在数据库实例的 CloudFormation 模板中，使用键/值对定义标签。

   以下示例将 Aurora 数据库实例的值 `Devops-guru-cfn-default` 分配给 `my-aurora-db-instance1`。

```
MyAuroraDBInstance1:
  Type: "AWS::RDS::DBInstance"
  Properties:
    DBClusterIdentifier: my-aurora-db-cluster
    DBInstanceIdentifier: my-aurora-db-instance1
    Tags:
      - Key: Devops-guru-cfn-default
        Value: devopsguru-my-aurora-db-instance1
```

2. 在 DevOps Guru 堆栈的 CloudFormation 模板中，在资源收集筛选条件中指定相同的标签。

以下示例将 DevOps Guru 配置为使用标签值 `my-aurora-db-instance1` 为资源提供覆盖范围。

```
DevOpsGuruResourceCollection:
  Type: AWS::DevOpsGuru::ResourceCollection
  Properties:
    ResourceCollectionFilter:
      Tags:
        - AppBoundaryKey: "Devops-guru-cfn-default"
          TagValues:
          - "devopsguru-my-aurora-db-instance1"
```

以下示例涵盖了应用程序边界 `Devops-guru-cfn-default` 内的所有资源。

```
DevOpsGuruResourceCollection:
  Type: AWS::DevOpsGuru::ResourceCollection
  Properties:
    ResourceCollectionFilter:
      Tags:
        - AppBoundaryKey: "Devops-guru-cfn-default"
          TagValues:
          - "*"
```

有关更多信息，请参阅《AWS CloudFormation 用户指南》中的 AWS::DevOpsGuru::ResourceCollection 和 AWS::RDS::DBInstance。

使用增强监控来监控操作系统指标

利用增强监测功能，您可以实时监控数据库实例的操作系统。若您想了解不同进程或线程对 CPU 的使用差异，增强监测指标非常有用。

主题

- 增强监测概述
- 设置和启用增强监控
- 在 RDS 控制台中查看操作系统指标
- 使用 CloudWatch Logs 查看操作系统指标

增强监测概述

Amazon RDS 为数据库实例运行的操作系统 (OS) 实时提供指标。您可以在控制台上查看 RDS 数据库实例的所有系统指标和过程信息。您可以管理要为每个实例监控哪些指标，并根据您的要求自定义控制面板。有关增强监控指标的说明，请参阅 增强监控中的操作系统指标。

RDS 将增强监测中的指标传输到您的 Amazon CloudWatch Logs 账户。您可以在 CloudWatch Logs 中创建指标筛选条件，并在 CloudWatch 控制面板上显示图表。此外，您可以在选择的监控系统中通过 Amazon CloudWatch Logs 使用增强监测 JSON 输出。有关更多信息，请参阅 Amazon RDS 常见问题中的增强监测。

主题

- CloudWatch 与增强监控指标的区别
- 保留增强监测指标
- 增强监测的成本

CloudWatch 与增强监控指标的区别

虚管理程序创建并运行虚拟机 (VM)。借助管理程序，实例可以通过虚拟共享内存和 CPU 支持多个来宾虚拟机。CloudWatch 从数据库实例管理程序收集关于 CPU 使用率的指标。而增强监测则从数据库实例上的代理收集这一指标。

您可能会发现 CloudWatch 和增强监测测量值之间存在差异，因为管理程序层执行的工作较少。如果数据库实例使用较小的实例类，差异可能会更大。在这种情况下，单个物理实例上的管理程序层可能会管理更多的虚拟机 (VM)。

有关增强监控指标的说明，请参阅 增强监控中的操作系统指标。有关 Amazon CloudWatch 的更多信息，请参阅 Amazon CloudWatch 用户指南。

保留增强监测指标

默认情况下，增强监测指标可在 CloudWatch Logs 中存储达 30 天。此保留期与典型的 CloudWatch 指标不同。

要修改指标存储在 CloudWatch Logs 中的时间量，请在 CloudWatch 控制台中更改 `RDSOSMetrics` 日志组的保留期。有关更多信息，请参阅 Amazon CloudWatch Logs User Guide 中的更改 CloudWatch Logs 中的日志数据保留期。

增强监测的成本

增强监测指标存储在 CloudWatch Logs 中，而不是存储在 CloudWatch 指标中。增强监测的成本取决于以下因素：

- 您需要为 Amazon CloudWatch Logs 提供的免费套餐之外的增强监测付费。收费是根据 CloudWatch Logs 数据传输和存储费率计算的。
- RDS 实例传输的信息量与为增强型监测功能定义的精细程度成正比。监控间隔越短，操作系统指标报告频率越高，监控成本也就越高。要管理成本，请为账户中的不同实例设置不同的精细度。
- 增强监测的使用成本适用于启用了增强监测的每个数据库实例。监控大量数据库实例的成本要高于监控少量数据库实例。
- 数据库实例支持的工作负载计算越密集，要报告的操作系统进程活动就越多，增强监测的成本也越高。

有关定价的更多信息，请参阅 Amazon CloudWatch 定价。

设置和启用增强监控

要使用增强监控，您必须创建 IAM 角色，然后启用增强监控。

主题

- 为增强监控创建 IAM 角色
- 打开和关闭增强监控
- 防范混淆代理问题

为增强监控创建 IAM 角色

增强监测需要代表您执行操作的权限来向 CloudWatch Logs发送操作系统指标信息。您使用 AWS Identity and Access Management（IAM）角色授予增强监控权限。您可以在启用增强监控时创建此角色，也可以事先创建该角色。

主题

- 在启用增强监控时创建 IAM 角色
- 在启用增强监控之前创建 IAM 角色

在启用增强监控时创建 IAM 角色

在 RDS 控制台中启用增强监控后，Amazon RDS 可以为您创建所需的 IAM 角色。该角色命名为 `rds-monitoring-role`。RDS 将此角色用于指定的数据库实例、只读副本或多可用区数据库集群。

在启用增强监控时创建 IAM 角色的方法

1. 按打开和关闭增强监控中的步骤操作。
2. 在选择角色的步骤中，将监视角色设置为默认值。

在启用增强监控之前创建 IAM 角色

您可以在启用增强监控之前创建所需的角色。在启用增强监控时，请指定新角色的名称。如果使用 AWS CLI 或 RDS API 启用增强监测，则必须创建此必需角色。

必须向启用增强监控的用户授予 `PassRole` 权限。有关更多信息，请参阅 IAM 用户指南的授予向 AWS 服务传递角色的用户权限中的示例 2。

为 Amazon RDS 增强监控创建 IAM 角色

1. 通过以下网址打开 IAM 控制台：https://console.aws.amazon.com。
2. 在导航窗格中，选择 Roles (角色)。
3. 选择创建角色。
4. 选择 AWS service（Amazon Web Services 服务）选项卡，然后从服务列表中选择 RDS。
5. 选择 RDS - Enhanced Monitoring（RDS - 增强监测），然后选择 Next（下一步）。
6. 确保 Permissions policies（权限策略）显示 AmazonRDSEnhancedMonitoringRole，然后选择 Next（下一步）。

7. 对于角色名称，请为您的角色输入一个名称。例如，输入 **emaccess**。

   您角色的可信实体是 AWS 服务 monitoring.rds.amazonaws.com。

8. 选择创建角色。

打开和关闭增强监控

可以使用 AWS Management Console、AWS CLI 或 RDS API 管理增强监控。可以为每个数据库集群上的指标收集设置不同的粒度。也可以从控制台切换现有数据库集群的增强监控。

控制台

您可以在创建数据库集群或只读副本，或者在修改数据库集群时打开增强监控。如果修改数据库实例或集群以开启增强监控，则不需要重启数据库实例，更改也会生效。

在 Databases（数据库）页面执行以下某种操作时，您可以在 RDS 控制台中打开增强监控：

- 创建数据库集群 - 选择 Create database（创建数据库）。
- Create a read replica（创建只读副本）：选择 Actions（操作），然后选择 Create read replica（创建只读副本）。
- 修改数据库实例或数据库集群：选择修改。

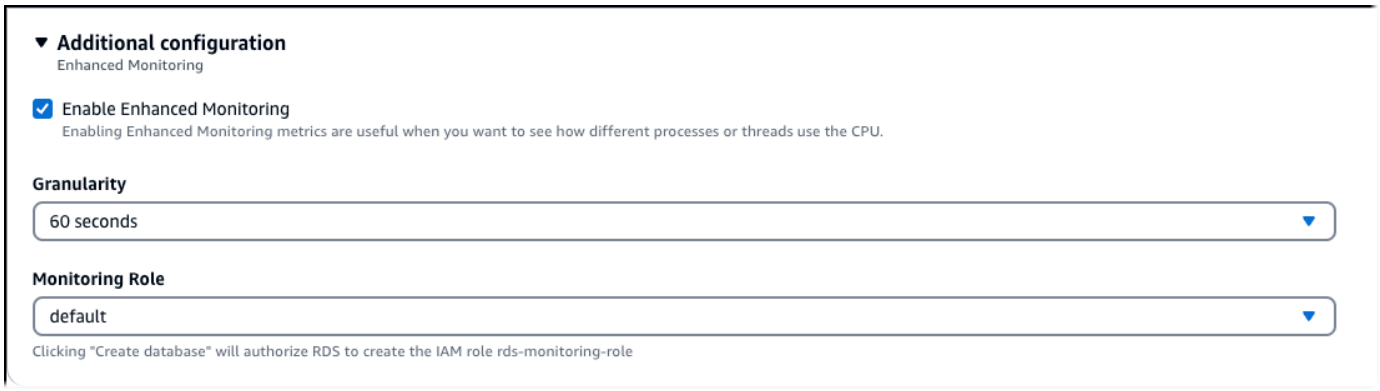
> **Note**
>
> 当您为数据库集群启用增强监控时，您无法管理集群中各个数据库实例的增强监控。

如果您正在为数据库集群中的各个数据库实例管理增强监控，并且对于不同的实例将粒度设置为不同的值，则数据库集群在增强监控方面是异构的。在此类情况下，无法修改数据库集群来在集群级别管理增强监控。

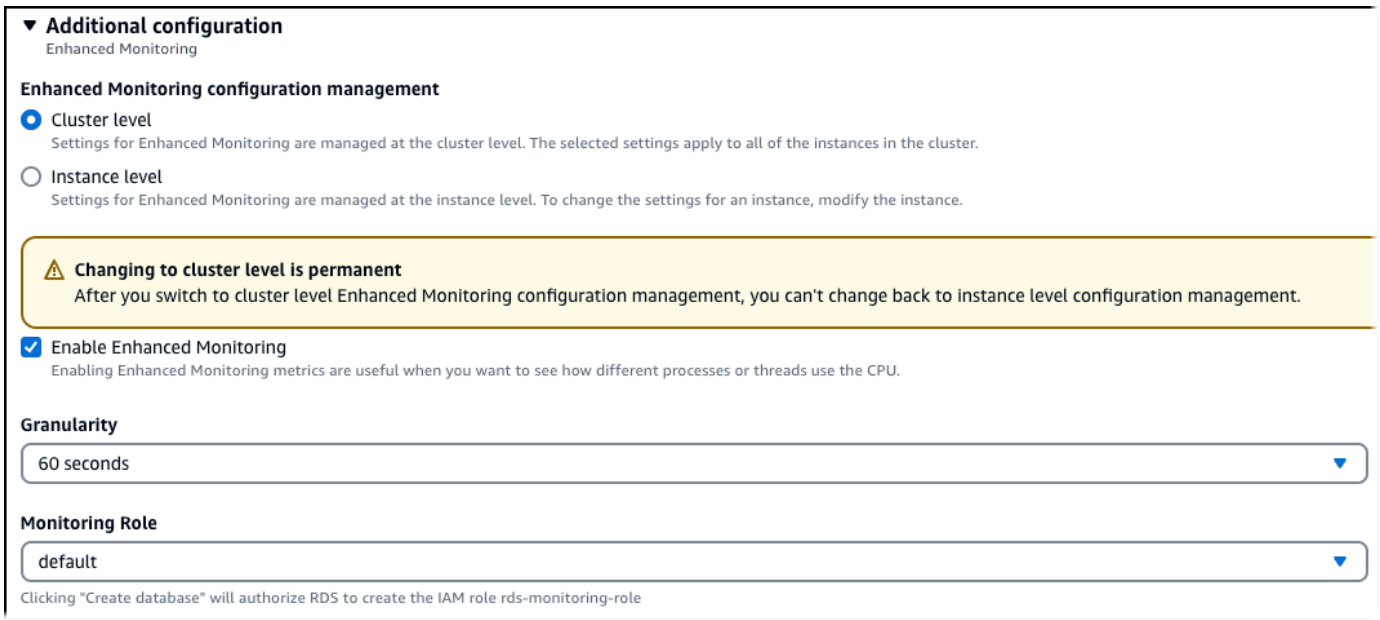
在 RDS 控制台中打开或关闭增强监控

1. 滚动到 Additional configuration（其他配置）。
2. 在监控中，为数据库集群或只读副本选择启用增强监控。在集群级别启用增强监控，可让您在集群级别管理增强监控设置和选项。集群级别设置适用于集群中的所有数据库实例。取消选择该选项可在集群级别禁用增强监控。可以稍后修改集群中各个数据库实例的增强监控设置。

在创建数据库页面中，可以选择在集群级别开启增强监控。



如果您在创建集群时未启用增强监控，则可以在修改数据库集群页面中修改集群。



> **Note**
>
> 在已在集群级别管理增强监控的数据库集群中，您无法管理单个数据库实例的增强监控。

3. 如果集群在增强监控方面是异构的，则不能选择在集群级别管理增强监控。要在集群级别管理增强监控，请更改每个实例的增强监控设置，使其匹配。现在，在修改集群时，可以选择在集群级别管理增强监控。
4. 将 Monitoring Role 属性设置为您创建的 IAM 角色以允许 Amazon RDS 代表您与 Amazon CloudWatch Logs 通信，或选择 Default 让 RDS 为您创建一个名为 `rds-monitoring-role` 的角色。

5. 将粒度属性设置成两次为数据库实例、数据库集群或只读副本收集指标之间的间隔，以秒为单位。Granularity 属性可以设置为以下值之一：1、5、10、15、30 或 60。

   RDS 控制台最快每 5 秒刷新一次。如果您在 RDS 控制台中将粒度设置为 1 秒，仍然会看到指标每 5 秒更新一次。使用 CloudWatch Logs 可以获得 1 秒的指标更新。

AWS CLI

要使用 AWS CLI 打开增强监控，请在以下命令中将 `--monitoring-interval` 选项设置为 `0` 以外的值，并将 `--monitoring-role-arn` 选项设置为您在 为增强监控创建 IAM 角色 中创建的角色。

- create-db-cluster
- modify-db-cluster
- create-db-instance
- create-db-instance-read-replica
- modify-db-instance

`--monitoring-interval` 选项指定收集增强监控指标的时间点之间的间隔，以秒为单位。选项的有效值为 `0`、`1`、`5`、`10`、`15`、`30`、和 `60`。

要使用 AWS CLI 关闭增强监控，请在这些命令中将 `--monitoring-interval` 选项设置为 `0`。

Example

以下示例将打开数据库实例的增强监控：

对于 Linux、macOS 或 Unix：

```
aws rds modify-db-instance \
    --db-instance-identifier mydbinstance \
    --monitoring-interval 30 \
    --monitoring-role-arn arn:aws:iam::123456789012:role/emaccess
```

对于 Windows：

```
aws rds modify-db-instance ^
    --db-instance-identifier mydbinstance ^
    --monitoring-interval 30 ^
    --monitoring-role-arn arn:aws:iam::123456789012:role/emaccess
```

Example

以下示例将开启数据库集群的增强监控：

对于 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbinstance \
    --monitoring-interval 30 \
    --monitoring-role-arn arn:aws:iam::123456789012:role/emaccess
```

对于 Windows：

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbinstance ^
    --monitoring-interval 30 ^
    --monitoring-role-arn arn:aws:iam::123456789012:role/emaccess
```

Example

以下示例将打开多可用区数据库集群的增强监控：

对于 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --monitoring-interval 30 \
    --monitoring-role-arn arn:aws:iam::123456789012:role/emaccess
```

对于 Windows：

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --monitoring-interval 30 ^
    --monitoring-role-arn arn:aws:iam::123456789012:role/emaccess
```

RDS API

要使用 RDS API 打开增强监控，请将 `MonitoringInterval` 参数设置为 `0` 以外的值，并将 `MonitoringRoleArn` 参数设置为您在 为增强监控创建 IAM 角色 中创建的角色。在以下操作中设置这些参数：

- CreateDBCluster
- ModifyDBCluster
- CreateDBInstance
- CreateDBInstanceReadReplica
- ModifyDBInstance

MonitoringInterval 参数指定收集增强监控指标的时间点之间的间隔，以秒为单位。有效值为 0、1、5、10、15、30 和 60。

要使用 RDS API 关闭增强监控，请将 MonitoringInterval 设置为 0。

防范混淆代理问题

混淆代理问题是一个安全性问题，即不具有某操作执行权限的实体可能会迫使具有更高权限的实体执行该操作。在 AWS 中，跨服务模拟可能会导致混淆代理问题。一个服务（呼叫服务）调用另一项服务（所谓的服务）时，可能会发生跨服务模拟。可以操纵调用服务，使用其权限以在其他情况下该服务不应有访问权限的方式对另一个客户的资源进行操作。为防止这种情况，AWS 提供可帮助您保护所有服务的数据的工具，而这些服务中的服务主体有权限访问账户中的资源。有关更多信息，请参阅混淆代理问题。

要限制 Amazon RDS 授予另一项服务对资源的访问权限，建议在增强监控角色的信任策略中使用 aws:SourceArn 和 aws:SourceAccount 全局条件上下文键。如果使用两个全局条件上下文键，则这两个键必须使用相同的账户 ID。

防范混淆代理问题最有效的方法是使用 aws:SourceArn 全局条件上下文键和资源的完整 ARN。对于 Amazon RDS，请将 aws:SourceArn 设置为 arn:aws:rds:*Region*:*my-account-id*:db:*dbname*。

以下示例在信任策略中使用 aws:SourceArn 和 aws:SourceAccount 全局条件上下文键来防范混淆代理问题。

JSON

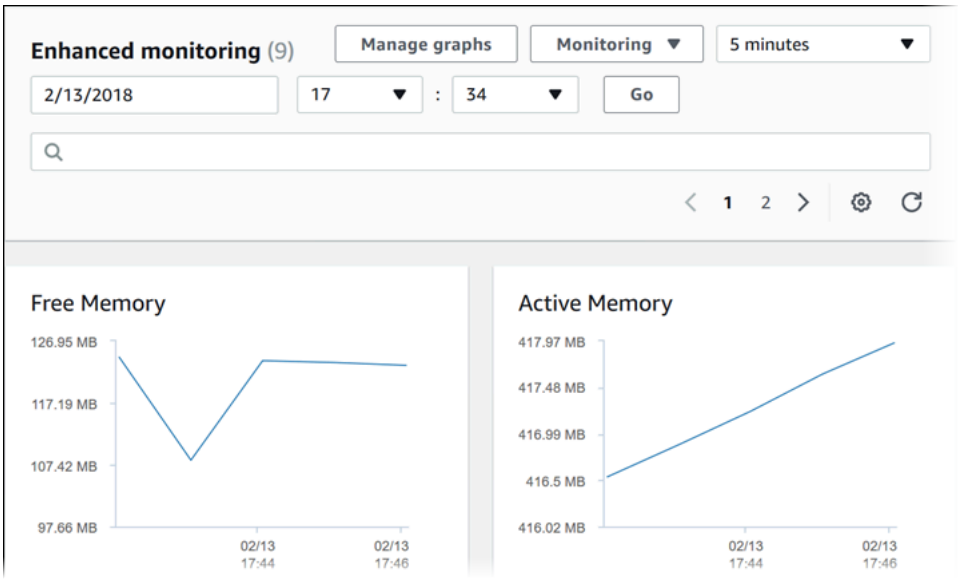
```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
```

```
      "Principal": {
        "Service": "monitoring.rds.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringLike": {
          "aws:SourceArn": "arn:aws:rds:Region:my-account-id:db:dbname"
        },
        "StringEquals": {
          "aws:SourceAccount": "my-account-id"
        }
      }
    }
  ]
}
```

在 RDS 控制台中查看操作系统指标

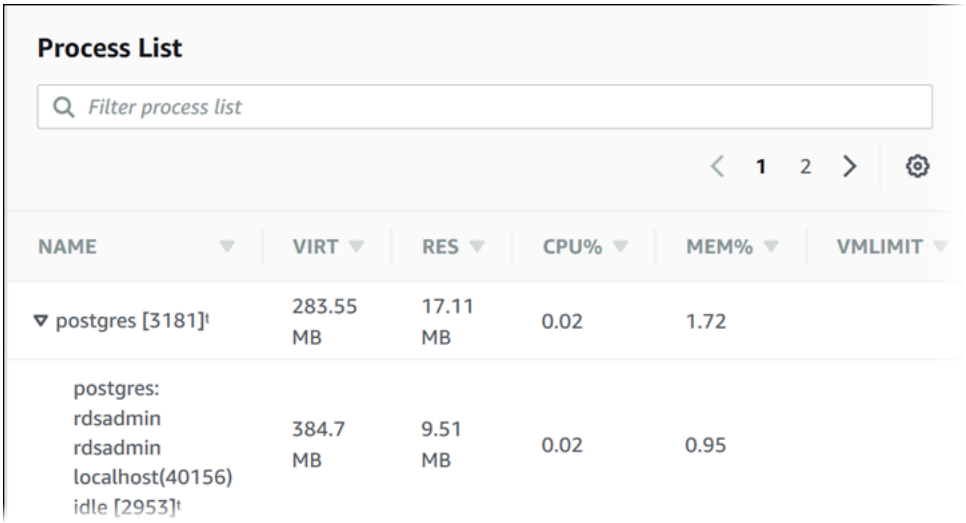
您可以在 RDS 控制台中查看增强监测报告的操作系统指标，方法是为监测选择 Enhanced monitoring (增强监测)。

以下示例显示“Enhanced Monitoring（增强监控）”页面。有关增强监控指标的说明，请参阅 增强监控中的操作系统指标。



如果要查看运行在数据库实例上的进程的详细信息，请为监测选择 OS 进程列表。

Process List (进程列表) 视图如下所示。



Process list (进程列表) 视图中所示的增强监测指标按以下方式组织：

- RDS child processes (RDS 子进程) – 显示支持数据库实例的 RDS 进程的摘要，例如 `aurora`（对于 Amazon Aurora 数据库集群）和 。进程线程在父进程下嵌套显示。进程线程仅显示 CPU 使用率，因为进程所有线程的其他指标都相同。控制台最多显示 100 个进程和线程。因此，显示结果将是占用最多 CPU 和内存的进程和线程。如果有超过 50 个进程和超过 50 个线程，则控制台只会显示每种类别的前 50 个。这种显示方式有助于您了解哪些进程对性能影响最大。
- RDS processes (RDS 进程) – 摘要显示 RDS 管理代理所用的资源、诊断监控进程以及支持 RDS 数据库实例所需的其他AWS进程。
- OS processes – 显示内核和系统进程摘要，这些进程通常对性能影响最小。

对每个进程列出的项目有：

- VIRT – 显示进程的虚拟大小。
- RES – 显示进程正在使用的实际物理内存。
- CPU% – 显示进程正在使用的总 CPU 带宽的百分比。
- MEM% – 显示进程使用的总内存的百分比。

RDS 控制台中显示的监控数据是从 Amazon CloudWatch Logs 中检索的。您还可以从 CloudWatch Logs 中检索数据库实例的指标作为日志流。有关更多信息，请参阅“使用 CloudWatch Logs 查看操作系统指标”。

以下情况下不会返回增强监测指标：

- 数据库实例发生了故障转移。
- 更改了数据库实例的实例类 (扩展计算)。

增强监测指标在重启数据库实例期间返回，因为只会重启数据库引擎。仍会报告操作系统的指标。

使用 CloudWatch Logs 查看操作系统指标

为数据库集群启用增强监控后，您可以使用 CloudWatch Logs 查看其指标，每个日志流表示一个被监控的数据库实例或数据库集群。日志流标识符是数据库实例或数据库集群的资源标识符 (`DbiResourceId`)。

查看增强监控日志数据

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 如有必要，请选择数据库集群所在的 AWS 区域。有关更多信息，请参阅 Amazon Web Services 一般参考 中的区域和终端节点。
3. 在导航窗格中，选择 Logs (日志)。
4. 从日志组列表中选择 RDSOSMetrics。
5. 从日志流列表中选择要查看的日志流。

Amazon Aurora 的指标参考

在此参考中，您可以找到 Amazon CloudWatch、Performance Insights 和增强监控的 Amazon Aurora 指标的说明。

主题

- Amazon Aurora 的 Amazon CloudWatch 指标
- Aurora 的 Amazon CloudWatch 维度
- Amazon RDS 控制台中 Aurora 指标的可用性
- Amazon RDS 性能详情的 Amazon CloudWatch 指标
- 性能详情计数器指标
- Performance Insights 的 SQL 统计数据
- 增强监控中的操作系统指标

Amazon Aurora 的 Amazon CloudWatch 指标

AWS/RDS 命名空间包含以下指标，这些指标适用于在 Amazon Aurora 上运行的数据库实体。某些指标适用于 Aurora MySQL 和/或 Aurora PostgreSQL。此外，某些指标特定于数据库集群、主数据库实例、副本数据库实例或所有数据库实例。

有关 Aurora Global Database 指标，请参阅 Aurora MySQL 中写入转发的 Amazon CloudWatch 指标和 Aurora PostgreSQL 中写入转发的 Amazon CloudWatch 指标。有关 Aurora 并行查询指标，请参阅监控 Aurora MySQL 的并行查询。

主题

- Amazon Aurora 的集群级指标
- Amazon Aurora 的实例级指标
- Amazon Aurora 的 Amazon CloudWatch 用量指标

Amazon Aurora 的集群级指标

下表介绍特定于 Aurora 集群的指标。

指标	描述	适用于	单位
AuroraGlobalDBDataTransferBytes	在 Aurora Global Database 中，这是从源 AWS 区域传输到辅助 AWS 区域的重做日志数据量。 **Note** 此指标仅适用于辅助 AWS 区域。	Aurora MySQL 和 Aurora PostgreSQL	字节
AuroraGlobalDBProgressLag	在 Aurora Global Database 中，用于衡量用户事务和系统事务的辅助集群落后于主集群的程度。 **Note** 此指标仅适用于辅助 AWS 区域。	Aurora MySQL 和 Aurora PostgreSQL	毫秒
AuroraGlobalDBReplicatedWriteIO	在 Aurora Global Database 中，这是从主要 AWS 区域复制到辅助 AWS 区域中的集群卷的写入输入/输出操作数。全局数据库中的辅助 AWS 区域的账单计算使用 VolumeWriteIOPs 以考虑到在集群中执行的写入。全局数据库中的主要 AWS 区域的账单计算使用 VolumeWriteIOPs 以考虑到在该集群中的写入活动，并使用 AuroraGlobalDBReplicatedWri	Aurora MySQL 和 Aurora PostgreSQL	计数

指标	描述	适用于	单位
	teIO 以考虑到全局数据库中的跨区域复制。 Note 此指标仅适用于辅助 AWS 区域。		
AuroraGlobalDBReplicationLag	对于 Aurora Global Database，这是从主要 AWS 区域中复制更新时的滞后时间。 Note 此指标仅适用于辅助 AWS 区域。	Aurora MySQL 和 Aurora PostgreSQL	毫秒
AuroraGlobalDBRPOLag	在 Aurora Global Database 中，恢复点目标 (RPO) 滞后时间。此指标用于衡量用户事务的辅助集群落后于主集群的程度。 Note 此指标仅适用于辅助 AWS 区域。	Aurora MySQL 和 Aurora PostgreSQL	毫秒

指标	描述	适用于	单位
AuroraVolumeBytesLeftTotal	集群卷的剩余可用空间。随着集群卷增长，此值会减小。如果它达到零，集群报告空间不足错误。 如果您要检测 Aurora MySQL 集群是否接近 128TiB 的大小限制，则可以比 VolumeBytesUsed 更轻松、更可靠地监控该值。AuroraVolumeBytesLeftTotal 考虑了用于内部事务管理的存储以及其他不影响存储账单的分配。	Aurora MySQL	字节
BacktrackChangeRecordsCreationRate	5 分钟内为您的数据库集群创建的回溯更改记录的数量。	Aurora MySQL	每 5 分钟计数
BacktrackChangeRecordsStored	您的数据库集群使用的回溯更改记录的数量。	Aurora MySQL	计数
BackupRetentionPeriodStorageUsed	用于支持 Aurora 数据库集群的备份保留时段内的时间点还原功能的备份存储总量。此数量包含在 TotalBackupStorageBilled 指标报告的总额中。针对每个 Aurora 集群单独计算。有关说明，请参阅了解 Amazon Aurora 备份存储使用量。	Aurora MySQL 和 Aurora PostgreSQL	字节

指标	描述	适用于	单位
ServerlessDatabaseCapacity	Aurora Serverless 数据库集群的当前容量。	Aurora MySQL 和 Aurora PostgreSQL	计数
SnapshotStorageUsed	Aurora 数据库集群的所有 Aurora 快照在其备份保留时段外消耗的备份存储总量。此数量包含在 TotalBackupStorageBilled 指标报告的总额中。针对每个 Aurora 集群单独计算。有关说明，请参阅了解 Amazon Aurora 备份存储使用量。	Aurora MySQL 和 Aurora PostgreSQL	字节
TotalBackupStorageBilled	为给定 Aurora 数据库集群计费时所针对的备份存储总量（单位为字节）。此指标包含由 BackupRetentionPeriodStorageUsed 和 SnapshotStorageUsed 指标度量的备份存储。此指标将分别针对每个 Aurora 集群进行计算。有关说明，请参阅了解 Amazon Aurora 备份存储使用量。	Aurora MySQL 和 Aurora PostgreSQL	字节

指标	描述	适用于	单位
VolumeBytesUsed	您的 Aurora 数据库集群使用的存储空间量。 此值将影响 Aurora 数据库集群的成本（有关定价信息，请参阅 Amazon RDS 定价页）。 此值不会体现一些不影响存储账单的内部存储分配。对于 Aurora MySQL，您可以测试 AuroraVolumeBytesLeftTotal 是否接近零，而不是将 VolumeBytesUsed 与 128TiB 的存储限制进行比较，从而更准确地预测空间不足问题。 对于作为克隆的集群，此指标的值取决于克隆上添加或更改的数据量。当删除原始集群或添加或删除新克隆时，该指标也可能增加或减少。有关详细信息，请参阅删除源集群卷 请注意，选择一个小的 --period 值没有意义，因为 Amazon RDS 按间隔收集这些指标，而不是连续收集它们。	Aurora MySQL 和 Aurora PostgreSQL	字节

指标	描述	适用于	单位
VolumeReadIOPs	每隔 5 分钟集群卷中计费读取 I/O 操作的数量。 计费读取操作数是在集群卷级别计算的，由 Aurora 数据库集群中的所有实例聚合而来，然后每隔 5 分钟报告一次。此值是通过采用 5 分钟以上的读取操作数指标的值计算得来的。您可通过采用计费读取操作数指标的值并除以 300 秒来确定每秒的计费读取操作数。例如，如果计费读取操作返回 13686，则每秒的计费读取操作数为 45 (13686 / 300 = 45.62)。 请求不在缓冲区缓存中而必须从存储中加载的数据库页的查询的累积计费读取操作数。您可能看到计费读取操作数出现峰值，因为查询结果是从存储中读取然后加载到缓冲区缓存中的。 请注意，选择一个小的 `--period` 值没有意义，因为 Amazon RDS 按间隔收集这些指标，而不是连续收集它们。 **Tip** 如果您的 Aurora MySQL 集群使用并	Aurora MySQL 和 Aurora PostgreSQL	每 5 分钟计数

指标	描述	适用于	单位
	行查询，您可能会看到 VolumeReadIOPS 值出现增长。并行查询不使用缓冲池。因此，尽管查询速度很快，但这种优化的处理可能会导致读取操作和相关费用的增加。		
VolumeWriteIOPs	集群卷的写入磁盘 I/O 操作数，每隔 5 分钟报告一次。有关如何计算计费写入操作的详细说明，请参阅 VolumeReadIOPs 。 请注意，选择一个小的 --period 值没有意义，因为 Amazon RDS 按间隔收集这些指标，而不是连续收集它们。	Aurora MySQL 和 Aurora PostgreSQL	每 5 分钟计数

Amazon Aurora 的实例级指标

除非另有说明，否则以下特定于实例的 Amazon CloudWatch 指标适用于所有 Aurora MySQL 和 Aurora PostgreSQL 实例。

指标	描述	适用于	单位
AbortedClients	未正确关闭的客户端连接数量。	Aurora MySQL	计数
ActiveTransactions	Aurora 数据库实例上每秒执行的当前事务的平均数目。	Aurora MySQL	每秒计数

指标	描述	适用于	单位
	默认情况下，Aurora 未启用此指标。要开始测量此值，请在特定数据库实例的数据库参数组中设置 `innodb_monitor_enable='all'` 。		
`ACUUtilization`	`ServerlessDatabaseCapacity` 指标的值除以数据库集群的最大 ACU 值。 此指标仅适用于 Aurora Serverless v2。	Aurora MySQL 和 Aurora PostgreSQL	百分比

指标	描述	适用于	单位
AuroraBinlogReplicaLag	Aurora MySQL 上运行的二进制日志副本数据库集群滞后于二进制日志复制源的时间量。滞后意味着源生成记录的速度比副本应用记录的速度更快。 此指标根据引擎版本报告不同的值： Aurora MySQL 版本 2 MySQL SHOW SLAVE STATUS 的 Seconds_Behind_Master 字段 Aurora MySQL 版本 3 SHOW REPLICA STATUS 您可以使用该指标监控充当二进制日志副本的集群中的错误和副本滞后。指标值指示以下各项： 上限值 副本滞后于复制源。 **0** 或接近 **0** 的值 复制过程处于活动状态且是最新的。 **-1** 在副本安装期间或副本处于错误状态时，可能会发生 Aurora 无法确定滞后的情况。	适用于 Aurora MySQL 的主实例	秒

指标	描述	适用于	单位
	由于二进制日志复制仅在集群的写入器实例上进行，因此我们建议使用与 WRITER 角色关联的该指标的版本。 有关管理复制的更多信息，请参阅 跨 AWS 区域复制 Amazon Aurora MySQL 数据库集群。有关排查故障的更多信息，请参阅 Amazon Aurora MySQL 复制问题。		
AuroraDMLRejectedMasterFull	由于写入器数据库实例上会话已满而遭拒绝的转发查询数。	适用于 Aurora MySQL 版本 2 的主实例	计数
AuroraDMLRejectedWriterFull	由于写入器数据库实例上会话已满而遭拒绝的转发查询数。	适用于 Aurora MySQL 版本 3 的主实例	计数
AuroraEstimatedSharedMemoryBytes	在上次配置的轮询间隔期间主动使用的共享缓冲区或缓冲池内存的估计量。	Aurora PostgreSQL	字节
AuroraMemoryHealthState	指示内存运行状况。值 0 代表 NORMAL。值 10 代表 RESERVED，这表示服务器的内存使用量已接近临界水平。 有关更多信息，请参阅 排查 Aurora MySQL 数据库内存不足问题。	Aurora MySQL 版本 3.06.1 及更高版本	计量表

指标	描述	适用于	单位
AuroraMemoryNumDeclinedSqlTotal	为避免内存不足（OOM）而拒绝的增量查询数。 有关更多信息，请参阅 排查 Aurora MySQL 数据库内存不足问题。	Aurora MySQL 版本 3.06.1 及更高版本	计数
AuroraMemoryNumKilledConnTotal	为避免 OOM 而关闭的增量连接数。 有关更多信息，请参阅 排查 Aurora MySQL 数据库内存不足问题。	Aurora MySQL 版本 3.06.1 及更高版本	计数
AuroraMemoryNumKilledQueryTotal	为避免 OOM 而结束的增量查询数。 有关更多信息，请参阅 排查 Aurora MySQL 数据库内存不足问题。	Aurora MySQL 版本 3.06.1 及更高版本	计数
AuroraMillisecondsSpentInOomRecovery	自内存运行状况低于正常状态以来的时间量。 有关更多信息，请参阅 排查 Aurora MySQL 数据库内存不足问题。	Aurora MySQL 版本 3.08.0 及更高版本	毫秒
AuroraNumOomRecoverySuccessful	内存运行状况恢复到正常状态的次数。 有关更多信息，请参阅 排查 Aurora MySQL 数据库内存不足问题。	Aurora MySQL 版本 3.08.0 及更高版本	计数

指标	描述	适用于	单位
AuroraNumOomRecoveryTriggered	内存运行状况低于正常状态的次数。 有关更多信息，请参阅 排查 Aurora MySQL 数据库内存不足问题。	Aurora MySQL 版本 3.08.0 及更高版本	计数
AuroraOptimizedReadsCacheHitRatio	优化型读取缓存提供服务的请求的百分比。 使用以下公式计算该值： orcache_blks_hit/(orcache_blks_hit + storage_blks_read) 当 AuroraOptimizedReadsCacheHitRatio 为 100% 时，表示从优化的读取缓存中读取了所有页面。当 AuroraOptimizedReadsCacheHitRatio 为 0 时，表示没有从优化的读取缓存中读取页面。	适用于 Aurora PostgreSQL 的主实例	百分比
AuroraReplicaLag	对于 Aurora 副本，从主实例中复制更新时的滞后量。	适用于 Aurora MySQL 和 Aurora PostgreSQL 的副本实例	毫秒

指标	描述	适用于	单位
AuroraReplicaLagMaximum	数据库集群中主实例和任何 Aurora 数据库实例之间的最大滞后量。 删除或重命名只读副本时，复制延迟可能会暂时达到峰值，因为旧资源会经历回收过程。为了获取该期间内的复制延迟的精确表示，建议您监控每个只读副本实例上的 AuroraReplicaLag 指标。	适用于 Aurora MySQL 和 Aurora PostgreSQL 的主实例	毫秒
AuroraReplicaLagMinimum	数据库集群中主实例和任何 Aurora 数据库实例之间的最小滞后量。	适用于 Aurora MySQL 和 Aurora PostgreSQL 的主实例	毫秒
AuroraSlowConnectionHandleCount	等待两秒钟或更长时间开始握手的连接数。 此指标仅适用于 Aurora MySQL 版本 3。	Aurora MySQL	计数
AuroraSlowHandshakeCount	花了 50 毫秒或更长时间完成握手的连接数。 此指标仅适用于 Aurora MySQL 版本 3。	Aurora MySQL	计数
BacktrackWindowActual	目标回溯时段与实际回溯时段之间的差异。	适用于 Aurora MySQL 的主实例	分钟

指标	描述	适用于	单位
BacktrackWindowAlert	对于给定的时间段，实际回溯时段小于目标回溯时段的次数。	适用于 Aurora MySQL 的主实例	计数
BlockedTransactions	每秒内数据库中被阻止的事务的平均数。	Aurora MySQL	每秒计数
BufferCacheHitRatio	缓冲区缓存提供的请求的百分比。	Aurora MySQL 和 Aurora PostgreSQL	百分比
CommitLatency	引擎和存储完成提交操作所花费的平均持续时间。	Aurora MySQL 和 Aurora PostgreSQL	毫秒
CommitThroughput	每秒平均提交操作数量。	Aurora MySQL 和 Aurora PostgreSQL	每秒计数
ConnectionAttempts	尝试连接实例的次数，无论成功与否。	Aurora MySQL	计数

指标	描述	适用于	单位
CPUCreditBalance	实例累积的 CPU 积分数，每隔 5 分钟报告一次。您可以使用此指标来确定数据库实例在给定的速率下可以突增至超出其基准性能水平的时长。 此指标仅适用于以下实例类： • Aurora MySQL：db.t2.small 、db.t2.medium 、db.t3 和 db.t4g • Aurora PostgreSQL：db.t3 和 db.t4g **Note** 建议仅将 T 数据库实例类用于开发和测试服务器，或其他非生产服务器。有关 T 实例类的更多详细信息，请参阅数据库实例类类型。 启动积分在 Amazon RDS 中的作用方式与在 Amazon EC2 中的作用方式相同。有关更多信息，请参阅《适用于 Linux 实例的 Amazon Elastic Compute Cloud 用户指南》中的启动积分。	Aurora MySQL 和 Aurora PostgreSQL	计数

指标	描述	适用于	单位
CPUCreditUsage	在指定时段内消耗的 CPU 积分数，每隔 5 分钟报告一次。此指标标识物理 CPU 在处理虚拟 CPU 分配给数据库实例的指令时所花费的时间。 此指标仅适用于以下实例类： • Aurora MySQL：db.t2.small 、db.t2.medium 、db.t3 和 db.t4g • Aurora PostgreSQL：db.t3 和 db.t4g **Note** 建议仅将 T 数据库实例类用于开发和测试服务器，或其他非生产服务器。有关 T 实例类的更多详细信息，请参阅数据库实例类类型。	Aurora MySQL 和 Aurora PostgreSQL	计数

指标	描述	适用于	单位
CPUSurplusCreditBalance	在 CPUCreditBalance 值为零时，无限实例花费的超额积分数。 CPUSurplusCreditBalance 值由获得的 CPU 积分支付。如果超额积分数超出实例可在 24 小时周期内获得的最大积分数，则超出最大积分数的已花费超额积分将产生额外费用。 CPU 信用指标仅每 5 分钟提供一次。	Aurora MySQL 和 Aurora PostgreSQL	积分（vCPU 分钟）
CPUSurplusCreditsCharged	未由获得的 CPU 积分支付并且会产生额外费用的已花费超额积分数。 在出现以下任一情况时，将对花费的超额积分收费： • 花费的超额积分超出实例可在 24 小时周期内获得的最大积分数。对于超出最大积分数的所花费超额积分，将在该小时结束时向您收费。 • 实例已停止或终止。 • 实例从 unlimited 切换为 standard。 CPU 信用指标仅每 5 分钟提供一次。	Aurora MySQL 和 Aurora PostgreSQL	积分（vCPU 分钟）

指标	描述	适用于	单位
CPUUtilization	Aurora 数据库实例占用的 CPU 百分比。	Aurora MySQL 和 Aurora PostgreSQL	百分比
DatabaseConnections	连接至数据库实例的客户端网络连接数。 数据库会话数可能高于指标值，因为指标值不包括以下内容： • 不再具有网络连接但数据库尚未清理的会话 • 数据库引擎出于自身目的创建的会话 • 由数据库引擎的并行执行功能创建的会话 • 由数据库引擎任务计划程序创建的会话 • Amazon Aurora 连接	Aurora MySQL 和 Aurora PostgreSQL	计数
DDLLatency	请求的平均持续时间，例如示例、创建、更改和删除请求。	Aurora MySQL	毫秒
DDLThroughput	每秒平均 DDL 请求数。	Aurora MySQL	每秒计数
Deadlocks	每秒内数据库中死锁的平均数。	Aurora MySQL 和 Aurora PostgreSQL	每秒计数
DeleteLatency	删除操作的平均持续时间。	Aurora MySQL	毫秒

指标	描述	适用于	单位
DeleteThroughput	每秒平均删除查询数。	Aurora MySQL	每秒计数
DiskQueueDepth	等待访问磁盘的未完成读取/写入请求的数量。	Aurora MySQL 和 Aurora PostgreSQL	计数
DMLLatency	插入、更新和删除的平均持续时间。	Aurora MySQL	毫秒
DMLThroughput	每秒平均插入、更新和删除数。	Aurora MySQL	每秒计数
EngineUptime	实例运行的时间长度。	Aurora MySQL 和 Aurora PostgreSQL	秒
FreeableMemory	随机存取内存的可用大小。对于 Aurora MySQL 和 Aurora PostgreSQL 数据库，此指标报告 /proc/meminfo 的 MemAvailable 字段的值。	Aurora MySQL 和 Aurora PostgreSQL	字节
FreeEphemeralStorage	可用临时 NVMe 存储的大小。	Aurora PostgreSQL	字节

指标	描述	适用于	单位
FreeLocalStorage	可用的本地存储空间量。 与其他数据库引擎不同，对于 Aurora 数据库实例，该指标报告每个数据库实例的可用存储量。此值取决于数据库实例类（有关定价信息，请参阅 Amazon RDS 定价页）。您可通过为实例选择较大的数据库实例类来增加对实例可用的存储空间量。 （这不适用于 Aurora Serverless v2。）	Aurora MySQL 和 Aurora PostgreSQL	字节
InsertLatency	插入操作的平均持续时间。	Aurora MySQL	毫秒
InsertThroughput	每秒平均插入操作数量。	Aurora MySQL	每秒计数
LoginFailures	每秒登录尝试失败的平均数目。	Aurora MySQL	每秒计数
MaximumUsedTransactionIDs	事务中最早的未执行 vacuum 操作的事务 ID 的期限。如果此值达到 2146483648 (2^31 - 1000000)，则强制数据库进入只读模式，避免事务 ID 重现。有关更多信息，请参阅 PostgreSQL 文档中的防止事务 ID 重现故障。	Aurora PostgreSQL	计数

指标	描述	适用于	单位
NetworkReceiveThroughput	Aurora 数据库集群中每个实例从客户端接收的网络吞吐量。此吞吐量不包括 Aurora 数据库集群中的实例与集群卷之间的网络流量。	Aurora MySQL 和 Aurora PostgreSQL	每秒字节数（控制台显示“兆字节/秒”）
NetworkThroughput	Aurora 数据库集群中每个实例从客户端接收和发送到客户端的网络吞吐量。此吞吐量不包括 Aurora 数据库集群中的实例与集群卷之间的网络流量。	Aurora MySQL 和 Aurora PostgreSQL	每秒字节数
NetworkTransmitThroughput	Aurora 数据库集群中每个实例发送到客户端的网络吞吐量。此吞吐量不包括数据库集群中的实例与集群卷之间的网络流量。	Aurora MySQL 和 Aurora PostgreSQL	每秒字节数（控制台显示“兆字节/秒”）
NumBinaryLogFiles	生成的二进制日志文件的数量。	Aurora MySQL	计数
OldestReplicationSlotLag	在接收预写日志 (WAL) 数据方面最滞后的副本的滞后大小。	Aurora PostgreSQL	字节
PurgeBoundary	允许进行 InnoDB 清除的截止事务编号。如果此指标在很长一段时间内没有进展，这很好地表明 InnoDB 清除被长时间运行的事务阻止了。要进行调查，请检查 Aurora MySQL 数据库集群上的活跃事务。	Aurora MySQL 版本 2、版本 2.11 及更高版本 Aurora MySQL 版本 3、版本 3.08 及更高版本	计数

指标	描述	适用于	单位
PurgeFinishedPoint	执行 InnoDB 清除的截止事务编号。此指标有助于您检查 InnoDB 清除的进展速度。	Aurora MySQL 版本 2、版本 2.11 及更高版本 Aurora MySQL 版本 3、版本 3.08 及更高版本	计数
Queries	每秒平均执行的查询数。	Aurora MySQL	每秒计数
RDSToAuroraPostgreSQLReplicaLag	在将更新从主 RDS PostgreSQL 实例复制到集群中的其他节点时的滞后量。	适用于 Aurora PostgreSQL 的副本	秒
ReadIOPS	每秒平均磁盘 I/O 操作数，但报告会每隔一分钟分别进行读取和写入。	Aurora MySQL 和 Aurora PostgreSQL	每秒计数
ReadIOPSEphemeralStorage	磁盘至临时 NVMe 存储的平均读取输入/输出操作数。	Aurora PostgreSQL	每秒计数
ReadLatency	每个磁盘 I/O 操作所需的平均时间。	Aurora MySQL 和 Aurora PostgreSQL	秒
ReadLatencyEphemeralStorage	每个磁盘至临时 NVMe 存储的读取输入/输出操作所需的平均时间。	Aurora PostgreSQL	毫秒

指标	描述	适用于	单位
ReadThroughput	每秒从磁盘读取的平均字节数。	Aurora MySQL 和 Aurora PostgreSQL	每秒字节数
ReadThroughputEphemeralStorage	每秒从磁盘向临时 NVMe 存储读取的平均字节数。	Aurora PostgreSQL	每秒字节数
ReplicationSlotDiskUsage	复制插槽文件使用的磁盘空间量。	Aurora PostgreSQL	字节
ResultSetCacheHitRatio	结果集缓存提供的请求的百分率。	Aurora MySQL 版本 2	百分比
RollbackSegmentHistoryListLength	记录已提交事务（带有删除标记的记录）的撤消日志。这些记录安排为由 InnoDB 清除操作进行处理。	Aurora MySQL	计数
RowLockTime	为 InnoDB 表获取行锁定所花的总时间。	Aurora MySQL	毫秒
SelectLatency	选定操作的平均时间长度。	Aurora MySQL	毫秒
SelectThroughput	每秒平均选择查询数。	Aurora MySQL	每秒计数
ServerlessDatabaseCapacity	Aurora Serverless 数据库集群的当前容量。	Aurora MySQL 和 Aurora PostgreSQL	计数

指标	描述	适用于	单位
StorageNetworkReceiveThroughput	数据库集群中每个实例从 Aurora 存储子系统接收的网络吞吐量。	Aurora MySQL 和 Aurora PostgreSQL	每秒字节数
StorageNetworkThroughput	Aurora 数据库集群中每个实例从 Aurora 存储子系统接收与发送的网络吞吐量。	Aurora MySQL 和 Aurora PostgreSQL	每秒字节数
StorageNetworkTransmitThroughput	Aurora 数据库集群中每个实例发送到 Aurora 存储子系统的网络吞吐量。	Aurora MySQL 和 Aurora PostgreSQL	每秒字节数
SumBinaryLogSize	二进制日志文件的总大小。	Aurora MySQL	字节
SwapUsage	已使用的交换空间量。此指标不适用于以下数据库实例类： • db.r3.*、db.r4.* 和 db.r7g.*（Aurora MySQL） • db.r7g.*（Aurora PostgreSQL）	Aurora MySQL 和 Aurora PostgreSQL	字节
TempStorageIOPS	连接到数据库实例的本地存储上的读写 IOPS 数量。此指标表示计数，每秒测量一次。 此指标仅适用于 Aurora Serverless v2。	Aurora MySQL 和 Aurora PostgreSQL	每秒计数

指标	描述	适用于	单位
TempStorageThroughput	与数据库实例关联的本地存储的传入或传出数据量。此指标表示字节数，每秒测量一次。 此指标仅适用于 Aurora Serverless v2。	Aurora MySQL 和 Aurora PostgreSQL	每秒字节数
TransactionAgeMaximum	最早的活动运行事务的年龄。	Aurora MySQL 版本 3、版本 3.08 及更高版本	秒
TransactionLogsDiskUsage	Aurora PostgreSQL 数据库实例上的事务日志所占的磁盘空间量。 此指标仅在 Aurora PostgreSQL 使用逻辑复制或 AWS Database Migration Service 时生成。默认情况下，Aurora PostgreSQL 使用日志记录，而不是事务日志。未使用事务日志时，此指标的值为 -1。	适用于 Aurora PostgreSQL 的主实例	字节
TruncateFinishedPoint	执行撤消截断的截止事务标识符。	Aurora MySQL 版本 2、版本 2.11 及更高版本 Aurora MySQL 版本 3、版本 3.08 及更高版本	计数

指标	描述	适用于	单位
UpdateLatency	更新操作所花的平均时间。	Aurora MySQL	毫秒
UpdateThroughput	每秒平均更新数。	Aurora MySQL	每秒计数
WriteIOPS	每秒生成的 Aurora 存储写入记录数。这或多或少是由数据库生成的日志记录数。这些不对应于 8K 页写入次数，也不对应于发送的网络数据包。	Aurora MySQL 和 Aurora PostgreSQL	每秒计数
WriteIOPSEphemeralStorage	磁盘至临时 NVMe 存储的平均写入输入/输出操作数。	Aurora PostgreSQL	每秒计数
WriteLatency	每个磁盘 I/O 操作所需的平均时间。	Aurora MySQL 和 Aurora PostgreSQL	秒
WriteLatencyEphemeralStorage	每个磁盘至临时 NVMe 存储的写入输入/输出操作所需的平均时间。	Aurora PostgreSQL	毫秒
WriteThroughput	平均每秒写入持久性存储的字节数。	Aurora MySQL 和 Aurora PostgreSQL	每秒字节数
WriteThroughputEphemeralStorage	临时 NVMe 存储每秒写入磁盘的平均字节数。	Aurora PostgreSQL	每秒字节数

Amazon Aurora 的 Amazon CloudWatch 用量指标

Amazon CloudWatch 中的 AWS/Usage 命名空间包括 Amazon RDS 服务配额的账户级用量指标。CloudWatch 自动收集所有 AWS 区域的使用量指标。

有关更多信息，请参阅《Amazon CloudWatch 用户指南》中的 CloudWatch 用量指标。有关配额的更多信息，请参阅《Service Quotas 用户指南》中 Amazon Aurora 的配额和限制和请求增加配额。

指标	描述	单位*
AuthorizationsPerDBSecurityGroup	您的 AWS 账户中每个数据库安全组的入口规则数。使用的值是账户中数据库安全组中入口规则的最大数。账户中的其他数据库安全组的入口规则数可能较少。	计数
CustomEndpointsPerDBCluster	您的 AWS 账户中每个数据库集群的自定义端点数。使用的值是账户中数据库集群内自定义端点的最大数。账户中的其他数据库集群可能具有更少的自定义端点数。	计数
DBClusterParameterGroups	您的 AWS 账户中数据库集群参数组的数量。该计数不包括默认参数组。	计数
DBClusterRoles	您的 AWS 账户中每个数据库集群的关联 AWS Identity and Access Management（IAM）角色数。使用的值是账户中数据库集群关联的 IAM 角色的最大数量。账户中的其他数据库集群可能具有更少的关联 IAM 角色数。	计数
DBClusters	您的 AWS 账户 中的 Amazon Aurora 数据库集群数量。	计数
DBInstanceRoles	您的 AWS 账户中每个数据库实例的关联 AWS Identity and Access Management（IAM）角色数。使用的值是账户中数据库实例关联的 IAM 角色的最大数量。账户中的其他数据库实例可能具有更少的关联 IAM 角色数。	计数
DBInstances	您的 AWS 账户 中的数据库实例数量。	计数
DBParameterGroups	您的 AWS 账户 中的数据库参数组数量。该计数不包括默认数据库参数组。	计数
DBSubnetGroups	您的 AWS 账户 中的数据库子网组数量。该计数不包括默认子网组。	计数
EventSubscriptions	您的 AWS 账户中的事件通知订阅数量。	计数

指标	描述	单位*
Integrations	您的 AWS 账户中与 Amazon Redshift 进行的零 ETL 集成数量。	计数
ManualClusterSnapshots	您的 AWS 账户 中手动创建的数据库集群快照数量。该计数不包括无效快照。	计数
OptionGroups	您的 AWS 账户 中的选项组数量。该计数不包括默认选项组。	计数
Proxies	您的 AWS 账户中 RDS 代理的数量。	计数
ReadReplicasPerMaster	您的账户中每个数据库实例的只读副本数。使用的值是账户中数据库实例的只读副本的最大数。账户中的其他数据库实例可能具有更少的只读副本数。	计数
ReservedDBInstances	您的 AWS 账户 中的预留数据库实例数量。该计数不包括停用或被拒绝的实例。	计数
SubnetsPerDBSubnetGroup	您的 AWS 账户中每个数据库子网组的子网数。账户中数据库子网组中子网的最大数。账户中的其他数据库子网组可能具有更少的子网数量。	计数

> **Note**
>
> Amazon RDS 不会向 CloudWatch 发布用量指标的单位。这些单位仅出现在文档中。

Aurora 的 Amazon CloudWatch 维度

您可以使用下表中的任何维度筛选 指标数据。

维度	筛选为 . . 请求的数据
DBInstanceIdentifier	特定的数据库实例。
DBClusterIdentifier	特定的 Aurora 数据库集群。

维度	筛选为 . . 请求的数据
DBClusterIdentifier, Role	特定 Aurora 数据库集群，并按实例角色 (WRITER/READER) 聚合指标。例如，您可以聚合属于某个群集的所有 READER 实例的指标。
DbClusterIdentifier, EngineName	特定 Aurora 数据库集群和引擎名称组合。例如，您可以查看集群 ams1 和引擎 aurora 的 VolumeReadIOPs 指标。
DatabaseClass	数据库类中的所有实例。例如，您可以聚合属于数据库类 db.r5.large 的所有实例的指标。
EngineName	仅标识的引擎名称。例如，您可以聚合具有引擎名称 aurora-postgresql 的所有实例的指标。
SourceRegion	仅指定区域。例如，您可以聚合 us-east-1 区域中的所有数据库实例的指标。

Amazon RDS 控制台中 Aurora 指标的可用性

并非 Amazon Aurora 提供的所有指标均在 Amazon RDS 控制台中可用。您可以使用工具（例如，AWS CLI 和 CloudWatch API）查看这些指标。此外，Amazon RDS 控制台中的某些指标仅对特定实例类显示，或具有不同的名称和度量单位。

主题

- 最后一小时视图中提供的 Aurora 指标
- 在特定情况下提供的 Aurora 指标
- 控制台中不可用的 Aurora 指标

最后一小时视图中提供的 Aurora 指标

您可以在 Amazon RDS 控制台默认的最后一小时视图中查看分类 Aurora 指标的子集。下表列出了在 Amazon RDS 控制台中为 Aurora 实例显示的类别和相关指标。

类别	指标
SQL	ActiveTransactions

类别	指标
	BlockedTransactions
	BufferCacheHitRatio
	CommitLatency
	CommitThroughput
	DatabaseConnections
	DDLLatency
	DDLThroughput
	Deadlocks
	DMLLatency
	DMLThroughput
	LoginFailures
	ResultSetCacheHitRatio
	SelectLatency
	SelectThroughput

类别	指标
系统	AuroraReplicaLag AuroraReplicaLagMaximum AuroraReplicaLagMinimum CPUCreditBalance CPUCreditUsage CPUUtilization FreeableMemory FreeLocalStorage （这不适用于 Aurora Serverless v2。） NetworkReceiveThroughput
部署	AuroraReplicaLag BufferCacheHitRatio ResultSetCacheHitRatio SelectThroughput

在特定情况下提供的 Aurora 指标

此外，某些 Aurora 指标可能仅针对特定实例类显示、仅针对数据库实例显示或具有不同的名称和度量单位：

- CPUCreditBalance 和 CPUCreditUsage 指标仅针对 Aurora MySQL db.t2 实例类和 Aurora PostgreSQL db.t3 实例类显示。
- 以下指标具有不同的名称，如下所示：

指标	显示名称
AuroraReplicaLagMaximum	最大副本滞后

指标	显示名称
AuroraReplicaLagMinimum	最小副本滞后
DDLThroughput	DDL
NetworkReceiveThroughput	网络吞吐量
VolumeBytesUsed	[计费] 卷已用字节数
VolumeReadIOPs	[计费] 卷读取 IOPS
VolumeWriteIOPs	[计费] 卷写入 IOPS

- 以下指标适用于整个 Aurora 数据库集群，只有在 Amazon RDS 控制台中查看 Aurora 数据库集群的数据库实例时，才会显示这些指标：
  - VolumeBytesUsed
  - VolumeReadIOPs
  - VolumeWriteIOPs
- 在 Amazon RDS 控制台中，以下指标的单位是兆字节，而不是字节：
  - FreeableMemory
  - FreeLocalStorage
  - NetworkReceiveThroughput
  - NetworkTransmitThroughput
- 以下指标适用于具有 Aurora 优化型读取功能的 Aurora PostgreSQL 数据库集群：
  - AuroraOptimizedReadsCacheHitRatio
  - FreeEphemeralStorage
  - ReadIOPSEphemeralStorage
  - ReadLatencyEphemeralStorage
  - ReadThroughputEphemeralStorage
  - WriteIOPSEphemeralStorage
  - WriteLatencyEphemeralStorage
  - WriteThroughputEphemeralStorage

控制台中不可用的 Aurora 指标

Amazon RDS 控制台中不提供以下 Aurora 指标：

- `AuroraBinlogReplicaLag`
- `DeleteLatency`
- `DeleteThroughput`
- `EngineUptime`
- `InsertLatency`
- `InsertThroughput`
- `NetworkThroughput`
- `Queries`
- `UpdateLatency`
- `UpdateThroughput`

Amazon RDS 性能详情的 Amazon CloudWatch 指标

性能详情会自动将一些指标发布到 Amazon CloudWatch。可以从 Performance Insights 查询相同的数据，但具有 CloudWatch 中的指标可以轻松地添加 CloudWatch 警报。还可以轻松地将指标添加到现有 CloudWatch 控制面板中。

指标	描述
DBLoad	数据库的活动会话的数量。通常，您需要活动会话的平均数量数据。在 Performance Insights 中，作为 `db.load.avg` 查询此数据。
DBLoadCPU	等待事件类型为 CPU 的活动会话的数量。在 Performance Insights 中，作为 `db.load.avg` 查询此数据，按等待事件类型 `CPU` 进行筛选。
DBLoadNonCPU	等待事件类型不为 CPU 的活动会话的数量。
DBLoadRelativeToNumVCPUs	数据库负载与数据库的虚拟 CPU 数量之比。

> **Note**
>
> 仅当数据库实例上有负载时，这些指标才会发布到 CloudWatch。

可以使用 CloudWatch 控制台、AWS CLI 或 CloudWatch API 来查看这些指标。您还可以使用特殊的指标数学函数检查其他性能详情计数器指标。有关更多信息，请参阅 在 CloudWatch 中查询其他性能详情计数器指标。

例如，可以通过运行 get-metric-statistics 命令来获取 DBLoad 指标的统计数据。

```
aws cloudwatch get-metric-statistics \
    --region us-west-2 \
    --namespace AWS/RDS \
    --metric-name DBLoad  \
    --period 60 \
    --statistics Average \
    --start-time 1532035185 \
    --end-time 1532036185 \
    --dimensions Name=DBInstanceIdentifier,Value=db-loadtest-0
```

该示例将生成与下类似的输出。

```
{
  "Datapoints": [
  {
  "Timestamp": "2021-07-19T21:30:00Z",
  "Unit": "None",
  "Average": 2.1
  },
  {
  "Timestamp": "2021-07-19T21:34:00Z",
  "Unit": "None",
  "Average": 1.7
  },
  {
  "Timestamp": "2021-07-19T21:35:00Z",
  "Unit": "None",
  "Average": 2.8
  },
  {
  "Timestamp": "2021-07-19T21:31:00Z",
```

```
"Unit": "None",
"Average": 1.5
},
{
"Timestamp": "2021-07-19T21:32:00Z",
"Unit": "None",
"Average": 1.8
},
{
"Timestamp": "2021-07-19T21:29:00Z",
"Unit": "None",
"Average": 3.0
},
{
"Timestamp": "2021-07-19T21:33:00Z",
"Unit": "None",
"Average": 2.4
}
],
"Label": "DBLoad"
}
```


有关 CloudWatch 的更多信息，请参阅 Amazon CloudWatch 用户指南中的什么是 Amazon CloudWatch？。

在 CloudWatch 中查询其他性能详情计数器指标

> **Note**
>
> 如果您启用数据库洞察的高级模式，Amazon RDS 会向 Amazon CloudWatch 发布性能详情计数器指标。使用数据库洞察，不需要使用 DB_PERF_INSIGHTS 指标数学函数。您可以使用 CloudWatch 数据库洞察控制面板来搜索、查询性能详情计数器指标并为其设置警报。

您可以对 CloudWatch 中的 RDS 性能详情指标进行查询、创建警报和绘制图表。您可以使用 CloudWatch 的 DB_PERF_INSIGHTS 指标数学函数，来访问有关数据库集群的信息。借助此功能，您可以使用未直接报告给 CloudWatch 的性能详情指标来创建新的时间序列。

您可以在 CloudWatch 控制台的选择指标屏幕中单击添加数学下拉菜单，来使用新的指标数学函数。您可以使用它来创建有关性能详情指标，或者 CloudWatch 和性能详情指标组合的警报和图表，包括亚分

钟指标的高分辨率警报。您也可以通过在 `get-metric-data` 请求中包含公制数学表达式来以编程方式使用该函数。有关更多信息，请参阅指标数学语法和函数，以及针对 AWS 数据库中的性能详情计数器指标创建警报。

性能详情计数器指标

计数器指标是性能详情控制面板中的操作系统和数据库性能指标。为帮助确定和分析性能问题，您可将计数器指标与数据库负载相关联。您必须向指标追加一个统计函数来获取指标值。例如，`os.memory.active` 指标支持的函数为 `.avg`、`.min`、`.max`、`.sum` 和 `.sample_count`。

每分钟收集一次计数器指标。操作系统指标收集取决于增强监控是开启还是关闭。如果关闭增强监控，则每分钟收集一次操作系统指标。如果开启增强监控，则会收集所选时间段内的操作系统指标。有关开启或关闭增强监控的更多信息，请参阅打开和关闭增强监控。

主题

- 性能详情操作系统计数器
- Aurora MySQL 的性能详情计数器
- 适用于 Aurora PostgreSQL 的性能详情计数器

性能详情操作系统计数器

以下操作系统计数器（前缀为 `os`）可用于 Aurora PostgreSQL 和 Aurora MySQL 的性能详情。

您可以使用 `ListAvailableResourceMetrics` API 获取数据库实例的可用计数器指标列表。有关更多信息，请参阅《Amazon RDS 性能详情 API 参考指南》中的 ListAvailableResourceMetrics。

计数器	类型	单位	指标	描述
Active	内存	千字节	os.memory .active	已分配的内存量 (以 KB 为单位)。
Buffers	内存	千字节	os.memory .buffers	在写入存储设备前用于缓存 I/O 请求的内存量 (以 KB 为单位)。
已缓存	内存	千字节	os.memory .cached	用于缓存基于文件系统的 I/O 的

计数器	类型	单位	指标	描述
				内存量，以 KB 为单位。
数据库缓存	内存	字节	os.memory.db.cache	包括 tmpfs (shmem) 在内的数据库进程用于页面缓存的内存量，以字节为单位。
数据库驻留集大小	内存	字节	os.memory.db.residentSetSize	不包括 tmpfs (shmem) 的数据库进程用于匿名缓存和交换缓存的内存量，以字节为单位。
数据库交换	内存	字节	os.memory.db.swap	数据库进程用于交换的内存量，以字节为单位。
脏	内存	千字节	os.memory.dirty	RAM 中已修改但未写入存储中的相关数据块的内存页面大小 (以 KB 为单位)。
Free	内存	千字节	os.memory.free	未分配的内存量 (以 KB 为单位)。
Huge Pages Free	内存	页面	os.memory.hugePagesFree	可用大页数。大页是 Linux 内核的一项功能。
Huge Pages Rsvd	内存	页面	os.memory.hugePagesRsvd	已提交大页数。

计数器	类型	单位	指标	描述
Huge Pages Size	内存	千字节	os.memory .hugePagesSize	每个大页单位的大小 (以 KB 为单位)。
Huge Pages Surp	内存	页面	os.memory .hugePagesSurp	剩余可用大页总数。
Huge Pages Total	内存	页面	os.memory .hugePagesTotal	大页总数。
Inactive	内存	千字节	os.memory .inactive	最不常用内存页面大小 (以 KB 为单位)。
已映射	内存	千字节	os.memory .mapped	在进程地址空间中有内存映射的文件系统内容的总大小 (以 KB 为单位)。
内存不足终止计数	内存	终止次数	os.memory .outOfMem oryKillCount	在上次收集间隔内发生的 OOM 终止次数。
Page Tables	内存	千字节	os.memory .pageTables	页表使用的内存量 (以 KB 为单位)。
碎片	内存	千字节	os.memory.slab	可重用内核数据结构大小 (以 KB 为单位)。
Total	内存	千字节	os.memory.total	内存总量 (以 KB 为单位)。

计数器	类型	单位	指标	描述
回写	内存	千字节	os.memory .writeback	RAM 中仍在写入备份存储的脏页大小 (以 KB 为单位)。
来宾	Cpu 利用率	百分比	os.cpuUtilization. guest	来宾程序使用的 CPU 百分比。
Idle	Cpu 利用率	百分比	os.cpuUtilization. idle	CPU 空闲百分比。
Irq	Cpu 利用率	百分比	os.cpuUtilization. irq	软件中断使用的 CPU 百分比。
良好	Cpu 利用率	百分比	os.cpuUtilization. nice	以最低优先级运行的程序使用的 CPU 百分比。
被盗用	Cpu 利用率	百分比	os.cpuUtilization. steal	其他虚拟机使用的 CPU 百分比。
System	Cpu 利用率	百分比	os.cpuUtilization. system	内核使用的 CPU 百分比。
Total	Cpu 利用率	百分比	os.cpuUtilization. total	使用中的 CPU 百分比总计。此值包含 nice（良好）值。
User	Cpu 利用率	百分比	os.cpuUtilization. user	用户程序使用的 CPU 百分比。
Wait	Cpu 利用率	百分比	os.cpuUtilization. wait	等待 I/O 访问时的未使用 CPU 百分比。

计数器	类型	单位	指标	描述
Aurora 存储 Aurora 存储字节 Rx	磁盘 IO	每秒字节数	os.diskIO .auroraSt orage.aur oraStorag eBytesRx	每秒从 Aurora 存储接收的字节数。
Aurora 存储 Aurora 存储字节 Tx	磁盘 IO	每秒字节数	os.diskIO .auroraSt orage.aur oraStorag eBytesTx	每秒上传到 Aurora 存储的字节数。
Aurora 存储磁盘队列深度	磁盘 IO	请求	os.diskIO .auroraSt orage.dis kQueueDepth	Aurora 存储磁盘队列的长度。
Aurora 存储读取 IO PS	磁盘 IO	每秒请求数	os.diskIO .auroraSt orage.rea dIOsPS	每秒的读取操作数。
Aurora 存储读取延迟	磁盘 IO	毫秒	os.diskIO .auroraSt orage.rea dLatency	从 Aurora 存储中读取 I/O 请求的平均延迟（以毫秒为单位）。
Aurora 存储读取吞吐量	磁盘 IO	每秒字节数	os.diskIO .auroraSt orage.rea dThroughput	向数据库集群发出的请求要使用的网络吞吐量（以字节/秒为单位）。

计数器	类型	单位	指标	描述
Aurora 存储写入 IO PS	磁盘 IO	每秒请求数	os.diskIO .auroraSt orage.wri teIOsPS	每秒的写入操作数。
Aurora 存储写入延迟	磁盘 IO	毫秒	os.diskio .auroraSt orage.wri teLatency	向 Aurora 存储中写入 I/O 请求的平均延迟（以毫秒为单位）。
Aurora 存储写入吞吐量	磁盘 IO	每秒字节数	os.diskIO .auroraSt orage.wri teThroughput	来自数据库集群的响应所使用的网络吞吐量（以字节/秒为单位）。
Rdstemp 平均队列长度	磁盘 IO	请求	os.diskIO .rdstemp. avgQueueLen	在 I/O 设备队列中等待的请求数。
Redstemp 平均请求大小	磁盘 IO	请求	os.diskIO .rdstemp. avgReqSz	在 I/O 设备队列中等待的请求数。
Rdstemp 等待	磁盘 IO	毫秒	os.diskIO .rdstemp.await	响应请求所需的毫秒数，包括排队时间和服务时间。
Rdstemp 读取 IO PS	磁盘 IO	请求	os.diskIO .rdstemp. readIOsPS	每秒的读取操作数。
Rdstemp 读取 KB	磁盘 IO	千字节	os.diskIO .rdstemp.readKb	读取的总 KB 数。

计数器	类型	单位	指标	描述
Rdstemp Read KB PS	磁盘 IO	千字节/秒	os.diskIO .rdstemp. readKbPS	每秒读取的 KB 数。
Rdstemp Rrqm PS	磁盘 IO	每秒请求数	os.diskIO .rdstemp.rrqmPS	每秒排队的合并读取请求数。
Rdstemp TPS	磁盘 IO	每秒事务数	os.diskIO .rdstemp.tps	每秒的 I/O 事务数。
Rdstemp Util	磁盘 IO	百分比	os.diskIO .rdstemp.util	发出请求所经历的 CPU 时间的百分比。
Rdstemp Write IOs PS	磁盘 IO	每秒请求数	os.diskIO .rdstemp. writeIOsPS	每秒的写入操作数。
Rdstemp Write KB	磁盘 IO	千字节	os.diskIO .rdstemp.writeKb	写入的总 KB 数。
Rdstemp Write KB PS	磁盘 IO	千字节/秒	os.diskIO .rdstemp. writeKbPS	每秒写入的 KB 数。
Rdstemp Wrqm PS	磁盘 IO	每秒请求数	os.diskIO .rdstemp. wrqmPS	每秒排队的合并写入请求数。
阻止	任务	任务	os.tasks.blocked	已阻止的任务的数量。
Running	任务	任务	os.tasks.running	正在运行的任务的数量。
Sleeping	任务	任务	os.tasks.sleeping	正在睡眠的任务的数量。

计数器	类型	单位	指标	描述
Stopped（已停止）	任务	任务	os.tasks.stopped	已停止的任务的数量。
Total	任务	任务	os.tasks.total	任务总数。
Zombie	任务	任务	os.tasks.zombie	有活动父任务的不活动子任务的数量。
一	加载平均分钟数	进程	os.loadAverageMinute.one	过去 1 分钟内请求 CPU 时间的进程数。
十五	加载平均分钟数	进程	os.loadAverageMinute.fifteen	过去 15 分钟内请求 CPU 时间的进程数。
五	加载平均分钟数	进程	os.loadAverageMinute.five	过去 5 分钟内请求 CPU 时间的进程数。
已缓存	Swap (交换)	千字节	os.swap.cached	用作缓存内存的交换内存量 (以 KB 为单位)。
免费	Swap (交换)	千字节	os.swap.free	空闲交换内存量（以 KB 为单位）。
In	Swap (交换)	千字节	os.swap.in	从磁盘换入的内存量（以 KB 为单位）。
Out	Swap (交换)	千字节	os.swap.out	换出到磁盘的内存量（以 KB 为单位）。

计数器	类型	单位	指标	描述
Total	Swap (交换)	千字节	os.swap.total	可用的总交换内存量（以 KB 为单位）。
最大文件数	文件系统	文件	os.fileSys.maxFiles	可为文件系统创建的文件的最大数量。
已用文件	文件系统	文件	os.fileSys.usedFiles	文件系统中的文件数。
已用文件百分比	文件系统	文件	os.fileSys.usedFilePercent	使用中的可用文件百分比。
已用百分比	文件系统	百分比	os.fileSys.usedPercent	使用中的文件系统磁盘空间百分比。
已用	文件系统	千字节	os.fileSys.used	文件系统中的文件所用的磁盘空间量 (以 KB 为单位)。
Total	文件系统	千字节	os.fileSys.total	文件系统的可用磁盘空间总量 (以 KB 为单位)。
Rx	网络	每秒字节数	os.network.rx	每秒接收的字节数。
Tx	网络	每秒字节数	os.network.tx	每秒上传的字节数。
Acu 利用率	常规	百分比	os.general.acuUtilization	当前容量占最大配置容量的百分比。

计数器	类型	单位	指标	描述
最大配置的 Acu	常规	ACU	os.general.maxConfiguredAcu	用户配置的最大容量，以 Aurora 容量单位（ACU）为单位。
最小配置的 Acu	常规	ACU	os.general.minConfiguredAcu	用户配置的最小容量，以 ACU 为单位。
VCPU 数量	常规	vCPU	os.general.numVCPUs	数据库实例的虚拟 CPU（vCPU）数量。
无服务器数据库容量	常规	ACU	os.general.serverlessDatabaseCapacity	实例以 ACU 表示的当前容量。

Aurora MySQL 的性能详情计数器

以下数据库计数器可用于 Aurora MySQL 的性能详情。

主题

- Aurora MySQL 的本机计数器
- Aurora MySQL 的非本机计数器

Aurora MySQL 的本机计数器

本机指标由数据库引擎定义，而不是由 Amazon Aurora 定义。您可以在 MySQL 文档中的服务器状态变量中找到这些原生指标的定义。

计数器	类型	单位	指标
Com_analyze	SQL	每秒查询数	db.SQL.Com_analyze
Com_optimize	SQL	每秒查询数	db.SQL.Com_optimize
Com_select	SQL	每秒查询数	db.SQL.Com_select
Innodb_rows_deleted	SQL	每秒行数	db.SQL.Innodb_rows_deleted
Innodb_rows_inserted	SQL	每秒行数	db.SQL.Innodb_rows_inserted
Innodb_rows_read	SQL	每秒行数	db.SQL.Innodb_rows_read
Innodb_rows_updated	SQL	每秒行数	db.SQL.Innodb_rows_updated
查询	SQL	每秒查询数	db.SQL.Queries
问题	SQL	每秒查询数	db.SQL.Questions
Select_full_join	SQL	每秒查询数	db.SQL.Select_full_join
Select_full_range_join	SQL	每秒查询数	db.SQL.Select_full_range_join
Select_range	SQL	每秒查询数	db.SQL.Select_range
Select_range_check	SQL	每秒查询数	db.SQL.Select_range_check
Select_scan	SQL	每秒查询数	db.SQL.Select_scan

计数器	类型	单位	指标
Slow_queries	SQL	每秒查询数	db.SQL.Slow_queries
Sort_merge_passes	SQL	每秒查询数	db.SQL.Sort_merge_passes
Sort_range	SQL	每秒查询数	db.SQL.Sort_range
Sort_rows	SQL	每秒查询数	db.SQL.Sort_rows
Sort_scan	SQL	每秒查询数	db.SQL.Sort_scan
Total_query_time	SQL	毫秒	db.SQL.Total_query_time
Table_locks_immediate	锁	每秒请求数	db.Locks.Table_locks_immediate
Table_locks_waited	锁	每秒请求数	db.Locks.Table_locks_waited
Innodb_row_lock_time	锁	毫秒数（平均值）	db.Locks.Innodb_row_lock_time
Aborted_clients	用户	连接	db.Users.Aborted_clients
Aborted_connects	用户	连接	db.Users.Aborted_connects
连接	Users	连接	db.Users.Connections
External_threads_connected	Users	连接	db.Users.External_threads_connected
max_connections	Users	连接	db.Users.max_connections
Threads_connected	Users	连接	db.Users.Threads_connected

计数器	类型	单位	指标
Threads_created	用户	连接	db.Users.Threads_created
Threads_running	用户	连接	db.Users.Threads_running
Created_tmp_disk_tables	临时文件	每秒表数	db.Temp.Created_tmp_disk_tables
Created_tmp_tables	临时文件	每秒表数	db.Temp.Created_tmp_tables
Innodb_buffer_pool_pages_data	缓存	页面	db.Cache.Innodb_buffer_pool_pages_data
Innodb_buffer_pool_pages_total	缓存	页面	db.Cache.Innodb_buffer_pool_pages_total
Innodb_buffer_pool_read_requests	缓存	每秒页数	db.Cache.Innodb_buffer_pool_read_requests
Innodb_buffer_pool_reads	缓存	每秒页数	db.Cache.Innodb_buffer_pool_reads
Opened_tables	缓存	表	db.Cache.Opened_tables
Opened_table_definitions	缓存	表	db.Cache.Opened_table_definitions
Qcache_hits	缓存	查询	db.Cache.Qcache_hits

Aurora MySQL 的非本机计数器

非本机计数器指标是 Amazon RDS 定义的计数器。非本机指标可以是您使用特定查询获取的指标。非本机指标还可以是派生指标，使用两个或更多个本机计数器来计算比率、命中率或延迟。

计数器	类型	单位	指标	描述	定义
active_transactions	事务	db.Transactions.a	总活动事务数。	`SELECT COUNT(1) AS active_transactions`	

计数器	类型	单位	指标	描述	定义
		tive_transaction		FROM INFORMATION_SCHEMA.INNODB_TRX	
innodb_buffer_pool_hit_rate	缓存	db.CacheinnoDBffer_poo_hit_rat	InnoDB 可满足的缓冲池中的读取百分比。	100 * innodb_buffer_pool_read_requests / (innodb_buffer_pool_read_requests + innodb_buffer_pool_reads)	
innodb_buffer_pool_hits	缓存	每秒页数	db.Cache.innoDB_buffer_pool_hits	InnoDB 可满足的缓冲池中的读取数。	innodb_buffer_pool_read_requests - innodb_buffer_pool_reads

计数器	类型	单位	指标	描述	定义
innodb_buffer_pool_usage	缓存	百分比	db.Cache.innoDB_buffer_pool_usage	包含数据（页面）的 InnoDB 缓冲池的百分比。 **Note** 在使用压缩表时，此值可改变。有关更多信息，请参阅 MySQL 文档的服务器状态变量中有关 Innodb_buffer_pool_pages_data 和 Innodb_buffer_pool_pages_total 的信息。	`Innodb_buffer_pool_pages_data / Innodb_buffer_pool_pages_total * 100.0`
innodb_deadlocks	锁	db.Lock	innodb_deadlocks	死锁总数。	`SELECT COUNT AS innodb_deadlocks FROM INFORMATION_SCHEMA.INNODB_METRICS WHERE NAME='lock_deadlocks'`

计数器	类型	单位	指标	描述	定义
innodb_lock_timeouts	锁	db.Lock.innodb_lock_timeouts	超时的死锁总数。	SELECT COUNT AS innodb_lock_timeouts FROM INFORMATION_SCHEMA.INNODB_METRICS WHERE NAME='lock_timeouts'	
innodb_row_lock_waits	锁	db.Lock.innodb_row_lock_waits	导致等待的行锁总数。	SELECT COUNT AS innodb_row_lock_waits FROM INFORMATION_SCHEMA.INNODB_METRICS WHERE NAME='lock_row_lock_waits'	
innodb_rows_changed	SQL	db.SQL.innodb_rows_changed	总 InnoDB 行操作数。	db.SQL.Innodb_rows_inserted + db.SQL.Innodb_rows_deleted + db.SQL.Innodb_rows_updated	

计数器	类型	单位	指标	描述	定义
query_cache_hit_rate	缓存	百分比	db.Cache.query_cache_hit_rate	MySQL 结果集缓存（查询缓存）的命中率。	`Qcache_hits / (QCache_hits + Com_select) * 100`
temp_disk_tables_percent	临时文件	db.Temp_disk_tables_percent	运行语句时，服务器在磁盘上创建的临时表的百分比。	`(db.Temp.Created_tmp_disk_tables / db.Temp.Created_tmp_tables) * 100`	
trx_rseg_history_len	事务	无	db.Transactions.trx_rseg_history_len	InnoDB 事务系统维护的已提交事务的撤消日志页面列表，用于实现多版本并发控制。有关撤消日志记录详细信息的更多信息，请参阅 MySQL 文档中的 https://dev.mysql.com/doc/refman/8.0/en/innodb-multi-versioning.html。	`SELECT COUNT AS trx_rseg_history_len FROM INFORMATION_SCHEMA.INNODB_METRICS WHERE NAME='trx_rseg_history_len'`

适用于 Aurora PostgreSQL 的性能详情计数器

以下数据库计数器可用于 Aurora PostgreSQL 的性能详情。

主题

- Aurora PostgreSQL 的本机计数器
- Aurora PostgreSQL 的非本机计数器

Aurora PostgreSQL 的本机计数器

本机指标由数据库引擎定义，而不是由 Amazon Aurora 定义。您可以在 PostgreSQL 文档中的查看统计数据中找到这些本机指标的定义。

计数器	类型	单位	指标
tup_deleted	SQL	每秒元组数	db.SQL.tup_deleted
tup_fetched	SQL	每秒元组数	db.SQL.tup_fetched
tup_inserted	SQL	每秒元组数	db.SQL.tup_inserted
tup_returned	SQL	每秒元组数	db.SQL.tup_returned
tup_updated	SQL	每秒元组数	db.SQL.tup_updated
blks_hit	缓存	每秒块数	db.Cache.blks_hit
buffers_alloc	缓存	每秒块数	db.Cache.buffers_alloc
buffers_checkpoint	检查点	每秒块数	db.Checkpoint.buffers_checkpoint
checkpoints_req	检查点	每分钟检查点数	db.Checkpoint.checkpoints_req
checkpoint_sync_time	检查点	每个检查点的毫秒数	db.Checkpoint.checkpoint_sync_time
checkpoints_timed	检查点	每分钟检查点数	db.Checkpoint.checkpoints_timed
checkpoint_write_time	检查点	每个检查点的毫秒数	db.Checkpoint.checkpoint_write_time
maxwritten_clean	检查点	每分钟 Bgwriter 清理停止数	db.Checkpoint.maxwritten_clean

计数器	类型	单位	指标
deadlocks	并发	每分钟死锁数	db.Concurrency.deadlocks
blk_read_time	I/O	毫秒	db.IO.blk_read_time
blks_read	I/O	每秒块数	db.IO.blks_read
buffers_backend	I/O	每秒块数	db.IO.buffers_backend
buffers_backend_fsync	I/O	每秒块数	db.IO.buffers_backend_fsync
buffers_clean	I/O	每秒块数	db.IO.buffers_clean
temp_bytes	临时文件	每秒字节数	db.Temp.temp_bytes
temp_files	临时文件	每分钟文件数	db.Temp.temp_files
xact_commit	事务	每秒提交数	db.Transactions.xact_commit
xact_rollback	事务	每秒回滚数	db.Transactions.xact_rollback
numbackends	User	连接	db.User.numbackends
archived_count	WAL	每分钟文件数	db.WAL.archived_count

Aurora PostgreSQL 的非本机计数器

非本机计数器指标是 Amazon Aurora 定义的计数器。非本机指标可以是您使用特定查询获取的指标。非本机指标还可以是派生指标，使用两个或更多个本机计数器来计算比率、命中率或延迟。

计数器	类型	单位	指标	描述	定义
checkpoint_sync_latency	检查点	毫秒	db.Checkpoint.checkpoint_sync_latency	在文件同步到磁盘时，检查点处理部分已花费的总时间量。	checkpoint_sync_time / (checkpoints_timed + checkpoints_req)
checkpoint_write_latency	检查点	毫秒	db.Checkpoint.checkpoint_write_latency	在文件写入磁盘时，检查点处理部分已花费的总时间量。	checkpoint_write_time / (checkpoints_timed + checkpoints_req)
local_blks_read	I/O	数据块	db.IO.local_blks_read	读取的本地块总数。	不适用
local_blk_read_time	I/O	毫秒	db.IO.local_blk_read_time	如果已启用 `track_io_timing` ，它将跟踪读取本地数据文件块所花费的总时间（以毫秒为单位），否则该值为零。有关更多信息，请参阅 track_io_timing。	不适用
num_blocked_sessions	锁	db.Locks.num_blocked_sessions	被阻止的会话数。	–	

计数器	类型	单位	指标	描述	定义
orcache_blks_hit	I/O	查询	db.IO.orcache_blks_hit	优化读取缓存中的共享块命中总数。	不适用
orcache_blk_read_time	I/O	毫秒	db.IO.orcache_blk_read_time	如果已启用 `track_io_timing` ，它将跟踪从优化型读取缓存中读取数据文件块所花费的总时间（以毫秒为单位），否则该值为零。有关更多信息，请参阅 track_io_timing。	不适用
read_latency	I/O	毫秒	db.IO.read_latency	此实例中的后端读取数据文件块所花费的时间。	`blk_read_time / blks_read`
storage_blks_read	I/O	数据块	db.IO.storage_blks_read	从 aurora 存储中读取的共享块总数。	不适用
storage_blk_read_time	I/O	毫秒	db.IO.storage_blk_read_time	如果已启用 `track_io_timing` ，它将跟踪从 Aurora 存储中读取数据文件块所花费的总时间（以毫秒为单位），否则该值为零。有关更多信息，请参阅 track_io_timing。	不适用

计数器	类型	单位	指标	描述	定义
num_blocked_sessions	锁	db.Locks.num_blocked_sessions	被阻止的会话数。	–	
active_count	状态	会话	db.state.active_count	处于 active 状态的会话数量。	不适用
idle_count	状态	会话	db.state.idle_count	处于 idle 状态的会话数量。	不适用
idle_in_transaction_aborted_count	状态	会话	db.state.idle_in_transaction_aborted_count	处于 idle in transaction (aborted) 状态的会话数量。	不适用
idle_in_transaction_count	状态	会话	db.state.idle_in_transaction_count	处于 idle in transaction 状态的会话数量。	不适用
idle_in_transaction_max_time	状态	秒	db.state.idle_in_transaction_max_time	idle in transaction 状态下运行时间最长的事务的持续时间（以秒为单位）。	不适用
logical_reads	SQL	数据块	db.SQL.logical_reads	命中和读取的总块数。	blks_hit + blks_read
queries_started	SQL	查询	db.SQL.queries	启动的查询数。	不适用

计数器	类型	单位	指标	描述	定义
queries_finished	SQL	查询	db.SQL.queries	完成的查询数。	不适用
total_query_time	SQL	毫秒	db.SQL.total_query_time	执行语句花费的总时间（以毫秒为单位）。	不适用
active_transactions	事务	事务	db.Transactions.active_transactions	活动事务的数量。	不适用
blocked_transactions	事务	事务	db.Transactions.blocked_transactions	阻止的事务的数量。	不适用
commit_latency	事务	微秒	db.Transactions.commit_latency	提交操作的平均持续时间。	`db.Transactions.duration_commits / db.Transactions.xact_commit`
duration_commits	事务	毫秒	db.Transactions.duration_commits	最后一分钟花费的事务处理总时间（以毫秒为单位）。	不适用
max_used_xact_ids	事务	事务	db.Transactions.max_used_xact_ids	尚未清空的事务数量。	不适用

计数器	类型	单位	指标	描述	定义
oldest_inactive_logical_replication_slot_xid_age	事务	Length	db.Transactions.oldest_inactive_logical_replication_slot_xid_age	非活动逻辑复制插槽中最早事务的期限。	不适用
oldest_active_logical_replication_slot_xid_age	事务	Length	db.Transactions.oldest_active_logical_replication_slot_xid_age	活动逻辑复制插槽中最早事务的期限。	不适用
oldest_reader_feedback_xid_age	事务	Length	db.Transactions.oldest_reader_feedback_xid_age	Aurora 读取器实例或 Aurora Global Database 读取器实例上长期运行的事务中最早事务的期限。	不适用
oldest_prepared_transaction_xid_age	事务	Length	db.Transactions.oldest_prepared_transaction_xid_age	最早的准备好的事务的期限。	不适用

计数器	类型	单位	指标	描述	定义
oldest_running_transaction_xid_age	事务	Length	db.Transactions.oldest_running_transaction_xid_age	最早的运行事务的期限。	不适用
max_connections	Users	Users	db.User.max_connections	`max_connections` 参数中配置的数据库允许的最大连接数。	不适用
total_auth_attempts	Users	Users	db.User.total_auth_attempts	尝试连接到此实例的次数。	不适用
archive_failed_count	WAL	每分钟文件数	db.WAL.archive_failed_count	尝试存档 WAL 文件的失败次数（以每分钟文件数为单位）。	不适用

Performance Insights 的 SQL 统计数据

SQL 统计数据是由 Performance Insights 收集的关于 SQL 查询的性能相关指标。Performance Insights 收集查询运行的每一秒的统计数据，以及每次 SQL 调用的统计数据。SQL 统计数据是所选时间范围内的平均值。

SQL 摘要 是具有给定模式但不一定具有相同文本值的所有查询的组合。摘要用问号替换文本值。例如，`SELECT * FROM emp WHERE lname= ?`。此摘要可能由以下子查询组成：

```
SELECT * FROM emp WHERE lname = 'Sanchez'
SELECT * FROM emp WHERE lname = 'Olagappan'
SELECT * FROM emp WHERE lname = 'Wu'
```

所有引擎都支持摘要查询的 SQL 统计数据。

有关此功能的区域、数据库引擎和实例类支持信息，请参阅[支持性能详情功能的 Amazon Aurora 数据库引擎、区域和实例类]

主题

- Aurora MySQL 的 SQL 统计数据
- Aurora PostgreSQL 的 SQL 统计数据

Aurora MySQL 的 SQL 统计数据

Aurora MySQL 仅在摘要级别收集 SQL 统计信息。语句级别没有显示任何统计数据。

主题

- Aurora MySQL 的摘要统计数据
- Aurora MySQL 的每秒统计数据
- Aurora MySQL 的每次调用统计数据
- Aurora MySQL 的主要统计数据

Aurora MySQL 的摘要统计数据

性能详情从 `events_statements_summary_by_digest` 表中收集 SQL 摘要统计数据。`events_statements_summary_by_digest` 表由您的数据库管理。

此摘要表不提供移出策略。当表已满时，AWS Management Console 将显示以下消息：

```
Performance Insights is unable to collect SQL Digest statistics on new queries because
 the table events_statements_summary_by_digest is full.
Please truncate events_statements_summary_by_digest table to clear the issue. Check the
 User Guide for more details.
```

在这种情况下， Aurora MySQL 不会跟踪 SQL 查询。为了解决此问题，性能详情会在满足以下两个条件时自动截取摘要表：

- 表已满。
- 性能详情会自动管理 Performance Schema。

  对于自动管理，`performance_schema` 参数必须设置为 `0`，Source (源) 不应设置为 `user`。如果性能详情没有自动管理 Performance Schema，请参阅 Aurora MySQL 上性能详情的性能架构概述。

在 AWS CLI 中，检查参数值的来源，方法是运行 describe-db-parameters 命令。

Aurora MySQL 的每秒统计数据

以下 SQL 统计数据适用于 Aurora MySQL 数据库集群。

指标	Unit
db.sql_tokenized.stats.count_star_per_sec	每秒调用数
db.sql_tokenized.stats.sum_timer_wait_per_sec	每秒平均延迟（以毫秒为单位）
db.sql_tokenized.stats.sum_select_full_join_per_sec	选择每秒完全联接数
db.sql_tokenized.stats.sum_select_range_check_per_sec	选择每秒范围检查数
db.sql_tokenized.stats.sum_select_scan_per_sec	选择每秒扫描数
db.sql_tokenized.stats.sum_sort_merge_passes_per_sec	对每秒合并传递数进行排序
db.sql_tokenized.stats.sum_sort_scan_per_sec	对每秒对扫描数进行排序
db.sql_tokenized.stats.sum_sort_range_per_sec	对每秒范围数进行排序
db.sql_tokenized.stats.sum_sort_rows_per_sec	对每秒行数进行排序
db.sql_tokenized.stats.sum_rows_affected_per_sec	每秒受影响的行数
db.sql_tokenized.stats.sum_rows_examined_per_sec	每秒检查的行数
db.sql_tokenized.stats.sum_rows_sent_per_sec	每秒发送的行数
db.sql_tokenized.stats.sum_created_tmp_disk_tables_per_sec	每秒创建的临时磁盘表数

指标	Unit
db.sql_tokenized.stats.sum_created_tmp_tables_per_sec	每秒创建的临时表数
db.sql_tokenized.stats.sum_lock_time_per_sec	每秒锁定时间（以毫秒为单位）

Aurora MySQL 的每次调用统计数据

以下指标提供了 SQL 语句的每次调用统计数据。

指标	单位
db.sql_tokenized.stats.sum_timer_wait_per_call	每个调用的平均延迟（以毫秒为单位）
db.sql_tokenized.stats.sum_select_full_join_per_call	选择每次调用完全联接数
db.sql_tokenized.stats.sum_select_range_check_per_call	选择每次调用范围检查数
db.sql_tokenized.stats.sum_select_scan_per_call	选择每次调用扫描数
db.sql_tokenized.stats.sum_sort_merge_passes_per_call	对每次调用合并传递数进行排序
db.sql_tokenized.stats.sum_sort_scan_per_call	对每次调用扫描数进行排序
db.sql_tokenized.stats.sum_sort_range_per_call	对每次调用范围数进行排序
db.sql_tokenized.stats.sum_sort_rows_per_call	对每次调用行数进行排序
db.sql_tokenized.stats.sum_rows_affected_per_call	每次调用受影响的行数
db.sql_tokenized.stats.sum_rows_examined_per_call	每次调用检查的行数

指标	单位
db.sql_tokenized.stats.sum_rows_sent_per_call	每次调用发送的行数
db.sql_tokenized.stats.sum_created_tmp_disk_tables_per_call	每次调用创建的临时磁盘表数
db.sql_tokenized.stats.sum_created_tmp_tables_per_call	每次调用创建的临时表数
db.sql_tokenized.stats.sum_lock_time_per_call	每个调用的锁定时间（以毫秒为单位）

Aurora MySQL 的主要统计数据

以下 SQL 统计数据适用于 Aurora MySQL 数据库集群。

指标	单位
db.sql_tokenized.stats.count_star	Calls
db.sql_tokenized.stats.sum_timer_wait	等待时间（以毫秒为单位）
db.sql_tokenized.stats.sum_select_full_join	选择完全联接
db.sql_tokenized.stats.sum_select_range_check	选择范围检查
db.sql_tokenized.stats.sum_select_scan	选择扫描
db.sql_tokenized.stats.sum_sort_merge_passes	对合并传递进行排序
db.sql_tokenized.stats.sum_sort_scan	对扫描进行排序
db.sql_tokenized.stats.sum_sort_range	对范围进行排序
db.sql_tokenized.stats.sum_sort_rows	排序行
db.sql_tokenized.stats.sum_rows_affected	受影响的行数
db.sql_tokenized.stats.sum_rows_examined	检查的行数

指标	单位
db.sql_tokenized.stats.sum_rows_sent	发送的行数
db.sql_tokenized.stats.sum_created_tmp_disk_tables	创建的临时磁盘表数
db.sql_tokenized.stats.sum_created_tmp_tables	创建的临时表数
db.sql_tokenized.stats.sum_lock_time	锁定时间（以毫秒为单位）

Aurora PostgreSQL 的 SQL 统计数据

对于每个 SQL 调用和查询运行的每一秒，性能详情都会收集 SQL 统计数据。所有 Aurora 引擎仅在摘要级别收集统计数据。

接下来，您可以了解有关 Aurora PostgreSQL 的摘要级别统计数据的信息。

主题

- Aurora PostgreSQL 的摘要统计数据
- Aurora PostgreSQL 的每秒摘要统计数据
- Aurora PostgreSQL 的每次调用摘要统计数据
- Aurora PostgreSQL 的主要统计数据

Aurora PostgreSQL 的摘要统计数据

要查看 SQL 摘要统计数据，则必须加载 `pg_stat_statements` 库。对于与 PostgreSQL 10 兼容的 Aurora PostgreSQL 数据库集群，默认为加载此库。对于与 PostgreSQL 9.6 兼容的 Aurora PostgreSQL 数据库集群，您可以手动启用此库。要手动启用该库，请在与数据库实例关联的数据库参数组中的 `pg_stat_statements` 中添加 `shared_preload_libraries`。然后重启数据库实例。有关更多信息，请参阅“Amazon Aurora 的参数组”。

> **Note**
>
> 性能详情只能收集 `pg_stat_activity` 中未被截断的查询的统计数据。默认情况下，PostgreSQL 数据库会截断长度超过 1,024 字节的查询。要增加查询大小，请更改与数据

库实例关联的数据库参数组中的 `track_activity_query_size` 参数。更改此参数后，需要重新启动数据库实例。

Aurora PostgreSQL 的每秒摘要统计数据

以下 SQL 摘要统计数据可用于 Aurora PostgreSQL 数据库实例。

指标	单位
db.sql_tokenized.stats.calls_per_sec	每秒调用数
db.sql_tokenized.stats.rows_per_sec	每秒行数
db.sql_tokenized.stats.total_time_per_sec	每秒平均活动执行次数 (AAE)
db.sql_tokenized.stats.shared_blks_hit_per_sec	每秒块命中次数
db.sql_tokenized.stats.shared_blks_read_per_sec	每秒块读取次数
db.sql_tokenized.stats.shared_blks_dirtied_per_sec	每秒损坏的块数
db.sql_tokenized.stats.shared_blks_written_per_sec	每秒块写入次数
db.sql_tokenized.stats.local_blks_hit_per_sec	每秒本地块命中次数
db.sql_tokenized.stats.local_blks_read_per_sec	每秒本地块读取次数
db.sql_tokenized.stats.local_blks_dirtied_per_sec	每秒弄脏的本地块数
db.sql_tokenized.stats.local_blks_written_per_sec	每秒本地块写入次数
db.sql_tokenized.stats.temp_blks_written_per_sec	每秒临时写入次数

指标	单位
db.sql_tokenized.stats.temp_blks_read_per_sec	每秒临时读取次数
db.sql_tokenized.stats.blk_read_time_per_sec	每秒平均并发读取次数
db.sql_tokenized.stats.blk_write_time_per_sec	每秒平均并发写入次数

Aurora PostgreSQL 的每次调用摘要统计数据

以下指标提供了 SQL 语句的每次调用统计数据。

指标	单位
db.sql_tokenized.stats.rows_per_call	每个调用的行数
db.sql_tokenized.stats.avg_latency_per_call	每个调用的平均延迟（以毫秒为单位）
db.sql_tokenized.stats.shared_blks_hit_per_call	每个调用的块命中次数
db.sql_tokenized.stats.shared_blks_read_per_call	每个调用的块读取次数
db.sql_tokenized.stats.shared_blks_written_per_call	每个调用的块写入次数
db.sql_tokenized.stats.shared_blks_dirtied_per_call	每个调用损坏的块数
db.sql_tokenized.stats.local_blks_hit_per_call	每个调用的本地块命中次数
db.sql_tokenized.stats.local_blks_read_per_call	每个调用的本地块读取次数
db.sql_tokenized.stats.local_blks_dirtied_per_call	每个调用弄脏的本地块数
db.sql_tokenized.stats.local_blks_written_per_call	每个调用的本地块写入次数

指标	单位
db.sql_tokenized.stats.temp_blks_written_per_call	每个调用的临时块写入次数
db.sql_tokenized.stats.temp_blks_read_per_call	每个调用的临时块读取次数
db.sql_tokenized.stats.blk_read_time_per_call	每个调用的读取时间（以毫秒为单位）
db.sql_tokenized.stats.blk_write_time_per_call	每个调用的写入时间（以毫秒为单位）

Aurora PostgreSQL 的主要统计数据

以下 SQL 统计数据可用于 Aurora PostgreSQL 数据库实例。

指标	单位
db.sql_tokenized.stats.calls	Calls
db.sql_tokenized.stats.rows	行
db.sql_tokenized.stats.total_time	总时间（以毫秒为单位）
db.sql_tokenized.stats.shared_blks_hit	块命中次数
db.sql_tokenized.stats.shared_blks_read	块读取次数
db.sql_tokenized.stats.shared_blks_dirtied	弄脏的块数
db.sql_tokenized.stats.shared_blks_written	块写入次数
db.sql_tokenized.stats.local_blks_hit	本地块命中次数
db.sql_tokenized.stats.local_blks_read	本地块读取次数
db.sql_tokenized.stats.local_blks_dirtied	弄脏的本地块数
db.sql_tokenized.stats.local_blks_written	本地块写入次数
db.sql_tokenized.stats.temp_blks_written	临时写入次数

指标	单位
db.sql_tokenized.stats.temp_blks_read	临时读取次数
db.sql_tokenized.stats.blk_read_time	平均并发读取次数（以毫秒为单位）
db.sql_tokenized.stats.blk_write_time	平均并发写入次数（以毫秒为单位）

有关这些指标的更多信息，请参阅 PostgreSQL 文档中的 pg_stat_statements。

增强监控中的操作系统指标

Amazon Aurora 为数据库集群运行于的操作系统 (OS) 实时提供指标。Aurora 将增强监控中的指标传输到您的 Amazon CloudWatch Logs 账户。下列各表列出了使用 Amazon CloudWatch Logs 可获得的操作系统指标。

主题

- Aurora 的操作系统指标

Aurora 的操作系统指标

组	指标	控制台名称	描述
General	engine	不适用	数据库实例的数据库引擎。
	instanceID	不适用	数据库实例标识符。
	instanceResourceID	不适用	数据库实例的不可变标识符，此标识符对于 AWS 区域是唯一的，也用作日志流标识符。
	numVCPUs	不适用	数据库实例的虚拟 CPU 数量。
	timestamp	不适用	提取指标的时间。

组	指标	控制台名称	描述
	uptime	不适用	数据库实例处于活动状态的时间量。
	version	不适用	操作系统指标的流 JSON 格式版本。
cpuUtilization	guest	CPU Guest (CPU 访客)	来宾程序使用的 CPU 百分比。
	idle	CPU Idle (CPU 空闲)	CPU 空闲百分比。
	irq	CPU IRQ	软件中断使用的 CPU 百分比。
	nice	CPU Nice (CPU 良好)	以最低优先级运行的程序使用的 CPU 百分比。
	steal	CPU Steal (CPU 被盗用)	其他虚拟机使用的 CPU 百分比。
	system	CPU System (CPU 系统)	内核使用的 CPU 百分比。
	total	CPU Total (CPU 总计)	使用中的 CPU 百分比总计。此值包含 nice 值。
	user	CPU User (CPU 用户)	用户程序使用的 CPU 百分比。
	wait	CPU Wait (CPU 等待)	等待 I/O 访问时的未使用 CPU 百分比。
diskIO	avgQueueLen	Avg Queue Size (平均队列大小)	在 I/O 设备队列中等待的请求数。

组	指标	控制台名称	描述
	avgReqSz	Ave Request Size (平均请求大小)	平均请求大小 (以 KB 为单位)。
	await	Disk I/O Await (磁盘 I/O 等待)	响应请求所需的毫秒数，包括排队时间和服务时间。
	device	不适用	使用中的磁盘设备的标识符。
	readIOsPS	Read IO/s (读取 IO/秒)	每秒的读取操作数。
	readKb	Read Total (读取总计)	读取的总 KB 数。
	readKbPS	Read Kb/s (读取 Kb/秒)	每秒读取的 KB 数。
	readLatency	Read Latency (读取延迟)	从提交读取 I/O 请求到完成该请求所经过的时间（以毫秒为单位）。 此指标仅对 Amazon Aurora 可用。
	readThroughput	Read Throughput (读取吞吐量)	向数据库集群发出的请求要使用的网络吞吐量（以字节/秒为单位）。 此指标仅对 Amazon Aurora 可用。
	rrqmPS	Rrqms	每秒排队的合并读取请求数。
	tps	TPS	每秒的 I/O 事务数。

组	指标	控制台名称	描述
	util	Disk I/O Util (磁盘 I/O 利用率)	发出请求所经历的 CPU 时间的百分比。
	writeIOsPS	Write IO/s (写入 IO/秒)	每秒的写入操作数。
	writeKb	Write Total (写入总计)	写入的总 KB 数。
	writeKbPS	Write Kb/s (写入 Kb/秒)	每秒写入的 KB 数。
	writeLatency	Write Latency (写入延迟)	从提交写入 I/O 请求到完成该请求所经历的平均时间（以毫秒为单位）。 此指标仅对 Amazon Aurora 可用。
	writeThroughput	Write Throughput (写入吞吐量)	来自数据库集群的响应所使用的网络吞吐量（以字节/秒为单位）。 此指标仅对 Amazon Aurora 可用。
	wrqmPS	Wrqms	每秒排队的合并写入请求数。
fileSys	maxFiles	Max Inodes (最大 Inode 数)	可为文件系统创建的文件的最大数量。
	mountPoint	不适用	文件系统的路径。
	name	不适用	文件系统的名称。

组	指标	控制台名称	描述
	total	Total Filesystem (文件系统总计)	文件系统的可用磁盘空间总量 (以 KB 为单位)。
	used	Used Filesystem (已使用的文件系统)	文件系统中的文件所用的磁盘空间量 (以 KB 为单位)。
	usedFilePercent	Used Inodes (已使用 Inode)	使用中的可用文件百分比。
	usedFiles	已使用百分比	文件系统中的文件数。
	usedPercent	Used Filesystem (已使用的文件系统)	使用中的文件系统磁盘空间百分比。
loadAverageMinute	fifteen	Load Avg 15 min (负载平均 15 分钟)	过去 15 分钟内请求 CPU 时间的进程数。
	five	Load Avg 5 min (负载平均 5 分钟)	过去 5 分钟内请求 CPU 时间的进程数。
	one	Load Avg 1 min (负载平均 1 分钟)	过去 1 分钟内请求 CPU 时间的进程数。

组	指标	控制台名称	描述
memory	active	Active Memory (活动内存)	已分配的内存量 (以 KB 为单位)。
	buffers	Buffered Memory (缓冲内存)	在写入存储设备前用于缓存 I/O 请求的内存量 (以 KB 为单位)。
	cached	Cached Memory (缓存内存)	用于缓存基于文件系统的输入/输出的内存量。
	dirty	Dirty Memory (脏内存)	RAM 中已修改但未写入存储中的相关数据块的内存页面大小 (以 KB 为单位)。
	free	Free Memory (空闲内存)	未分配的内存量 (以 KB 为单位)。
	hugePagesFree	Huge Pages Free (可用大页)	可用大页数。大页是 Linux 内核的一项功能。
	hugePagesRsvd	Huge Pages Rsvd (预留大页)	已提交大页数。
	hugePagesSize	Huge Pages Size (大页大小)	每个大页单位的大小 (以 KB 为单位)。

组	指标	控制台名称	描述
	hugePagesSurp	Huge Pages Surp (剩余大页)	剩余可用大页总数。
	hugePagesTotal	Huge Pages Total (大页总计)	大页总数。
	inactive	Inactive Memory (非活动内存)	最不常用内存页面大小 (以 KB 为单位)。
	mapped	Mapped Memory (映射的内存)	在进程地址空间中有内存映射的文件系统内容的总大小 (以 KB 为单位)。
	pageTables	Page Tables (页表)	页表使用的内存量 (以 KB 为单位)。
	slab	Slab Memory (Slab 内存)	可重用内核数据结构大小 (以 KB 为单位)。
	total	总内存	内存总量 (以 KB 为单位)。
	writeback	Writeback Memory (回写内存)	RAM 中仍在写入备份存储的脏页大小 (以 KB 为单位)。
network	interface	不适用	数据库实例现用网络接口的标识符。

组	指标	控制台名称	描述
	rx	RX	每秒接收的字节数。
	tx	TX	每秒上传的字节数。
processList	cpuUsedPc	CPU % (CPU 百分比)	进程使用的 CPU 百分比。
	id	不适用	进程的标识符。
	memoryUsedPc	MEM%	进程使用的内存百分比。
	name	不适用	进程的名称。
	parentID	不适用	进程的父进程的进程标识符。
	rss	RES	分配给进程的 RAM 量 (以 KB 为单位)。
	tgid	不适用	线程组标识符，是表示线程所属进程的 ID 的数字。此标识符用于将同一进程内的线程分入一组。
	vss	VIRT	分配给进程的虚拟内存量 (以 KB 为单位)。
swap	total	Swap (交换)	可用的交换内存量（以 KB 为单位）。
	in	Swaps in (换入)	从磁盘换入的内存量（以 KB 为单位）。
	out	Swaps out (换出)	换出到磁盘的内存量（以 KB 为单位）。
	free	Free Swap (可用交换)	空闲交换内存量（以 KB 为单位）。

组	指标	控制台名称	描述
	cached	Committed Swap (已提交的交换)	用作缓存内存的交换内存量 (以 KB 为单位)。
tasks	blocked	Tasks Blocked (阻止的任务)	已阻止的任务的数量。
	running	Tasks Running (正在运行的任务)	正在运行的任务的数量。
	sleeping	Tasks Sleeping (正在休眠的任务)	正在睡眠的任务的数量。
	stopped	Tasks Stopped (已停止的任务)	已停止的任务的数量。
	total	Tasks Total (任务总计)	任务总数。
	zombie	Tasks Zombie (停滞的任务)	有活动父任务的不活动子任务的数量。

监控 Amazon Aurora 数据库集群中的事件、日志和流

监控您的 Amazon Aurora 数据库和其他 AWS 解决方案时，您的目标是维持以下各项：

- 可靠性
- 可用性
- 性能
- 安全性

监控 Amazon Aurora 集群中的指标 说明如何使用指标监控集群。完整的解决方案还必须监控数据库事件、日志文件和活动流。AWS 为您提供以下监控工具：

- Amazon EventBridge 是一种无服务器事件总线服务，可以轻松地将应用程序与来自各种来源的数据相连接。EventBridge 可以从您自己的应用程序、软件即服务（SaaS）应用程序和 AWS 服务传输实时数据流。EventBridge 将该数据路由到诸如 AWS Lambda 之类的目标。这样，您就可以监控服务中发生的事件，并构建事件驱动的架构。有关更多信息，请参阅 Amazon EventBridge 用户指南。
- Amazon CloudWatch Logs 提供了一种方法，让您可以监控、存储和访问来自 Amazon Aurora 实例、AWS CloudTrail 和其他来源的日志文件。Amazon CloudWatch Logs 可以监控日志文件中的信息，并在达到特定阈值时通知您。您还可以在高持久性存储中检索您的日志数据。有关更多信息，请参阅 Amazon CloudWatch Logs 用户指南。
- AWS CloudTrail 捕获由某个 AWS 账户 发出或代表该账户发出的 API 调用和相关事件。CloudTrail 将日志文件传送到您指定的 Amazon S3 桶。您可以标识哪些用户和账户调用了 AWS、发出调用的源 IP 地址以及调用的发生时间。有关更多信息，请参阅 AWS CloudTrail《用户指南》。
- 数据库活动流是一项 Amazon Aurora 功能，它提供数据库集群中近乎实时的活动流。Amazon Aurora 会将活动推送到 Amazon Kinesis 数据流。系统将自动创建 Kinesis 流。在 Kinesis 中，您可以配置 AWS 服务（如 Amazon Data Firehose）和 AWS Lambda 来使用 Kinesis 流并存储数据。

主题

- 在 Amazon RDS 控制台中查看日志、事件和流
- 监控 Amazon Aurora 事件
- 监控 Amazon Aurora 日志文件
- 监控 AWS CloudTrail 中的 Amazon Aurora API 调用
- 使用数据库活动流监控 Amazon Aurora

- 使用适用于 Amazon Aurora 的 Amazon GuardDuty RDS 保护功能监控威胁

在 Amazon RDS 控制台中查看日志、事件和流

Amazon RDS 与 AWS 服务 集成，以在 RDS 控制台中显示关于日志、事件和数据库活动流的信息。

Aurora 数据库集群的 Logs & events（日志和事件）选项卡显示以下信息：

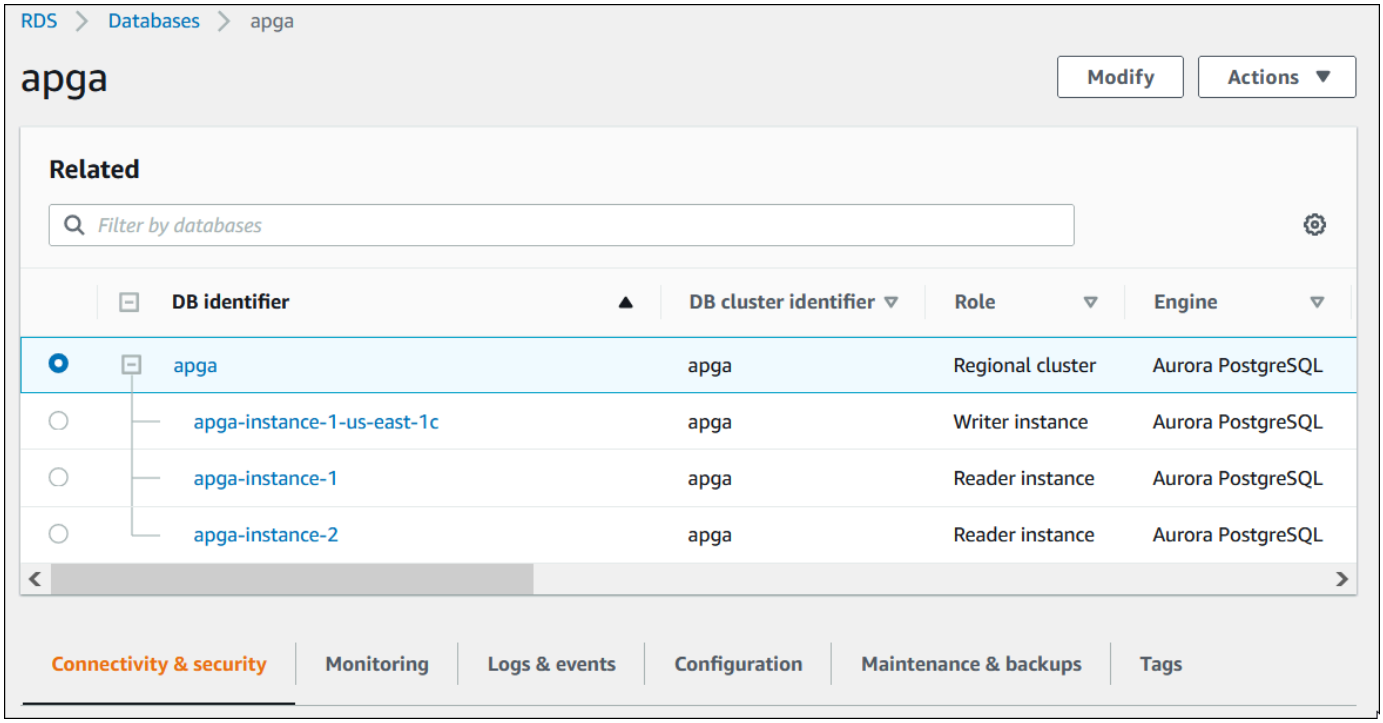
- 自动伸缩策略和活动 - 显示与 Aurora Auto Scaling 功能有关的策略和活动。此信息仅在集群级别的 Logs & events（日志和事件）选项卡上显示。
- Amazon CloudWatch 告警 - 显示为 Aurora 集群中的数据库实例配置的任何指标告警。如果尚未配置告警，您可以在 RDS 控制台中创建告警。
- Recent events（近期事件）- 显示 Aurora 数据库实例或集群事件（环境变更）的摘要。有关更多信息，请参阅 查看 Amazon RDS 事件。
- Logs（日志）- 显示 Aurora 集群中数据库实例生成的数据库日志文件。有关更多信息，请参阅 监控 Amazon Aurora 日志文件。

Configuration（配置）选项卡显示关于数据库活动流的信息。

在 RDS 控制台中查看 Aurora 数据库集群的日志、事件和流

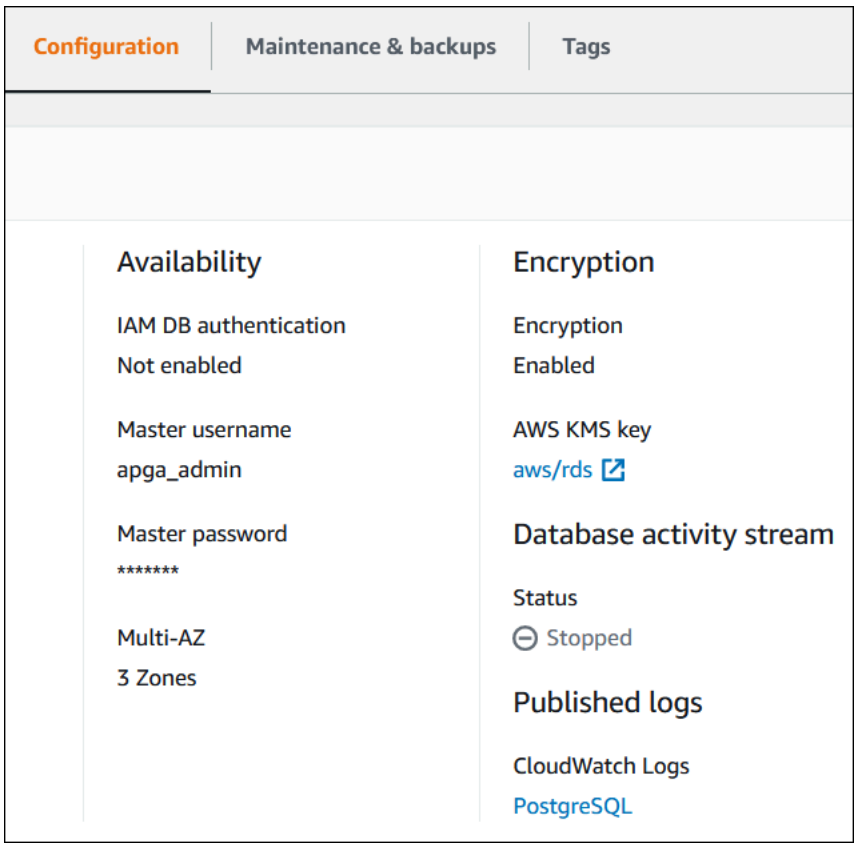
1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Databases（数据库）。
3. 选择要监控的 Aurora 数据库集群的名称。

   随后会显示数据库页面。以下示例显示名为 apga 的 Amazon Aurora PostgreSQL 数据库集群。



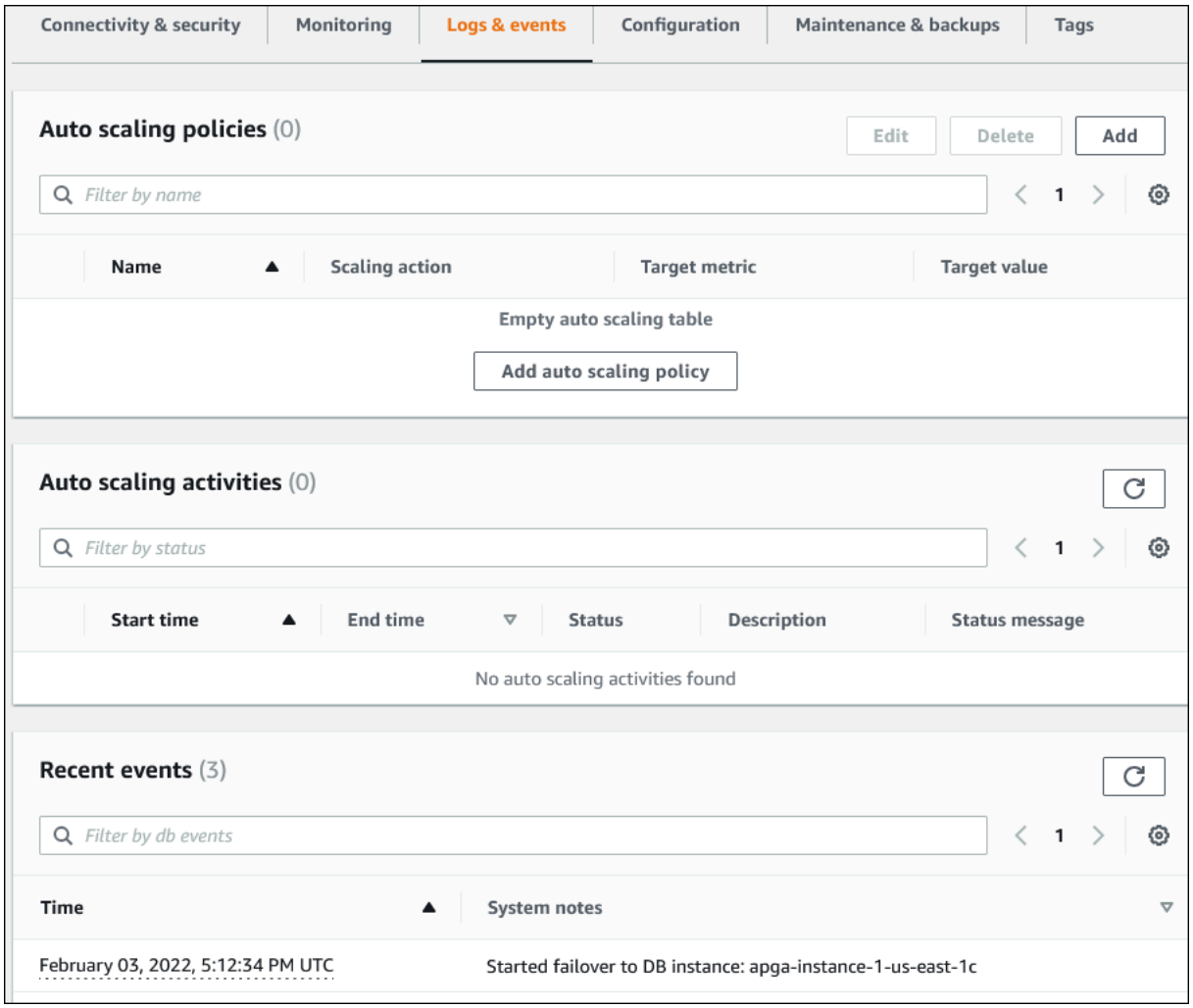
4. 向下滚动并选择 Configuration（配置）。

以下示例显示集群的数据库活动流的状态。



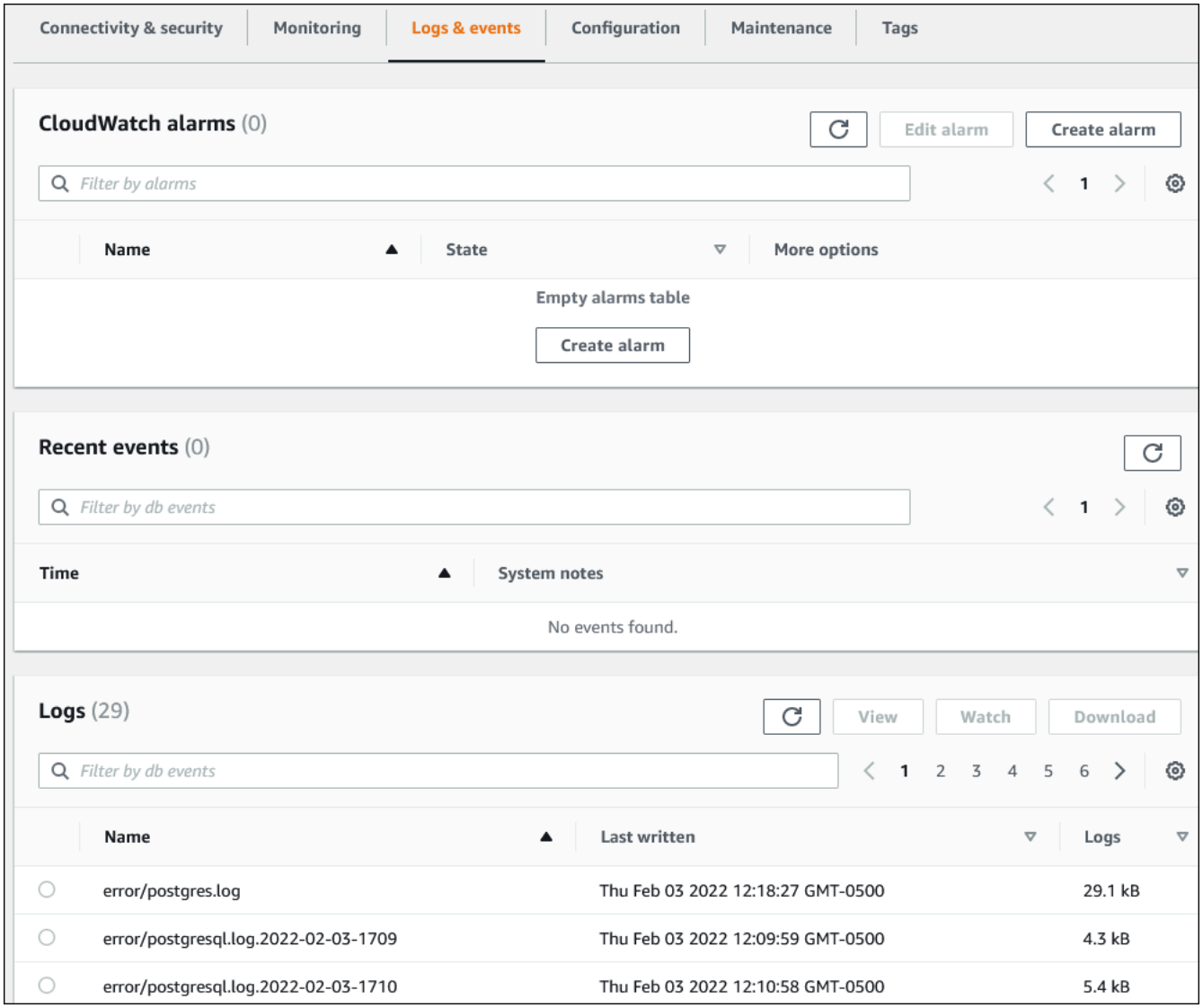
5. 选择日志和事件。

   此时将显示 Logs & events（日志和事件）部分。



6. 在 Aurora 集群中选择一个数据库实例，然后为该实例选择 Logs & events（日志和事件）。

   以下示例显示数据库实例页面和数据库集群页面之间的内容不同。数据库实例页面显示日志和告警。


Connectivity & security
Monitoring
Logs & events
Configuration
Maintenance
Tags
CloudWatch alarms (0)
Edit alarm
Create alarm
Filter by alarms
1
Name
State
More options
Empty alarms table
Create alarm
Recent events (0)
Filter by db events
1
Time
System notes
No events found.
Logs (29)
View
Watch
Download
Filter by db events
1 2 3 4 5 6
Name
Last written
Logs
error/postgres.log
Thu Feb 03 2022 12:18:27 GMT-0500
29.1 kB
error/postgresql.log.2022-02-03-1709
Thu Feb 03 2022 12:09:59 GMT-0500
4.3 kB
error/postgresql.log.2022-02-03-1710
Thu Feb 03 2022 12:10:58 GMT-0500
5.4 kB

监控 Amazon Aurora 事件

事件表示环境中的更改。这可以是 AWS 环境、SaaS 合作伙伴服务或应用程序，或自定义应用程序或服务。有关 Aurora 事件的说明，请参阅 Aurora 的 Amazon RDS 事件类别和事件消息。

主题

- Aurora 事件概述
- 查看 Amazon RDS 事件
- 使用 Amazon RDS 事件通知
- 创建对 Amazon Aurora 事件触发的规则
- Aurora 的 Amazon RDS 事件类别和事件消息

Aurora 事件概述

RDS 事件表示 Aurora 环境中的更改。例如，当数据库集群经过修补时，Amazon Aurora 将生成事件。Amazon Aurora 将事件近乎实时地传输到 EventBridge。

> **Note**
>
> Amazon RDS 尽最大努力发出事件。我们建议您避免编写取决于通知事件的顺序或存在的程序，因为它们可能是乱序或缺失的。

Amazon RDS 记录与以下资源相关的事件：

- 数据库集群

  有关集群事件的列表，请参阅 数据库集群事件。
- 数据库实例

  有关数据库实例事件的列表，请参阅 数据库实例事件。
- 数据库参数组

  有关数据库参数组事件的列表，请参阅 数据库参数组事件。
- 数据库安全组

  有关数据库安全组事件的列表，请参阅 数据库安全组事件。

- 数据库集群快照

  有关数据库集群快照事件的列表，请参阅 数据库集群快照事件。

- RDS Proxy 事件

  有关 RDS Proxy 事件的列表，请参阅 RDS Proxy 事件。

- 蓝绿部署事件

  有关蓝绿部署事件的列表，请参阅 蓝绿部署事件。

这些信息包含：

- 事件的日期和时间
- 事件的源名称和源类型
- 与事件关联的消息
- 事件通知包括发送消息时的标签，可能不反映事件发生时的标签。

查看 Amazon RDS 事件

您可以检索 Amazon Aurora 资源的以下事件信息：

- 资源名称
- 资源类型
- 事件的时间
- 事件的消息摘要

您可以访问 AWS Management Console的以下各部分中的事件：

- 事件选项卡，显示过去 24 小时的事件。
- 数据库选项卡中日志和事件部分的近期事件表，可以显示多达过去 2 周的事件。

您还可以通过使用 describe-events AWS CLI 命令或 DescribeEvents RDS API 操作来检索事件。如果您使用 AWS CLI 或 RDS API 查看事件，则可检索过去长达 14 天内的事件。

> **Note**
>
> 如果需要将事件存储更长的时间段，则可以将 Amazon RDS 事件发送到 EventBridge。有关更多信息，请参阅 创建对 Amazon Aurora 事件触发的规则

有关 Amazon Aurora 事件的说明，请参阅 Aurora 的 Amazon RDS 事件类别和事件消息。

要访问有关使用 AWS CloudTrail 的事件的详细信息，包括请求参数，请参阅 CloudTrail 事件。

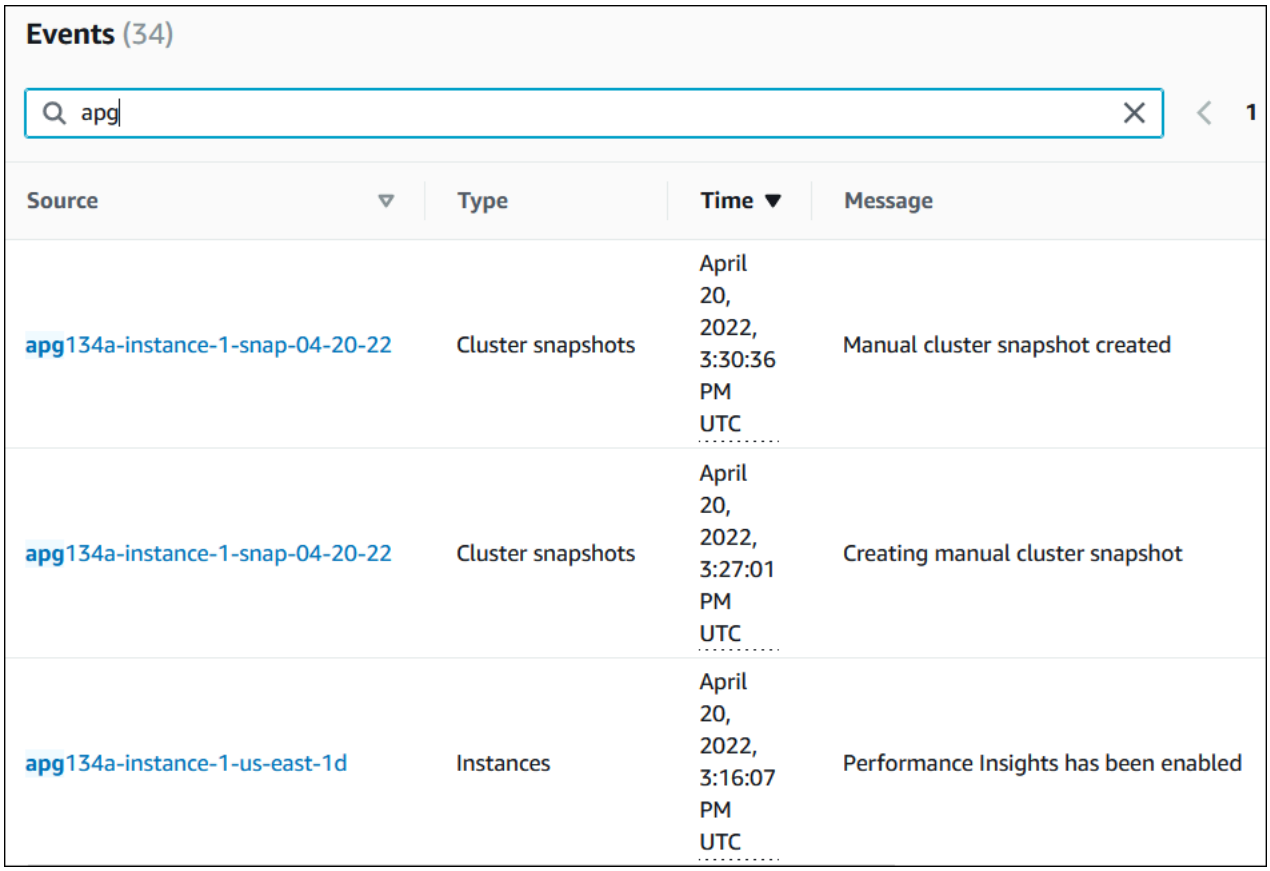
控制台

查看过去 24 小时的所有 Amazon RDS 事件

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Events (事件)。

   列表中显示可用的事件。
3. （可选）输入搜索词以筛选结果。

以下示例显示按字符 **apg** 筛选的事件列表。



AWS CLI

要查看过去一小时内生成的所有事件，请调用 describe-events，不带参数。

```
aws rds describe-events
```

以下示例输出显示数据库集群实例已开始恢复。

```
{
    "Events": [
        {
            "EventCategories": [
                "recovery"
            ],
            "SourceType": "db-instance",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:db:mycluster-instance-1",
            "Date": "2022-04-20T15:02:38.416Z",
```

```
            "Message": "Recovery of the DB instance has started. Recovery time will
 vary with the amount of data to be recovered.",
            "SourceIdentifier": "mycluster-instance-1"
        }, ...
```

要查看过去 10080 分钟（7 天）的所有 Amazon RDS 事件，请调用 describe-events AWS CLI 命令并将 `--duration` 参数设置为 `10080`。

```
aws rds describe-events --duration 10080
```

以下示例显示数据库实例 *test-instance* 在指定时间范围内的事件。

```
aws rds describe-events \
    --source-identifier test-instance \
    --source-type db-instance \
    --start-time 2022-03-13T22:00Z \
    --end-time 2022-03-13T23:59Z
```

以下示例输出显示备份的状态。

```
{
    "Events": [
        {
            "SourceType": "db-instance",
            "SourceIdentifier": "test-instance",
            "EventCategories": [
                "backup"
            ],
            "Message": "Backing up DB instance",
            "Date": "2022-03-13T23:09:23.983Z",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:db:test-instance"
        },
        {
            "SourceType": "db-instance",
            "SourceIdentifier": "test-instance",
            "EventCategories": [
                "backup"
            ],
            "Message": "Finished DB Instance backup",
            "Date": "2022-03-13T23:15:13.049Z",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:db:test-instance"
        }
```

```
    ]
}
```

API

通过调用 DescribeEvents RDS API 操作并将 `Duration` 参数设置为 `20160`，您可查看过去 14 天内的所有 Amazon RDS 实例事件。

使用 Amazon RDS 事件通知

Amazon RDS 使用 Amazon Simple Notification Service (Amazon SNS) 在发生 Amazon RDS 事件时提供通知。这些通知可以采用 AWS 区域 Amazon SNS 支持的任何通知形式，例如电子邮件、文本消息或对 HTTP 终端节点的调用。

主题

- Amazon RDS 事件通知概述
- 授予向 Amazon SNS 主题发布通知的权限。
- 订阅 Amazon RDS 事件通知
- Amazon RDS 事件通知标签和属性
- 列出 Amazon RDS 事件通知订阅
- 修改 Amazon RDS 事件通知订阅
- 在 Amazon RDS 事件通知订阅中添加源标识符
- 从 Amazon RDS 事件通知订阅中删除源标识符
- 列出 Amazon RDS 事件通知类型
- 删除 Amazon RDS 事件通知订阅

Amazon RDS 事件通知概述

Amazon RDS 将事件分组为您可以订阅的类型，以便您在出现该类事件时收取通知。

主题

- 符合事件订阅条件的 RDS 资源
- 订阅 Amazon RDS 事件通知的基本流程
- 发送 RDS 事件通知
- Amazon RDS 事件通知的计费
- 使用 Amazon EventBridge 的 Aurora 事件的示例

符合事件订阅条件的 RDS 资源

对于 Amazon Aurora，事件在数据库集群和数据库实例级别发生。您可以订阅以下资源的事件类别：

- 数据库实例
- 数据库群集

- 数据库集群快照
- 数据库参数组
- 数据库安全组
- RDS 代理
- 自定义引擎版本

例如，如果您订阅给定数据库实例的备份类别，那么无论何时出现影响该数据库实例的备份相关事件，您都将收到通知。如果您订阅对于数据库实例的配置更改类别，则会在数据库实例出现更改时收到通知。您还将在事件通知订阅更改时收到通知。

您可能需要创建多个不同的订阅。例如，您可能创建一个订阅以接收所有数据库实例的所有事件通知，并创建另一个订阅，其中仅包括数据库实例子集的严重事件。对于第二个订阅，请在筛选条件中指定一个或多个数据库实例。

订阅 Amazon RDS 事件通知的基本流程

订阅 Amazon RDS 事件通知的流程如下：

1. 您使用 Amazon RDS 控制台、AWS CLI 或者 API 创建 Amazon RDS 事件通知订阅。

   Amazon RDS 使用 Amazon SNS 主题的 ARN 标识每个订阅。Amazon RDS 控制台在您创建订阅时为您创建 ARN。使用 Amazon SNS 控制台、AWS CLI 或 Amazon SNS API 创建 ARN。
2. Amazon RDS 发送批准电子邮件或者 SMS 消息给您在订阅时提交的地址。
3. 您可以通过选择所收到的通知中的链接来确认您的订阅。
4. Amazon RDS 控制台更新 My Event Subscriptions（我的事件订阅）部分，其中包含您的订阅状态。
5. Amazon RDS 开始将通知发送到您在创建订阅时提供的地址。

要了解使用 Amazon SNS 时的 Identity of Access Management，请查看 Amazon Simple Notification Service 开发人员指南中的 Amazon SNS 中的 Identity of Access Management。

您可以使用处理 AWS Lambda 来自数据库实例的事件通知。有关更多信息，请参阅 AWS Lambda 开发人员指南中的将 AWS Lambda 与 Amazon RDS 结合使用。

发送 RDS 事件通知

Amazon RDS 会将通知发送到您在创建订阅时提供的地址。通知可以包含消息属性，以提供有关消息的结构化元数据。有关消息属性的更多信息，请参阅Aurora 的 Amazon RDS 事件类别和事件消息。

事件通知的传递可能需要长达五分钟。

> **Important**
>
> Amazon RDS 不保证在事件流中发送的事件的顺序。事件顺序可能会发生变化。

当 Amazon SNS 向已经订阅的 HTTP 或 HTTPS 端点发送一条通知时，发至端点的 POST 消息具有包含 JSON 格式文档的消息正文。有关更多信息，请参阅 Amazon Simple Notification Service 开发人员指南 中的 Amazon SNS 消息和 JSON 格式。

您可以将 SNS 配置为通过短信通知。有关更多信息，请参阅 Amazon Simple Notification Service 开发人员指南中的移动文本消息 (SMS)。

要在不删除订阅的情况下关闭通知，请在 Amazon RDS 控制台中为已启用选择否。或者，可以使用 AWS CLI 或 Amazon RDS API 将 `Enabled` 参数设置为 `false`。

Amazon RDS 事件通知的计费

Amazon RDS 事件通知的计费是通过 Amazon SNS 执行的。Amazon SNS 费用在使用事件通知时适用。有关 Amazon SNS 计费的更多信息，请参阅 Amazon Simple Notification Service 定价。

使用 Amazon EventBridge 的 Aurora 事件的示例

以下示例说明了 JSON 格式的不同类型的 Aurora 事件。有关向您说明如何以 JSON 格式捕获和查看事件的教程，请参阅教程：使用 Amazon EventBridge 记录数据库实例的状态更改。

主题

- 数据库集群事件的示例
- 数据库参数组事件的示例
- 数据库集群快照事件的示例

数据库集群事件的示例

以下是采用 JSON 格式的数据库集群事件的示例。该事件显示名为 `my-db-cluster` 的集群已修补。事件 ID 为 `RDS-EVENT-0173`。

```
{
  "version": "0",
```

```
  "id": "844e2571-85d4-695f-b930-0153b71dcb42",
  "detail-type": "RDS DB Cluster Event",
  "source": "aws.rds",
  "account": "123456789012",
  "time": "2018-10-06T12:26:13Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:rds:us-east-1:123456789012:cluster:my-db-cluster"
  ],
  "detail": {
    "EventCategories": [
      "notification"
    ],
    "SourceType": "CLUSTER",
    "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:my-db-cluster",
    "Date": "2018-10-06T12:26:13.882Z",
    "Message": "Database cluster has been patched",
    "SourceIdentifier": "my-db-cluster",
    "EventID": "RDS-EVENT-0173"
  }
}
```

数据库参数组事件的示例

以下是采用 JSON 格式的数据库参数组事件的示例。该事件显示参数 time_zone 已在参数组 my-db-param-group 中更新。事件 ID 为 RDS-EVENT-0037。

```
{
  "version": "0",
  "id": "844e2571-85d4-695f-b930-0153b71dcb42",
  "detail-type": "RDS DB Parameter Group Event",
  "source": "aws.rds",
  "account": "123456789012",
  "time": "2018-10-06T12:26:13Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:rds:us-east-1:123456789012:pg:my-db-param-group"
  ],
  "detail": {
    "EventCategories": [
      "configuration change"
    ],
    "SourceType": "DB_PARAM",
```

```
    "SourceArn": "arn:aws:rds:us-east-1:123456789012:pg:my-db-param-group",
    "Date": "2018-10-06T12:26:13.882Z",
    "Message": "Updated parameter time_zone to UTC with apply method immediate",
    "SourceIdentifier": "my-db-param-group",
    "EventID": "RDS-EVENT-0037"
  }
}
```

数据库集群快照事件的示例

以下是采用 JSON 格式的数据库集群快照事件的示例。该事件显示名为 my-db-cluster-snapshot 的快照的创建情况。事件 ID 为 RDS-EVENT-0074。

```
{
  "version": "0",
  "id": "844e2571-85d4-695f-b930-0153b71dcb42",
  "detail-type": "RDS DB Cluster Snapshot Event",
  "source": "aws.rds",
  "account": "123456789012",
  "time": "2018-10-06T12:26:13Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:rds:us-east-1:123456789012:cluster-snapshot:rds:my-db-cluster-snapshot"
  ],
  "detail": {
    "EventCategories": [
      "backup"
    ],
    "SourceType": "CLUSTER_SNAPSHOT",
    "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster-snapshot:rds:my-db-
cluster-snapshot",
    "Date": "2018-10-06T12:26:13.882Z",
    "SourceIdentifier": "my-db-cluster-snapshot",
    "Message": "Creating manual cluster snapshot",
    "EventID": "RDS-EVENT-0074"
  }
}
```

授予向 Amazon SNS 主题发布通知的权限。

要授予 Amazon RDS 将通知发布到 Amazon Simple Notification Service (Amazon SNS) 主题的权限，请将 AWS Identity and Access Management (IAM) 策略附加到目标主题。有关权限的更多信息，请参阅《Amazon Simple Notification Service 开发人员指南》中的 Amazon Simple Notification Service 访问控制示例案例。

原定设置情况下，Amazon SNS 主题具有一项策略，它允许同一账户中的所有 Amazon RDS 资源向其发布通知。您可以附加自定义策略以允许跨账户通知，或限制对某些资源的访问。

以下是您附加到目标 Amazon SNS 主题的 IAM 策略示例。它将主题限制为名称与指定前缀匹配的数据库实例。要使用此策略，请指定以下值：

- `Resource` – Amazon SNS 主题的 Amazon 资源名称 (ARN)
- `SourceARN` – 您的 RDS 资源 ARN
- `SourceAccount` – 您的 AWS 账户 ID

有关资源类型及其 ARN 的列表，请参阅《服务授权参考》中的 Amazon RDS 定义的资源。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "events.rds.amazonaws.com"
      },
      "Action": [
        "sns:Publish"
      ],
      "Resource": "arn:aws:sns:us-east-1:123456789012:topic_name",
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:rds:us-east-1:123456789012:db:prefix-*"
        },
        "StringEquals": {
          "aws:SourceAccount": "123456789012"
        }
```

```
      }
    }
  ]
}
```

订阅 Amazon RDS 事件通知

最简单的订阅创建方法是使用 RDS 控制台。如果您选择使用 CLI 或 API 创建事件通知，则必须创建 Amazon Simple Notification Service 主题并订阅有关 Amazon SNS 控制台或 Amazon SNS API 的主题。您还必须保留该主题的 Amazon Resource Name (ARN)，因为在提交 CLI 命令或者 API 操作时会用到。有关创建和订阅 SNS 主题的信息，请参阅 Amazon Simple Notification Service 开发人员指南中的 Amazon SNS 入门。

您可以指定希望收取其通知的源类型以及触发该事件的 Amazon RDS 源：

Source type（源类型）

源类型。例如，Source type（源类型）可能是 Instances（实例）。必须选择一种源类型。

Resources to include（要包含的资源）

生成事件的 Amazon RDS 资源。例如，您可以选择 Select specific instances（选择特定实例），然后选择 myDBInstance1。

下表说明了指定或不指定 ***Resources*** to include（要包含的资源）时的结果。

要包含的资源	描述	示例
指定	RDS 仅向您通知指定资源的所有事件。	如果 Source type（源类型）为 Instances（实例），资源为 myDBInstance1，则 RDS 仅向您通知 myDBInstance1 的所有事件。
未指定	RDS 会向您通知所有 Amazon RDS 资源的指定源类型的事件。	如果 Source type（源类型）是 Instances（实例），RDS 会向您通知您的账户中所有与实例相关的事件。

原定设置情况下，Amazon SNS 主题订阅用户会收到发布到该主题的每条消息。要仅接收一部分消息，订阅用户必须将筛选策略分配给主题订阅。有关更多信息，请参阅《Amazon Simple Notification Service 开发人员指南》中的 Amazon SNS 消息筛选。

控制台

订阅 RDS 事件通知

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择事件订阅。
3. 在事件订阅窗格中，选择创建事件订阅。
4. 输入您的订阅详情，如下所示：

   a. 对于名称，输入事件通知订阅的名称。

   b. 对于 Send notifications to（发送通知到），执行以下操作之一：

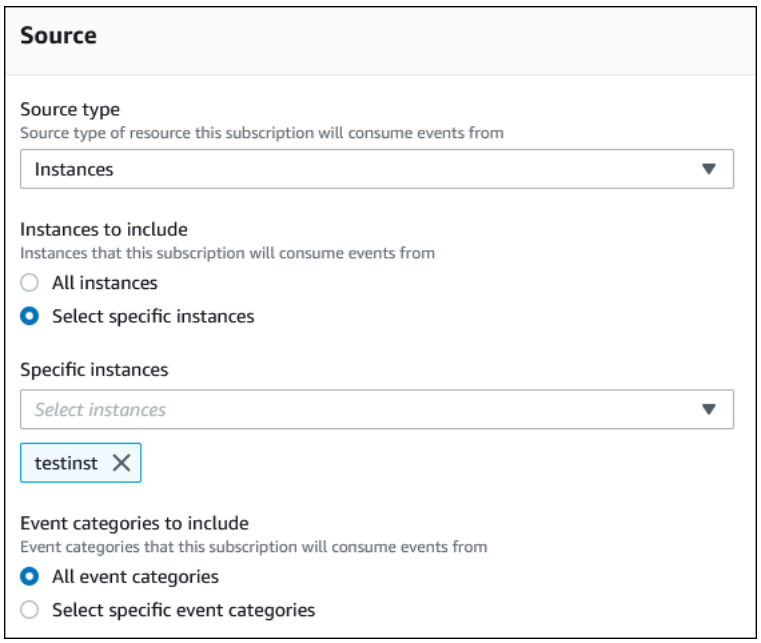
      - 选择 New email topic（新建电子邮件主题）。输入电子邮件主题的名称和收件人列表。我们建议您将事件订阅配置为与主要账户联系人相同的电子邮件地址。建议、服务事件和个人健康信息是通过不同的渠道发送的。订阅同一个电子邮件地址可确保将所有邮件合并到一个位置。
      - 选择 Amazon Resource Name (ARN) [Amazon 资源名称（ARN）]。然后，为 Amazon SNS 主题选择现有的 Amazon SNS ARN。

        如果您想使用已启用服务器端加密 (SSE) 的主题，请向 Amazon RDS 授予访问 AWS KMS key 的必需权限。有关更多信息，请参阅《Amazon Simple Notification Service 开发人员指南》中的实现 AWS 服务中的事件源与加密主题之间的兼容性。

   c. 对于源类型，请选择一种源类型。例如，选择 Clusters（集群）或 Cluster snapshots（集群快照）。

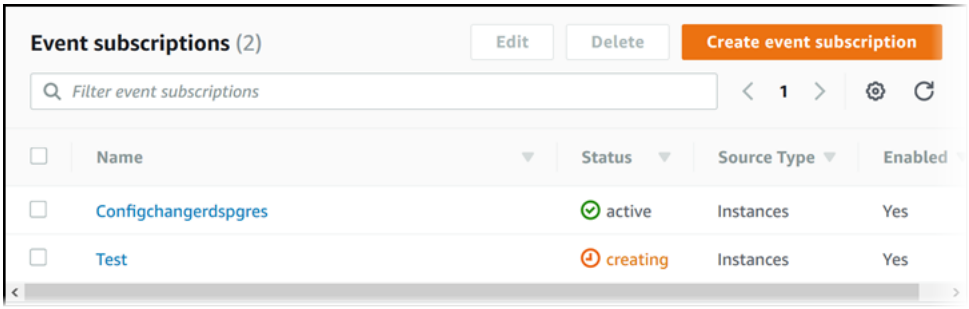
   d. 选择要接收事件通知的事件类别和资源。

      以下示例为名为 `testinst` 的数据库实例配置事件通知。



e. 选择创建。

Amazon RDS 控制台会表明正在创建订阅。



AWS CLI

要订阅 RDS 事件通知，请使用 AWS CLI `create-event-subscription` 命令。包括以下必需参数：

- `--subscription-name`
- `--sns-topic-arn`

Example

对于 Linux、macOS 或 Unix：

```
aws rds create-event-subscription \
```

```
    --subscription-name myeventsubscription \
    --sns-topic-arn arn:aws:sns:us-east-1:123456789012:myawsuser-RDS \
    --enabled
```

对于 Windows：

```
aws rds create-event-subscription ^
    --subscription-name myeventsubscription ^
    --sns-topic-arn arn:aws:sns:us-east-1:123456789012:myawsuser-RDS ^
    --enabled
```

API

要订阅 Amazon RDS 事件通知，请调用 Amazon RDS API 函数 CreateEventSubscription。包括以下必需参数：

- SubscriptionName
- SnsTopicArn

Amazon RDS 事件通知标签和属性

当 Amazon RDS 向 Amazon Simple Notification Service（SNS）或 Amazon EventBridge 发送事件通知时，通知包含消息属性和事件标签。RDS 将消息属性与消息一起单独发送，而事件标签位于消息正文中。使用消息属性和 Amazon RDS 标签将元数据添加到您的资源中。您可以使用您自己的关于数据库实例、Aurora 集群等的表示法来修改这些标签。有关标记 Amazon RDS 资源的更多信息，请参阅为 Amazon Aurora 和 Amazon RDS 资源添加标签。

默认情况下，Amazon SNS 和 Amazon EventBridge 会接收发送给它们的每条消息。SNS 和 EventBridge 可以筛选消息，并将通知发送到首选的通信模式，例如电子邮件、短信或对 HTTP 端点的调用。

> **Note**
>
> 通过电子邮件或短信发送的通知将没有事件标签。

下表显示了发送给主题订阅用户的 RDS 事件的消息属性。

Amazon RDS 事件属性	描述
EventID	RDS 事件消息的标识符，例如 RDS-EVENT-0006。
资源	发出事件的资源的 ARN 标识符，例如 `arn:aws:rds:ap-southeast-2:123456789012:db:database-1` 。

RDS 标签提供有关受服务事件影响的资源的数据。当通知发送到 SNS 或 EventBridge 时，RDS 会在消息正文中添加标签的当前状态。

有关筛选 SNS 的消息属性的更多信息，请参阅《Amazon Simple Notification Service 开发人员指南》中的 Amazon SNS 消息筛选。

有关筛选 EventBridge 的事件标签的更多信息，请参阅《Amazon EventBridge 用户指南》中的 Comparison operators for use in event patterns in Amazon EventBridge。

有关筛选 SNS 的基于有效载荷的标签的更多信息，请参阅 Introducing payload-based message filtering for Amazon SNS

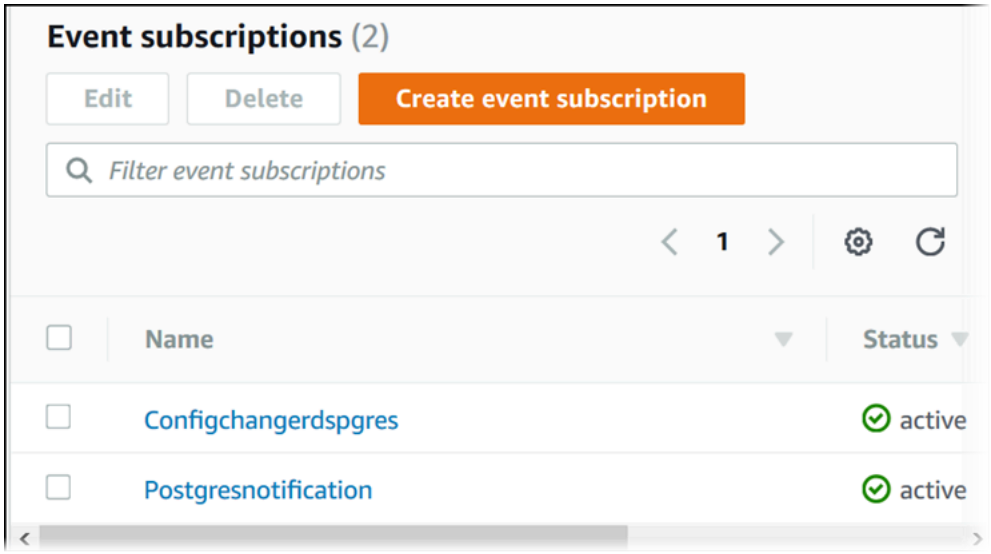
列出 Amazon RDS 事件通知订阅

您可以列出当前的 Amazon RDS 事件通知订阅。

控制台

列出当前的 Amazon RDS 事件通知订阅

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择事件订阅。事件订阅窗格中会显示您的所有事件通知订阅。



AWS CLI

要列出当前的 Amazon RDS 事件通知订阅，请使用 AWS CLI `describe-event-subscriptions` 命令。

Example

以下示例描述所有事件订阅。

```
aws rds describe-event-subscriptions
```

以下示例描述 `myfirsteventsubscription`。

```
aws rds describe-event-subscriptions --subscription-name myfirsteventsubscription
```

API

要列出当前的 Amazon RDS 事件通知订阅，请调用 Amazon RDS API `DescribeEventSubscriptions` 操作。

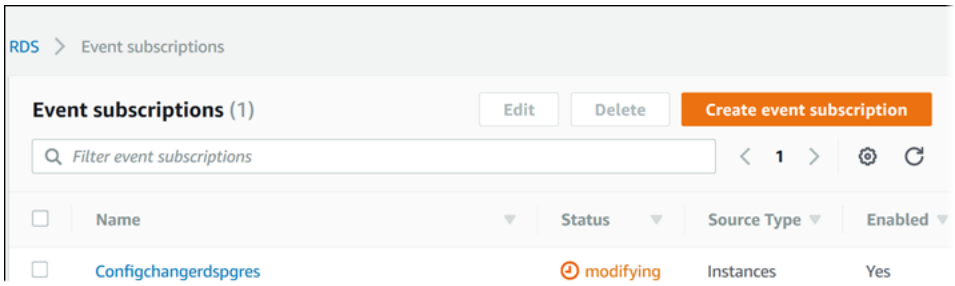
修改 Amazon RDS 事件通知订阅

创建订阅后，您可以更改订阅名称、源标识符、类别或主题 ARN。

控制台

修改 Amazon RDS 事件通知订阅

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择事件订阅。
3. 在事件订阅窗格中，选择您要修改的订阅，然后选择编辑。
4. 在目标或来源部分中对订阅进行更改。
5. 选择编辑。Amazon RDS 控制台会表明正在修改订阅。



AWS CLI

要修改 Amazon RDS 事件通知订阅，请使用 AWS CLI `modify-event-subscription` 命令。包括以下必需参数：

- `--subscription-name`

Example

以下代码实现 `myeventsubscription`。

对于 Linux、macOS 或 Unix：

```
aws rds modify-event-subscription \
    --subscription-name myeventsubscription \
    --enabled
```

对于 Windows：

```
aws rds modify-event-subscription ^
    --subscription-name myeventsubscription ^
    --enabled
```

API

要修改 Amazon RDS 事件，请调用 Amazon RDS API 操作 ModifyEventSubscription。包括以下必需参数：

- SubscriptionName

在 Amazon RDS 事件通知订阅中添加源标识符

您可以给现有的订阅添加源标识符 (生成事件的 Amazon RDS 源)。

控制台

您可以使用 Amazon RDS 控制台在修改订阅时通过选择或者取消选择操作轻松地添加或删除源标识符。有关更多信息，请参阅“修改 Amazon RDS 事件通知订阅”。

AWS CLI

要将源标识符添加到 Amazon RDS 事件通知订阅，请使用 AWS CLI `add-source-identifier-to-subscription` 命令。包括以下必需参数：

- `--subscription-name`
- `--source-identifier`

Example

以下示例将源标识符 `mysqldb` 添加到 `myrdseventsubscription` 订阅。

对于 Linux、macOS 或 Unix：

```
aws rds add-source-identifier-to-subscription \
    --subscription-name myrdseventsubscription \
    --source-identifier mysqldb
```

对于 Windows：

```
aws rds add-source-identifier-to-subscription ^
    --subscription-name myrdseventsubscription ^
    --source-identifier mysqldb
```

API

要将源标识符添加到 Amazon RDS 事件通知订阅，请调用 Amazon RDS API `AddSourceIdentifierToSubscription`。包括以下必需参数：

- `SubscriptionName`
- `SourceIdentifier`

从 Amazon RDS 事件通知订阅中删除源标识符

如果您不再希望收到源的事件通知，可以从订阅中删除源标识符 (生成事件的 Amazon RDS 源)。

控制台

您可以使用 Amazon RDS 控制台在修改订阅时通过选择或者取消选择操作轻松地添加或删除源标识符。有关更多信息，请参阅“修改 Amazon RDS 事件通知订阅”。

AWS CLI

要从 Amazon RDS 事件通知订阅删除源标识符，请使用 AWS CLI `remove-source-identifier-from-subscription` 命令。包括以下必需参数：

- `--subscription-name`
- `--source-identifier`

Example

以下示例将从 `mysqldb` 订阅中删除源标识符 `myrdseventsubscription`。

对于 Linux、macOS 或 Unix：

```
aws rds remove-source-identifier-from-subscription \
    --subscription-name myrdseventsubscription \
    --source-identifier mysqldb
```

对于 Windows：

```
aws rds remove-source-identifier-from-subscription ^
    --subscription-name myrdseventsubscription ^
    --source-identifier mysqldb
```

API

要从 Amazon RDS 事件通知订阅删除源标识符，请使用 Amazon RDS API `RemoveSourceIdentifierFromSubscription` 命令。包括以下必需参数：

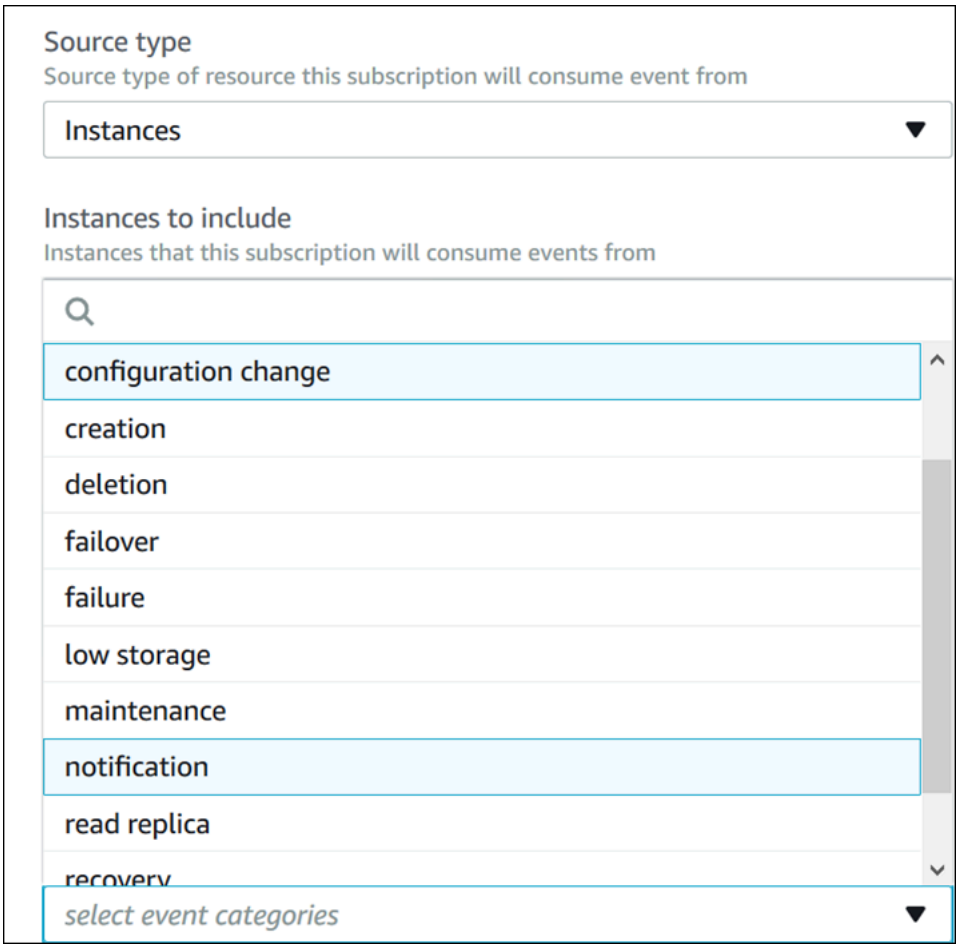
- `SubscriptionName`
- `SourceIdentifier`

列出 Amazon RDS 事件通知类型

资源类型的所有事件都可以分组为类型。要查看可用类型的列表，请使用下列步骤。

控制台

当创建或者修改事件通知订阅时，事件类型会显示在 Amazon RDS 控制台中。有关更多信息，请参阅“修改 Amazon RDS 事件通知订阅”。



AWS CLI

要列出 Amazon RDS 事件通知类别，请使用 AWS CLI `describe-event-categories` 命令。此命令没有必需参数。

Example

```
aws rds describe-event-categories
```

API

要列出 Amazon RDS 事件通知类别，请使用 Amazon RDS API `DescribeEventCategories` 命令。此命令没有必需参数。

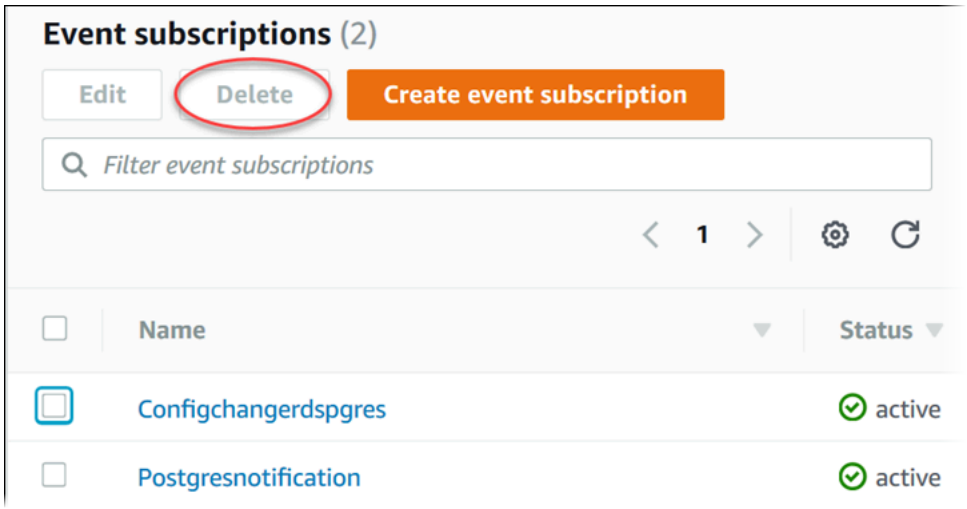
删除 Amazon RDS 事件通知订阅

当您不再需要时，可以删除订阅。该主题的所有用户都将再也不会收到订阅指定的事件通知。

控制台

删除 Amazon RDS 事件通知订阅

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择数据库事件订阅。
3. 在我的数据库事件订阅窗格中，选择您希望删除的订阅。
4. 选择 Delete。
5. Amazon RDS 控制台会表明正在删除订阅。



AWS CLI

要删除 Amazon RDS 事件通知订阅，请使用 AWS CLI `delete-event-subscription` 命令。包括以下必需参数：

- `--subscription-name`

Example

以下示例删除订阅 `myrdssubscription`。

```
aws rds delete-event-subscription --subscription-name myrdssubscription
```

API

要删除 Amazon RDS 事件通知订阅，请使用 RDS API DeleteEventSubscription 命令。包括以下必需参数：

- SubscriptionName

创建对 Amazon Aurora 事件触发的规则

使用 Amazon EventBridge，您可以自动执行 AWS 服务和响应系统事件，例如应用程序可用性问题或资源更改。

主题

- 教程：使用 Amazon EventBridge 记录数据库实例的状态更改

教程：使用 Amazon EventBridge 记录数据库实例的状态更改

在本教程中，您可以创建 AWS Lambda 函数来记录 实例的状态更改。然后，您可以创建一项规则，只要现有 RDS 数据库实例的状态发生更改，便运行该函数。本教程假定您有一个小型的可以暂时关闭的正在运行的测试实例。

> **Important**
>
> 请勿在正在运行的生产数据库实例上执行本教程。

主题

- 步骤 1：创建 AWS Lambda 函数
- 步骤 2：创建规则
- 步骤 3：测试规则

步骤 1：创建 AWS Lambda 函数

创建 Lambda 函数以记录状态更改事件。在创建规则时，您可以指定此函数。

创建 Lambda 函数

1. 打开 AWS Lambda 控制台，地址：https://console.aws.amazon.com/lambda/。
2. 如果您是首次接触 Lambda，您将看到欢迎页面。选择 Get Started Now (立即开始)。否则，选择创建函数。
3. 选择 Author from scratch (从头创作)。
4. 在 Create function (创建函数) 页面上，执行以下操作：

   a. 输入 Lambda 函数的名称和说明。例如，将函数命名为 **`RDSInstanceStateChange`**。

b. 在 Runtime（运行时）中，选择 Node.js 16x。

c. 对于 Architecture（架构），选择 x86_64。

d. 对于 Execution role（执行角色），请执行以下任一操作：

- 选择 Create a new role with basic Lambda permissions (创建具有基本 Lambda 权限的新角色)。
- 对于 Existing role（现有角色），选择 Use an existing role（使用现有角色）。选择要使用的角色。

e. 选择创建函数。

5. 在 RDSInstanceStateChange 页面上，请执行以下操作：

a. 在 Code source (代码源) 中，请选择index.js。

b. 在 index.js 窗格中，删除现有代码。

c. 输入以下代码：

```
console.log('Loading function');

exports.handler = async (event, context) => {
    console.log('Received event:', JSON.stringify(event));
};
```

d. 选择 Deploy (部署)。

步骤 2：创建规则

创建一个规则，以便启动 Lambda 实例时运行 Amazon RDS 函数。

创建 EventBridge 规则

1. 打开位于 https://console.aws.amazon.com/events/ 的 Amazon EventBridge 控制台。
2. 在导航窗格中，选择规则。
3. 选择创建规则。
4. 为规则输入名称和描述。例如，输入 **`RDSInstanceStateChangeRule`**。
5. 选择 Rule with an event pattern（具有事件模式的规则），然后选择 Next（下一步）。
6. 对于事件源，选择AWS 事件或 EventBridge 合作伙伴事件。
7. 向下滚动到 Event pattern（事件模式）部分。

8. 对于事件源，选择 AWS 服务。
9. 对于 AWS 服务，请选择关系数据库服务（RDS）。
10. 对于 Event type (事件类型)，请选择 RDS DB Instance Event (RDS 数据库实例事件)。
11. 保留原定设置事件模式。然后选择下一步。
12. 对于目标类型，选择AWS 服务。
13. 对于 Select a target（选择目标），选择 Lambda function（Lambda 函数）。
14. 对于 Function（函数），选择您创建的 Lambda 函数。然后选择下一步。
15. 在 Configure tags（配置标签）中，选择 Next（下一步）。
16. 查看您的规则中的步骤。然后，选择 Create rule（创建规则）。

步骤 3：测试规则

要测试您的规则，请关闭 RDS 数据库实例。等待几分钟，在实例关闭后，验证您的 Lambda 函数是否已调用。

通过停止一个数据库实例来测试您的规则

1. 通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 停止一个 RDS 数据库实例
3. 打开位于 https://console.aws.amazon.com/events/ 的 Amazon EventBridge 控制台。
4. 在导航窗格中，选择 Rules (规则)，再选择所创建规则的名称。
5. 在规则详细信息中，选择监控。

   随后您将被重定向至 Amazon CloudWatch 控制台。如果您未被重定向，请单击查看 CloudWatch 中的指标。
6. 在 All metrics (所有指标)中，选择所创建规则的名称。

   该图表表明规则已被调用。
7. 在导航窗格中，选择 Log groups (日志组)。
8. 选择 Lambda 函数 (/aws/lambda/***function-name***)的日志组的名称。
9. 选择日志流的名称，以查看您启动的实例的函数提供的数据。您应该会看到接收的事件，类似以下内容：

```
{
    "version": "0",
```

```
    "id": "12a345b6-78c9-01d2-34e5-123f4ghi5j6k",
    "detail-type": "RDS DB Instance Event",
    "source": "aws.rds",
    "account": "111111111111",
    "time": "2021-03-19T19:34:09Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:rds:us-east-1:111111111111:db:testdb"
    ],
    "detail": {
        "EventCategories": [
            "notification"
        ],
        "SourceType": "DB_INSTANCE",
        "SourceArn": "arn:aws:rds:us-east-1:111111111111:db:testdb",
        "Date": "2021-03-19T19:34:09.293Z",
        "Message": "DB instance stopped",
        "SourceIdentifier": "testdb",
        "EventID": "RDS-EVENT-0087"
    }
}
```


有关 JSON 格式的 RDS 事件的更多示例，请参阅Aurora 事件概述。

10. （可选）完成后，您可以打开 Amazon RDS 控制台并启动之前停止的实例。

Aurora 的 Amazon RDS 事件类别和事件消息

Amazon RDS 会在各种类型中生成许多事件，您可以使用 Amazon RDS 控制台、AWS CLI 或 API 订阅这些事件。

主题

- 数据库集群事件
- 数据库实例事件
- 数据库参数组事件
- 数据库安全组事件
- 数据库集群快照事件
- RDS Proxy 事件
- 蓝绿部署事件

数据库集群事件

下表显示了数据库集群为源类型时的事件类别和事件列表。

> **Note**
>
> 没有以数据库集群事件类型存在的 Aurora Serverless 的事件类别。Aurora Serverless 事件范围从 RDS-EVENT-0141 到 RDS-EVENT-0149 不等。

类别	RDS 事件 ID	消息	注意
配置更改	RDS-EVENT-0016	重置主凭证。	
配置更改	RDS-EVENT-0179	在数据库集群上启动数据库活动流。	有关更多信息，请参阅“使用数据库活动流监控 Amazon Aurora”。
配置更改	RDS-EVENT-0180	数据库集群上的数据库活动流已停止。	有关更多信息，请参阅使用数据库活动流监控 Amazon Aurora。

类别	RDS 事件 ID	消息	注意
创建	RDS-EVENT-0170	已创建数据库集群。	
删除	RDS-EVENT-0171	已删除数据库集群。	
故障转移	RDS-EVENT-0069	集群故障转移失败，请检查集群实例的运行状况，然后重试。	
故障转移	RDS-EVENT-0070	再次提升之前的主实例：*name*。	
故障转移	RDS-EVENT-0071	已完成到数据库实例 *name* 的故障转移。	
故障转移	RDS-EVENT-0072	已开始将同一可用区故障转移到数据库实例：*name*。	
故障转移	RDS-EVENT-0073	已开始跨可用区故障转移到数据库实例：*name*。	
失败	RDS-EVENT-0083	Amazon RDS 无法为您的数据库集群 *name* 创建访问您的 Amazon S3 桶的凭证。这是由于在您的账户中未正确配置 S3 快照摄取 IAM 角色或找不到指定的 Amazon S3 桶。有关更多详细信息，请参阅 Amazon RDS 文档中的故障排除部分。	有关更多信息，请参阅使用 Percona XtraBackup 和 Amazon S3 从 MySQL 进行物理迁移 。
失败	RDS-EVENT-0143	由于以下原因，数据库集群未能从 *units* 扩展为 *units*：*reason*。	对 Aurora Serverless 数据库集群的扩展失败。
失败	RDS-EVENT-0354	由于资源不兼容，您无法创建数据库集群。*message*。	*message* 包括有关失败的详细信息。

类别	RDS 事件 ID	消息	注意
失败	RDS-EVENT-0355	由于资源不足限制，无法创建数据库集群。*message*。	*message* 包括有关失败的详细信息。
全局故障转移	RDS-EVENT-0181	已开始全球切换到区域 *name* 中的数据库集群 *name*。	此事件适用于切换操作（以前称为“托管式计划内故障转移”）。 该过程可能会延迟，因为有其他操作正在数据库集群上运行。
全局故障转移	RDS-EVENT-0182	区域 *name* 中的旧主数据库集群 *name* 已成功关闭。	此事件适用于切换操作（以前称为“托管式计划内故障转移”）。 全局数据库中的旧主实例不接受写入操作。所有卷都已同步。
全局故障转移	RDS-EVENT-0183	正在等待全局集群成员间的数据同步。当前滞后于主数据库集群：*reason*。	此事件适用于切换操作（以前称为“托管式计划内故障转移”）。 在全局数据库故障转移的同步阶段出现复制滞后。
全局故障转移	RDS-EVENT-0184	已成功提升区域 *name* 中的新主数据库集群 *name*。	此事件适用于切换操作（以前称为“托管式计划内故障转移”）。 已使用新的主卷重新建立全局数据库的卷拓扑。

类别	RDS 事件 ID	消息	注意
全局故障转移	RDS-EVENT-0185	已完成全球切换到区域 *name* 中的数据库集群 *name*。	此事件适用于切换操作（以前称为“托管式计划内故障转移”）。 主数据库集群上的全球数据库切换已完成。故障转移完成后，副本可能需要很长时间才能上线。
全局故障转移	RDS-EVENT-0186	已取消全球切换到区域 *name* 中的数据库集群 *name*。	此事件适用于切换操作（以前称为“托管式计划内故障转移”）。
全局故障转移	RDS-EVENT-0187	全球切换到区域 *name* 中的数据库集群 *name* 已失败。	此事件适用于切换操作（以前称为“托管式计划内故障转移”）。
全局故障转移	RDS-EVENT-0238	已完成全球故障转移到区域 *name* 中的数据库集群 *name*。	
全局故障转移	RDS-EVENT-0239	全局故障转移到区域 *name* 中的数据库集群 *name* 已失败。	
全局故障转移	RDS-EVENT-0240	全球故障转移后，已开始重新同步区域 *name* 中数据库集群 *name* 的成员。	
全局故障转移	RDS-EVENT-0241	全球故障转移后，已完成重新同步区域 *name* 中数据库集群 *name* 的成员。	
maintenance	RDS-EVENT-0156	数据库集群具有数据库引擎次要版本升级。	

类别	RDS 事件 ID	消息	注意
maintenance	RDS-EVENT-0173	数据库集群引擎版本已升级。	数据库集群的修补已完成。
maintenance	RDS-EVENT-0176	数据库集群引擎主要版本已升级。	
maintenance	RDS-EVENT-0177	数据库集群升级正在进行中。	
maintenance	RDS-EVENT-0286	数据库集群引擎版本升级已开始。	
maintenance	RDS-EVENT-0287	检测到操作系统升级要求。	
maintenance	RDS-EVENT-0288	集群操作系统升级开始。	
maintenance	RDS-EVENT-0289	集群操作系统升级已完成。	
maintenance	RDS-EVENT-0363	正在进行升级准备：*cluster_name*	数据库集群的升级预检查已开始。
notification	RDS-EVENT-0076	从 *name* 迁移到 *name* 失败。原因：*reason*。	迁移到 Aurora 数据库集群失败。
通知	RDS-EVENT-0077	将 *name*.*name* 转换为 InnoDB 失败。原因：*reason*。	在迁移到 Aurora 数据库集群的过程中，将表从源数据库转换为 InnoDB 的尝试失败。
通知	RDS-EVENT-0085	无法升级数据库集群 *name*，因为实例 *name* 的状态为 *name*。解决问题或删除实例，然后重试。	尝试修补 Aurora 数据库集群时出错。检查实例状态、解决问题，然后重试。有关更多信息，请参阅“维护 Amazon Aurora 数据库集群”。

类别	RDS 事件 ID	消息	注意
通知	RDS-EVENT-0141	由于以下原因，将数据库集群从 *units* 扩展为 *units*：*reason*。	已启动对 Aurora Serverless 数据库集群的扩展。
通知	RDS-EVENT-0142	数据库集群已从 *units* 扩展为 *units*。	已完成对 Aurora Serverless 数据库集群的扩展。
通知	RDS-EVENT-0144	正在暂停数据库集群。	已对 Aurora Serverless 数据库集群启动自动暂停。
notification	RDS-EVENT-0145	数据库集群已暂停。	Aurora Serverless 数据库集群已暂停。
notification	RDS-EVENT-0146	已取消对数据库集群的暂停。	已取消对 Aurora Serverless 数据库集群的暂停。
notification	RDS-EVENT-0147	正在恢复数据库集群。	已启动对 Aurora Serverless 数据库集群的恢复操作。
notification	RDS-EVENT-0148	已恢复数据库集群。	已完成对 Aurora Serverless 数据库集群的恢复操作。
notification	RDS-EVENT-0149	数据库集群已从 *units* 扩展到 *units*，但由于以下原因，扩展并不是无缝的：*reason*。	为 Aurora Serverless 数据库集群使用强制选项完成了无缝扩展。连接可能已经根据要求中断。
通知	RDS-EVENT-0150	已停止数据库集群。	
notification	RDS-EVENT-0151	已启动数据库集群。	
notification	RDS-EVENT-0152	数据库集群停止失败。	
notification	RDS-EVENT-0153	数据库集群由于它超过最大允许停止的时间而正在被启动。	

类别	RDS 事件 ID	消息	注意
notification	RDS-EVENT-0172	已将集群从 *name* 重命名为 *name*。	
notification	RDS-EVENT-0234	导出任务失败。	数据库集群导出任务失败。
notification	RDS-EVENT-0235	已取消导出任务。	数据库集群导出任务已取消。
notification	RDS-EVENT-0236	已完成导出任务。	数据库集群导出任务已完成。

数据库实例事件

下表显示的是数据库实例为源类型时的事件类型和事件列表。

类别	RDS 事件 ID	消息	注意
可用性	RDS-EVENT-0004	数据库实例已关闭。	
可用性	RDS-EVENT-0006	已重新启动数据库实例。	
可用性	RDS-EVENT-0022	重新启动 mysql 时出错：*message*。	重新启动 Aurora MySQL 或 RDS for MariaDB 时出现了错误。
回溯	RDS-EVENT-0131	实际回溯时段小于您指定的目标回溯时段。请考虑减少目标回溯时段的小时数。	有关回溯的更多信息，请参阅 回溯 Aurora 数据库集群。
回溯	RDS-EVENT-0132	实际回溯时段与目标回溯时段相同。	
配置更改	RDS-EVENT-0011	已更新为使用 DBParameterGroup *name*。	
配置更改	RDS-EVENT-0012	将修改应用于数据库实例类。	

类别	RDS 事件 ID	消息	注意
配置更改	RDS-EVENT-0014	已完成将修改应用于数据库实例类。	
配置更改	RDS-EVENT-0017	已完成对分配的存储应用修改。	
配置更改	RDS-EVENT-0025	已完成应用修改以转换为多可用区数据库实例。	
配置更改	RDS-EVENT-0029	已完成应用修改以转换为标准（单可用区）数据库实例。	
配置更改	RDS-EVENT-0033	*number* 位用户与主用户名相匹配；仅重置未绑定到特定主机的用户名。	
配置更改	RDS-EVENT-0067	无法重置密码。错误信息：*message*。	
配置更改	RDS-EVENT-0078	监控间隔已更改为 *number*。	增强监控配置进行了更改。
配置更改	RDS-EVENT-0092	已完成数据库参数组更新。	
创建	RDS-EVENT-0005	数据库实例已创建。	
删除	RDS-EVENT-0003	已删除数据库实例。	
失败	RDS-EVENT-0035	数据库实例进入 *state*。*message*。	数据库实例有无效参数。例如，如果数据库实例因为此实例类的内存相关参数设置过高而无法启动，您就应该修改内存参数，并重启数据库实例。

类别	RDS 事件 ID	消息	注意
失败	RDS-EVENT-0036	数据库实例处于 *state*。*message*。	数据库实例处于不兼容的网络中。有些指定的子网 ID 无效或者不存在。
失败	RDS-EVENT-0079	Amazon RDS 无法创建用于增强监控的凭证，此特征已被禁用。这可能是由于您的账户中 rds-monitoring-role 不存在且配置不正确而致。有关更多详细信息，请参阅 Amazon RDS 文档中的故障排除部分。	如果没有增强监控 IAM 角色，则无法启用增强监控。有关创建 IAM 角色的信息，请参阅为 Amazon RDS 增强监控创建 IAM 角色。
失败	RDS-EVENT-0080	Amazon RDS 无法在您的实例 *name* 上配置增强监控，此特征已被禁用。这可能是由于您的账户中 rds-monitoring-role 不存在且配置不正确而致。有关更多详细信息，请参阅 Amazon RDS 文档中的故障排除部分。	增强监控因在配置更改期间出错而处于禁用状态。可能是未正确配置增强监控 IAM 角色。有关创建增强监控 IAM 角色的信息，请参阅为 Amazon RDS 增强监控创建 IAM 角色。
失败	RDS-EVENT-0082	Amazon RDS 无法为您的数据库实例 *name* 创建访问您的 Amazon S3 桶的凭证。这是由于在您的账户中未正确配置 S3 快照摄取 IAM 角色或找不到指定的 Amazon S3 桶。有关更多详细信息，请参阅 Amazon RDS 文档中的故障排除部分。	Aurora 无法从 Amazon S3 存储桶复制备份数据。很可能是用来访问 Amazon S3 存储桶的 Aurora 权限配置不正确。有关更多信息，请参阅使用 Percona XtraBackup 和 Amazon S3 从 MySQL 进行物理迁移 。

类别	RDS 事件 ID	消息	注意
失败	RDS-EVENT-0254	此客户账户的基础存储限额已超过限制。请增加允许的存储限额，以便在实例上进行扩展。	
失败	RDS-EVENT-0353	由于资源不足限制，无法创建数据库实例。*message*。	*message* 包括有关失败的详细信息。
存储不足	RDS-EVENT-0007	分配的存储空间已耗尽。分配更多存储空间以解决问题。	分配的数据库实例存储空间已使用。要解决此问题，请为数据库实例分配额外存储。有关更多信息，请参阅 RDS 常见问题。您可以使用可用存储空间指标监控数据库实例的存储空间。
存储不足	RDS-EVENT-0089	数据库实例 *name* 的可用存储容量低至预调配存储空间的 *percentage* [预调配存储空间：*size*，可用存储空间：*size*]。您可能需要增加预调配存储空间以解决此问题。	数据库实例已使用其分配的存储空间的 90% 以上。您可以使用可用存储空间指标监控数据库实例的存储空间。
存储不足	RDS-EVENT-0227	您的 Aurora 集群的存储空间极其低，只剩下 *amount* 太字节。请采取措施以减少集群上的存储负载。	Aurora 存储子系统的空间不足。
维护	RDS-EVENT-0026	将离线补丁应用到数据库实例。	正在进行数据库实例的脱机维护。数据库实例目前无法使用。
维护	RDS-EVENT-0027	已完成将离线补丁应用到数据库实例。	数据库实例的脱机维护已完成。现在可以使用数据库实例。

类别	RDS 事件 ID	消息	注意
维护	RDS-EVENT-0047	数据库实例已修补。	
maintenance	RDS-EVENT-0155	数据库实例具有数据库引擎次要版本升级。	
maintenance	RDS-EVENT-0178	数据库实例升级正在进行中。	
通知	RDS-EVENT-0044	*message*	这是操作员发出的通知。有关详细信息，请参阅事件消息。
通知	RDS-EVENT-0048	延迟数据库引擎升级，因为此实例有需要先升级的只读副本。	数据库实例的修补已延迟。
通知	RDS-EVENT-0087	数据库实例已停止。	
notification	RDS-EVENT-0088	数据库实例已启动。	
只读副本	RDS-EVENT-0045	复制已停止。	数据库实例上的复制由于存储空间不足已停止。扩展存储空间或减小重做日志的最大大小以让复制继续。要容纳大小为 *amount* MiB 的重做日志，您至少需要 *amount* MiB 的可用存储空间。
只读副本	RDS-EVENT-0046	已恢复只读副本的复制。	此消息会在您首次创建只读副本时出现，或显示为确认复制在正常运行的监控消息。如果此消息在 RDS-EVENT-0045 通知之后出现，则复制已在出现错误之后或停止复制之后恢复。

类别	RDS 事件 ID	消息	注意
只读副本	RDS-EVENT-0057	复制流式传输已终止。	
恢复	RDS-EVENT-0020	已启动数据库实例的还原。恢复时间会随待恢复数据量的变化而变化。	
恢复	RDS-EVENT-0021	数据库实例的恢复已完成。	
恢复	RDS-EVENT-0023	紧急快照请求：*message*。	已请求手动备份，但 Amazon RDS 目前处于创建数据库快照的过程中。请在 Amazon RDS 完成数据库快照后再次提交请求。
恢复	RDS-EVENT-0052	已启动多可用区实例恢复。	恢复时间会随待恢复数据量的变化而变化。
恢复	RDS-EVENT-0053	已完成多可用区实例恢复。等待故障转移或激活。	
恢复	RDS-EVENT-0361	备用数据库实例的恢复已开始。	备用数据库实例在恢复过程中会重新构建。数据库性能在恢复过程中会受到影响。
恢复	RDS-EVENT-0362	备用数据库实例的恢复已完成。	备用数据库实例在恢复过程中会重新构建。数据库性能在恢复过程中会受到影响。
还原	RDS-EVENT-0019	已从数据库实例 *name* 还原为 *name*。	已从时间点备份中恢复数据库实例。
安全修补	RDS-EVENT-0230	系统更新可用于您的数据库实例。有关应用更新的信息，请参阅《RDS 用户指南》中的“维护数据库实例”。	推出了新的操作系统补丁。 一个新的、次要版本的操作系统更新可用于您的数据库实例。有关应用更新的信息，请参阅???。

数据库参数组事件

下表显示的是数据库参数组为源类型时的事件类型和事件列表。

类别	RDS 事件 ID	消息	注意
配置更改	RDS-EVENT-0037	已使用 apply 方法 *method* 将参数 *name* 更新为 *value*。	

数据库安全组事件

下表显示了数据库安全组为源类型时的事件类别和事件列表。

> **Note**
>
> 数据库安全组是 EC2-Classic 的资源。EC2-Classic 已于 2022 年 8 月 15 日停用。如果您尚未从 EC2-Classic 迁移到 VPC，建议您尽快迁移。有关更多信息，请参阅《Amazon EC2 用户指南》中的从 EC2-Classic 迁移到 VPC 和博客 EC2-Classic Networking is Retiring – Here's How to Prepare。

类别	RDS 事件 ID	消息	注意
配置更改	RDS-EVENT-0038	对安全组应用了更改。	
失败	RDS-EVENT-0039	撤消 *user* 的授权。	*user* 拥有的安全组不存在。针对安全组的授权已被吊销，因为它无效。

数据库集群快照事件

下表显示了数据库集群快照为源类型时的事件类别和事件列表。

类别	RDS 事件 ID	消息	注意
备份	RDS-EVENT-0074	正在创建手动集群快照。	

类别	RDS 事件 ID	消息	注意
备份	RDS-EVENT-0075	已创建手动集群快照。	
notification	RDS-EVENT-0162	集群快照导出任务失败。	
notification	RDS-EVENT-0163	集群快照导出任务已取消。	
notification	RDS-EVENT-0164	集群快照导出任务已完成。	
备份	RDS-EVENT-0168	正在创建自动集群快照。	
backup	RDS-EVENT-0169	已创建自动集群快照。	

RDS Proxy 事件

下表显示了 RDS Proxy 为源类型时的事件类别和事件列表。

类别	RDS 事件 ID	消息	注意
配置更改	RDS-EVENT-0204	RDS 修改了数据库代理 *name*。	
配置更改	RDS-EVENT-0207	RDS 修改了数据库代理 *name* 的端点。	
配置更改	RDS-EVENT-0213	RDS 检测到已添加数据库实例并自动将其添加到数据库代理 *name* 的目标组。	
配置更改	RDS-EVENT-0213	RDS 检测到已创建数据库实例 *name*，并自动将其添加到数据库代理 *name* 的目标组 *name*。	
配置更改	RDS-EVENT-0214	RDS 检测到已删除数据库实例 *name* 并自动将其从数据库代理 *name* 的目标组 *name* 中删除。	

类别	RDS 事件 ID	消息	注意
配置更改	RDS-EVENT-0215	RDS 检测到已删除数据库集群 *name* 并自动将其从数据库代理 *name* 的目标组 *name* 中删除。	
创建	RDS-EVENT-0203	RDS 创建了数据库代理 *name*。	
创建	RDS-EVENT-0206	RDS 为数据库代理 *name* 创建了端点 *name*。	
删除	RDS-EVENT-0205	RDS 删除了数据库代理 *name*。	
删除	RDS-EVENT-0208	RDS 删除了数据库代理 *name* 的端点 *name*。	
失败	RDS-EVENT-0243	RDS 无法为代理 *name* 预调配容量，因为您的子网 *name* 中没有足够的 IP 地址可用。要解决此问题，请确保您的子网具有最少的未使用 IP 地址数（如 RDS 代理文档中所建议）。	要确定您的实例类的建议数量，请参阅计划 IP 地址容量。
失败	RDS-EVENT-0275	RDS 限制了一些与数据库代理##的连接。从客户端到代理的并发连接请求数已超出了限制。	

蓝绿部署事件

下表显示了蓝绿部署为源类型时的事件类型和事件列表。

有关蓝绿部署的更多信息，请参阅使用 Amazon Aurora 蓝绿部署进行数据库更新。

类别	Amazon RDS 事件 ID	消息	注意
创建	RDS-EVENT-0244	蓝绿部署任务已完成。您可以对绿色环境数据库进行更多修改或切换部署。	
失败	RDS-EVENT-0245	创建蓝绿部署失败，因为找不到（源/目标）数据库（实例/集群）。	
删除	RDS-EVENT-0246	蓝绿部署已删除。	
notification	RDS-EVENT-0247	已开始从##切换到##。	
notification	RDS-EVENT-0248	蓝绿部署已完成切换。	
失败	RDS-EVENT-0249	蓝绿部署已取消切换。	
notification	RDS-EVENT-0259	已开始从数据库集群##切换为##。	
notification	RDS-EVENT-0260	已完成从数据库集群##切换为##。将##重命名为#### #，将##重命名为##。	
失败	RDS-EVENT-0261	由于##，取消了从数据库集群##到##的切换。	
notification	RDS-EVENT-0311	已启动将数据库集群从##切换为##的序列同步。使用序列时切换可能会导致停机时间延长。	
notification	RDS-EVENT-0312	已完成将数据库集群从##切换为##的序列同步。	

类别	Amazon RDS 事件 ID	消息	注意
失败	RDS-EVENT-0314	由于序列无法同步，因此取消了将数据库集群从##切换到##的序列同步。	

监控 Amazon Aurora 日志文件

每个 RDS 数据库引擎都会生成日志，您可以访问这些日志以进行审计和故障排除。日志的类型取决于数据库引擎。

可以使用 AWS Management Console、AWS Command Line Interface（AWS CLI）或 Amazon RDS API 访问数据库实例的数据库日志。您无法查看、监控或下载事务日志。

> **Note**
>
> 在某些情况下，日志包含隐藏的数据。因此，AWS Management Console 可能会显示日志文件中的内容，但在下载时日志文件可能是空的。

主题

- 查看和列出数据库日志文件
- 下载数据库日志文件
- 监视数据库日志文件
- 将数据库日志发布到 Amazon CloudWatch Logs
- 使用 REST 读取日志文件内容
- Aurora MySQL 数据库日志文件
- Aurora PostgreSQL 数据库日志文件

查看和列出数据库日志文件

您可以使用 AWS Management Console 查看 Amazon Aurora 数据库引擎的数据库日志文件。您可使用 AWS CLI 或 Amazon RDS API 列出可下载或监控的日志文件。

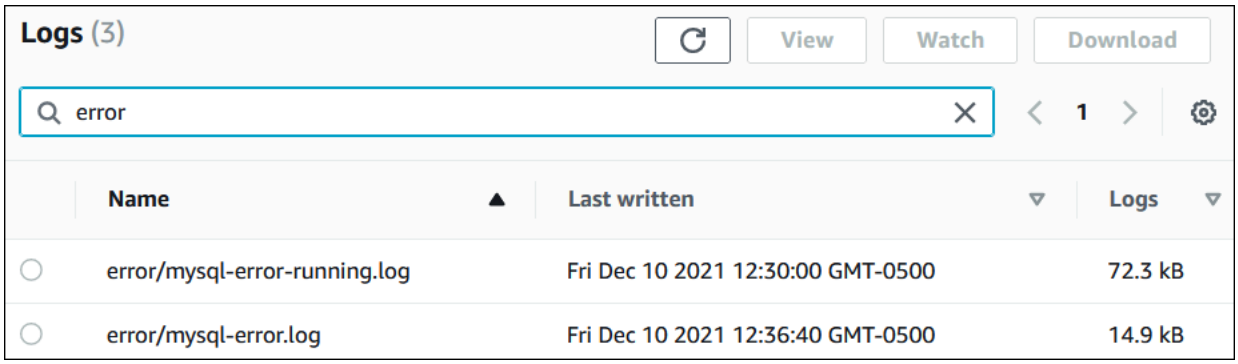
> **Note**
>
> 您无法在 RDS 控制台中查看 Aurora Serverless v1 数据库集群的日志文件。但是，您可以在 Amazon CloudWatch 控制台中查看它们，网址为 https://console.aws.amazon.com/cloudwatch/。

控制台

要查看数据库日志文件，请执行以下操作

1. 通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Databases (数据库)。
3. 选择要查看其日志文件的数据库实例的名称。
4. 选择 Logs & events (日志和事件) 选项卡。
5. 向下滚动到日志部分。
6. （可选）输入搜索词以筛选结果。

   以下示例列出了按文本 **error** 筛选的日志。



7. 选择要查看的日志，然后选择 View（查看）。

AWS CLI

要列出数据库实例的可用数据库日志文件，请使用 AWS CLI `describe-db-log-files` 命令。

以下示例将返回名为 `my-db-instance` 的数据库实例的日志文件列表。

Example

```
aws rds describe-db-log-files --db-instance-identifier my-db-instance
```

RDS API

要列出数据库实例的可用数据库日志文件，请使用 Amazon RDS API `DescribeDBLogFiles` 操作。

下载数据库日志文件

您可使用 AWS Management Console、AWS CLI 或 API 下载数据库日志文件。

控制台

要下载数据库日志文件，请执行以下操作

1. 通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Databases (数据库)。
3. 选择要查看其日志文件的数据库实例的名称。
4. 选择 Logs & events (日志和事件) 选项卡。
5. 向下滚动到日志部分。
6. 在 Logs (日志) 部分中，选择要下载的日志旁边的按钮，然后选择 Download (下载)。
7. 打开提供的链接的上下文 (右键单击) 菜单，然后选择将链接另存为。输入您希望保存日志文件的位置，然后选择保存。



AWS CLI

要下载数据库日志文件，请使用 AWS CLI 命令 `download-db-log-file-portion`。默认情况下，该命令仅下载日志文件的最新部分。但是，您可以通过指定参数 `--starting-token 0` 下载整个文件。

以下示例演示如何下载一个名为 log/ERROR.4 的日志文件的全部内容并将其存储在一个名为 errorlog.txt 的本地文件中。

Example

对于 Linux、macOS 或 Unix：

```
aws rds download-db-log-file-portion \
    --db-instance-identifier myexampledb \
    --starting-token 0 --output text \
    --log-file-name log/ERROR.4 > errorlog.txt
```

对于 Windows：

```
aws rds download-db-log-file-portion ^
    --db-instance-identifier myexampledb ^
    --starting-token 0 --output text ^
    --log-file-name log/ERROR.4 > errorlog.txt
```

RDS API

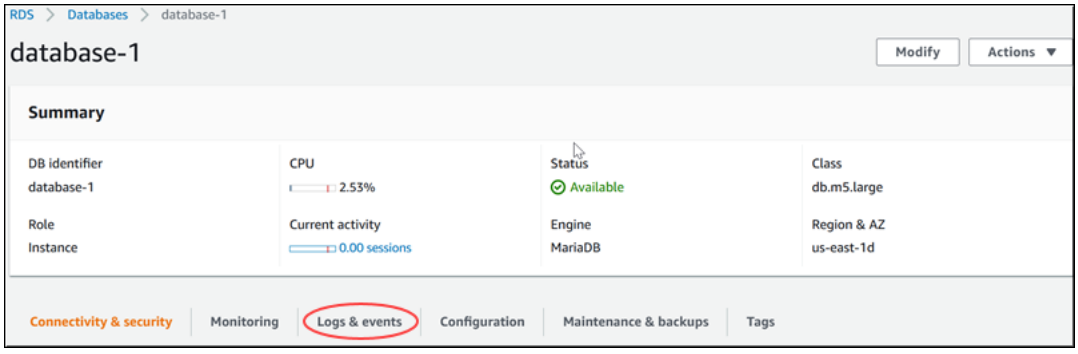
要下载数据库日志文件，请使用 Amazon RDS API DownloadDBLogFilePortion 操作。

监视数据库日志文件

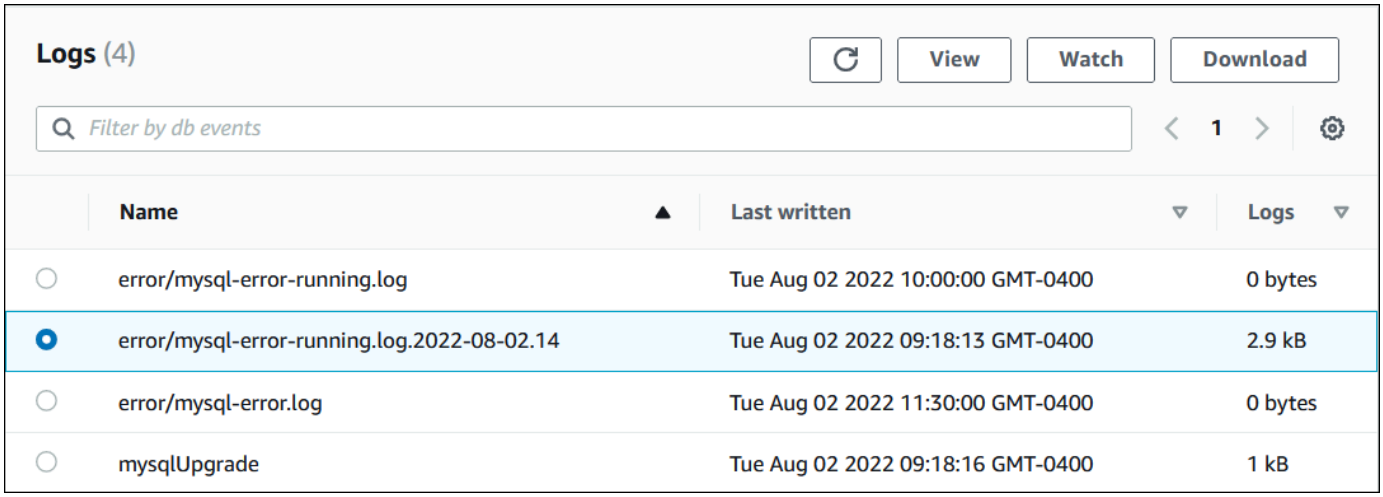
监视数据库日志文件等同于在 UNIX 或 Linux 系统上跟踪该文件。您可使用 AWS Management Console监控日志文件。RDS 每 5 秒刷新一次日志的跟踪。

要监视数据库日志文件，请执行以下操作

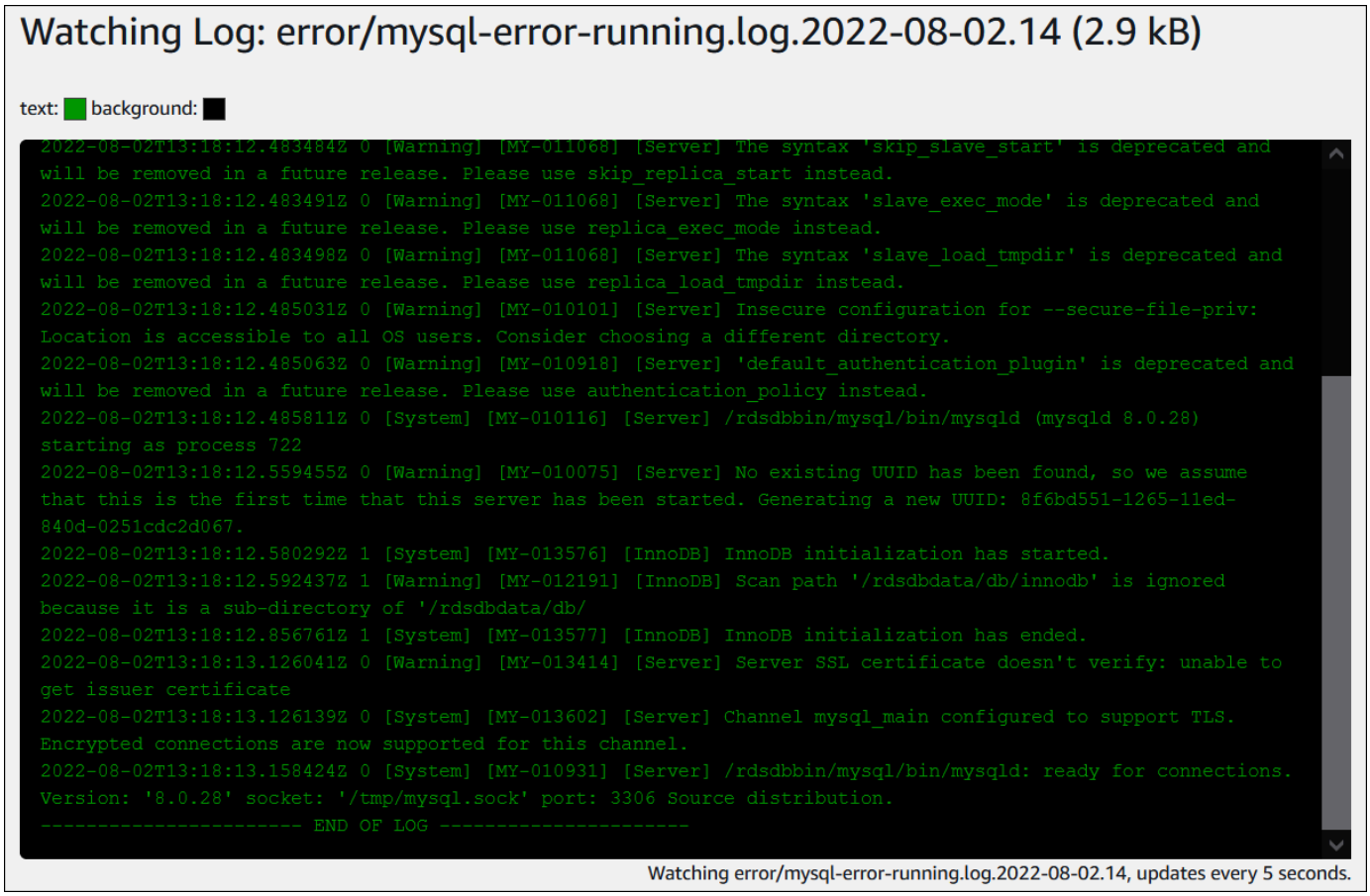
1. 通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Databases (数据库)。
3. 选择要查看其日志文件的数据库实例的名称。
4. 选择 Logs & events (日志和事件) 选项卡。



5. 在 Logs (日志) 部分中，选择一个日志文件，然后选择 Watch (监视)。



RDS 显示日志的跟踪，如以下 MySQL 示例所示。



将数据库日志发布到 Amazon CloudWatch Logs

在本地数据库中，数据库日志驻留在文件系统上。Amazon RDS 不向主机提供对数据库集群的文件系统上的数据库日志的访问权限。出于此原因，Amazon RDS 可让您将数据库日志导出到 Amazon

CloudWatch Logs。利用 CloudWatch Logs，您可以对日志数据执行实时分析。您还可以将数据存储在高持久性存储中，并使用 CloudWatch Logs 代理管理数据。

主题

- RDS 与 CloudWatch Logs 集成概述
- 决定将哪些日志发布到 CloudWatch Logs
- 指定要发布到 CloudWatch Logs 的日志
- 在 CloudWatch Logs 中搜索和筛选您的日志

RDS 与 CloudWatch Logs 集成概述

在 CloudWatch Logs 中，日志流 是共享同一个源的一系列日志事件。CloudWatch Logs 中每个独立的日志源构成一个独立的日志流。日志组是一组具有相同保留期、监控和访问控制设置的日志流。

Amazon Aurora 将您的数据库集群日志记录持续流式传输到日志组。例如，对于您发布的每种类型的日志，您具有日志组 `/aws/rds/cluster/cluster_name/log_type`。此日志组与生成日志的数据库实例位于同一 AWS 区域中。

AWS 无限期地保留已发布到 CloudWatch Logs 的日志数据，除非您指定了保留期。有关更多信息，请参阅更改 CloudWatch Logs 中的日志数据保留。

决定将哪些日志发布到 CloudWatch Logs

每个 RDS 数据库引擎都支持自己的一组日志。要了解数据库引擎的选项，请查看以下主题：

- the section called “将 Aurora MySQL 日志发布到 CloudWatch Logs”
- the section called “将 Aurora PostgreSQL 日志发布到 CloudWatch Logs”

指定要发布到 CloudWatch Logs 的日志

您可以在控制台中指定要发布哪些日志。确保您在 AWS Identity and Access Management（IAM）中具有服务相关角色。有关服务相关角色的更多信息，请参阅将服务相关角色用于 Amazon Aurora。

指定要发布的日志

1. 通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Databases（数据库）。

3. 请执行以下任一操作：

   - 选择创建数据库。
   - 从列表中选择数据库，然后选择 Modify（修改）。

4. 在 Logs exports（日志导出）中，选择要发布哪些日志。

   以下示例指定 Aurora MySQL 数据库集群的审计日志、错误日志、常规日志，实例日志、IAM 数据库身份验证错误日志，以及慢速查询日志。

在 CloudWatch Logs 中搜索和筛选您的日志

您可以使用 CloudWatch Logs 控制台搜索满足指定条件的日志条目。您可以通过 RDS 控制台访问日志，这会引导您进入 CloudWatch Logs 控制台，也可以直接从 CloudWatch Logs 控制台访问日志。

使用 RDS 控制台搜索 RDS 日志

1. 通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Databases（数据库）。
3. 选择数据库集群或数据库实例。
4. 选择配置。
5. 在 Published logs（已发布日志）下，选择要查看的数据库日志。

使用 CloudWatch Logs 控制台搜索 RDS 日志

1. 通过 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Log groups（日志组）。
3. 在筛选框中，输入 **`/aws/rds`**。
4. 对于 Log Groups，选择包含要搜索的日志流的日志组的名称。
5. 对于 Log Streams，选择要搜索的日志流的名称。
6. 在 Log events（日志事件）下，输入要使用的筛选条件语法。

有关更多信息，请参阅《Amazon CloudWatch Logs 用户指南》中的搜索和筛选日志数据。有关介绍如何监控 RDS 日志的博客教程，请参阅使用 Amazon CloudWatch Logs、AWS Lambda 和 Amazon SNS 为 Amazon RDS 构建主动式数据库监控。

使用 REST 读取日志文件内容

Amazon RDS 提供允许访问数据库实例日志文件的 REST 终端节点。如果您需要编写应用程序来流式传输 Amazon RDS 日志文件内容，则这很有用。

语法如下：

```
GET /v13/downloadCompleteLogFile/DBInstanceIdentifier/LogFileName HTTP/1.1
Content-type: application/json
host: rds.region.amazonaws.com
```

以下参数为必需参数：

- *DBInstanceIdentifier* — 包含要下载的日志文件的数据库实例的名称。
- *LogFileName* — 要下载的日志文件的名称。

响应将包含流形式的请求日志文件的内容。

以下示例为 us-west-2 区域中名为 sample-sql 的数据库实例下载名为 log/ERROR.6 的日志文件。

```
GET /v13/downloadCompleteLogFile/sample-sql/log/ERROR.6 HTTP/1.1
host: rds.us-west-2.amazonaws.com
X-Amz-Security-Token: AQoDYXdzEIH//////////
wEa0AIXLhngC5zp9CyB1R6abwKrXHVR5efnAVN3XvR7IwqKYalFSn6UyJuEFTft9nObglx4QJ+GXV9cpACkETq=
X-Amz-Date: 20140903T233749Z
X-Amz-Algorithm: AWS4-HMAC-SHA256
X-Amz-Credential: AKIADQKE4SARGYLE/20140903/us-west-2/rds/aws4_request
X-Amz-SignedHeaders: host
X-Amz-Content-SHA256: e3b0c44298fc1c229afbf4c8996fb92427ae41e4649b934de495991b7852b855
X-Amz-Expires: 86400
X-Amz-Signature: 353a4f14b3f250142d9afc34f9f9948154d46ce7d4ec091d0cdabbcf8b40c558
```

如果您指定的数据库实例不存在，则响应将包含以下错误：

- DBInstanceNotFound — *DBInstanceIdentifier* 不引用现有数据库实例。(HTTP 状态代码: 404)

Aurora MySQL 数据库日志文件

您可直接通过 Amazon RDS 控制台、Amazon RDS API、AWS CLI 或 AWS 开发工具包监控 Aurora MySQL 日志。您还可通过将 MySQL 日志引向主数据库中的数据库表并查询该表，访问这些日志。可使用 mysqlbinlog 实用程序下载二进制日志。

有关查看、下载和监视基于文件的数据库日志的更多信息，请参阅 监控 Amazon Aurora 日志文件。

主题

- Aurora MySQL 数据库日志概览
- 将 Aurora MySQL 日志输出发送到表
- 为单可用区数据库配置 Aurora MySQL 二进制日志记录
- 访问 MySQL 二进制日志

Aurora MySQL 数据库日志概览

您可以监控以下类型的 Aurora MySQL 日志文件：

- 错误日志
- 慢速查询日志
- 常规日志
- 审计日志
- 实例日志
- IAM 数据库身份验证错误日志

默认情况下，系统会生成 Aurora MySQL 错误日志。您可以通过在数据库参数组中设置参数来生成慢速查询日志和一般日志。

主题

- Aurora MySQL 错误日志
- Aurora MySQL 慢速查询日志和一般日志
- Aurora MySQL 审计日志
- Aurora MySQL 实例日志
- Aurora MySQL 的日志轮换和保留
- 将 Aurora MySQL 日志发布到 Amazon CloudWatch Logs

Aurora MySQL 错误日志

Aurora MySQL 在 `mysql-error.log` 文件中写入错误。每个日志文件的名称上都会附有生成时间的信息 (以 UTC 时间标记)。日志文件还会拥有时间戳，帮助您确定日志项的写入时间。

Aurora MySQL 仅在启动、关闭和遇到错误时才向错误日志写入内容。数据库实例可以运行数小时或者数天，而不向错误日志中写入新项。如果看不到最近的条目，则原因是服务器未遇到导致生成日志条目的错误。

根据设计，将筛选错误日志，以便仅显示错误等意外事件。但是，错误日志还包含一些未显示的其他数据库信息，例如查询进度。因此，即使没有任何实际错误，由于持续进行的数据库活动，错误日志的大小也可能会增加。虽然您可能会在 AWS Management Console 中看到错误日志有一定大小（以字节或千字节为单位），但当您下载它们时，它们可能为 0 字节。

Aurora MySQL 每 5 分钟将 `mysql-error.log` 写入磁盘。它会将日志的内容附加到 `mysql-error-running.log` 中。

Aurora MySQL 每小时轮换一次 `mysql-error-running.log` 文件。

> **Note**
>
> Amazon RDS 和 Aurora 之间的日志保留期不同。

Aurora MySQL 慢速查询日志和一般日志

可以将 Aurora MySQL 慢速查询日志和一般日志写入文件或数据库表中。为此，请设置数据库参数组中的参数。有关创建和修改数据库参数组的信息，请参阅 Amazon Aurora 的参数组。您必须先设置这些参数，然后才能在 Amazon RDS 控制台中或使用 Amazon RDS API、Amazon RDS CLI 或AWS软件开发工具包查看慢速查询日志或一般日志。

可通过使用下面列表中的参数来控制 Aurora MySQL 日志记录：

- `slow_query_log`：要创建慢速查询日志，请设置为 1。默认值为 0。
- `general_log`：要创建一般日志，请设置为 1。默认值是 0。
- `long_query_time`：要防止在慢速查询日志中记录快速运行的查询，请指定需要记录的最短查询运行时间值，以秒为单位。默认值为 10 秒；最小值为 0。如果 log_output = FILE，则可以指定精确到微秒的浮点值。如果 log_output = TABLE，则必须指定精确到秒的整数值。系统只记录运行时间超过 `long_query_time` 值的查询。例如，将 `long_query_time` 设置为 0.1 可防止记录任何运行时间少于 100 毫秒的查询。

- log_queries_not_using_indexes：要将所有不使用索引的查询记录到慢速查询日志，请设置为 1。将记录不使用索引的查询，即使它们的运行时间小于 long_query_time 参数的值。默认值是 0。
- log_output *option*：您可为 log_output 参数指定下列选项之一。
  - TABLE — 将一般查询写入 mysql.general_log 表，将慢速查询写入 mysql.slow_log 表。
  - FILE– 将一般查询日志和慢速查询日志写入文件系统。
  - NONE– 禁用日志记录。

  对于 Aurora MySQL 版本 2 和 3，log_output 的默认值为 FILE。

要使慢速查询数据出现在 Amazon CloudWatch Logs 中，必须满足以下条件：

- 必须将 CloudWatch Logs 配置为包含慢速查询日志。
- 必须启用 slow_query_log。
- log_output 必须设置为 FILE。
- 查询花费的时间必须比为 long_query_time 配置的时间长。

有关慢速查询日志和一般日志的更多信息，请参阅 MySQL 文档中的以下主题：

- 慢速查询日志
- 一般查询日志

Aurora MySQL 审计日志

Aurora MySQL 的审计日志记录称为高级审计。要开启高级审计，应设置某些数据库集群参数。有关更多信息，请参阅 在 Amazon Aurora MySQL 数据库集群中使用高级审计。

Aurora MySQL 实例日志

Aurora 为启用了自动暂停的数据库实例创建一个单独的日志文件。此 instance.log 文件记录无法按预期暂停这些数据库实例的所有原因。有关实例日志文件行为和 Aurora 自动暂停功能的更多信息，请参阅监控 Aurora Serverless v2 暂停和恢复活动。

Aurora MySQL 的日志轮换和保留

启用日志记录时，Amazon Aurora 会定期轮换或删除日志文件。这是一种预防措施，用于降低大型日志文件阻止数据库使用或影响性能的可能性。Aurora MySQL 按如下方式处理轮换和删除：

- Aurora MySQL 错误日志文件大小限制为不超过数据库实例的本地存储空间的 15%。为了维护此阈值，日志每小时自动轮换一次。Aurora MySQL 会在 30 天后或达到 15% 的磁盘空间时删除日志。如果在删除旧日志文件后，日志文件的总体大小超出此阈值，则将删除最早的日志文件，直到日志文件大小不再超出此阈值。
- Aurora MySQL 会在 24 小时后或占用 15% 的存储空间后删除审计、一般和慢速查询日志。
- 启用了 FILE 日志记录时，系统会每小时检查一次一般日志和慢速查询日志文件，并删除超过 24 小时的日志文件。在一些情况下，删除之后的剩余日志文件的组合大小可能超过数据库实例的本地空间的 15% 阈值。在这些情况下，将删除最旧的日志文件，直到日志文件大小不再超过此阈值。
- 启用了 TABLE 日志记录时，不会轮换或删除日志表。当所有日志的组合总大小过大时，日志表将被截断。您可以订阅 low storage 事件类别，以便在需要手动轮换或删除日志表以释放空间时收到通知。有关更多信息，请参阅 使用 Amazon RDS 事件通知。

  您可以通过调用 mysql.general_log 过程手动轮换 mysql.rds_rotate_general_log 表。您可以通过调用 mysql.slow_log 过程来轮换 mysql.rds_rotate_slow_log 表。

  当您手动轮换日志表时，当前日志表将复制到备份日志表，随后删除当前日志表中的条目。如果备份日志表已存在，则先将其删除，然后将当前日志表复制到备份。如果需要，您可以查询备份日志表。mysql.general_log 表的备份日志表名为 mysql.general_log_backup。mysql.slow_log 表的备份日志表名为 mysql.slow_log_backup。
- 当文件大小达到 100MB 时，将轮换 Aurora MySQL 审计日志，且 24 小时后将其删除。
- Amazon RDS 会轮换大于 10 MB 的 IAM 数据库身份验证错误日志文件。Amazon RDS 会移除存在时间超过五天或大于 100 MB 的 IAM 数据库身份验证错误日志文件。

要通过 Amazon RDS 控制台、Amazon RDS API、Amazon RDS CLI 或 AWS 开发工具包使用日志，请将 log_output 参数设置为 FILE。就像 Aurora MySQL 错误日志一样，这些日志文件也每小时轮换一次。将保留过去 24 小时内生成的日志文件。请注意，Amazon RDS 和 Aurora 之间的保留期不同。

将 Aurora MySQL 日志发布到 Amazon CloudWatch Logs

您可以配置 Aurora MySQL 数据库集群以将日志数据发布到 Amazon CloudWatch Logs 中的日志组。利用 CloudWatch Logs，可以对日志数据进行实时分析并使用 CloudWatch 创建警报和查看指标。您可以使用 CloudWatch Logs 在高持久性存储中存储日志记录。有关更多信息，请参阅 将 Amazon Aurora MySQL 日志发布到 Amazon CloudWatch Logs。

将 Aurora MySQL 日志输出发送到表

可通过创建数据库参数组并将 log_output 服务器参数设置为 TABLE，将一般日志和慢速查询日志引向数据库实例上的表。系统随后会将一般查询记录到 mysql.general_log 表，并将慢速查询记录到 mysql.slow_log 表。可以查询表来访问日志信息。启用此日志记录功能会增加写入数据库的数据量，导致性能降低。

默认情况下，一般日志和慢速查询日志处于禁用状态。要启用将日志记录到表的功能，必须将 general_log 和 slow_query_log 服务器参数设置为 1。

日志表将不断增大，直至通过将相应的参数重置为 0 来关闭相应的日志记录活动。随着时间的推移，通常会累积大量的数据，这些数据会占用相当大比例的分配存储空间。Amazon Aurora 不允许截断日志表，但可以移动其中的内容。表的交替会将表的内容保存到备份表，然后创建一个新的空日志表。可用以下命令行过程手动轮换日志表，其中 PROMPT> 表示命令提示符：

```
PROMPT> CALL mysql.rds_rotate_slow_log;
PROMPT> CALL mysql.rds_rotate_general_log;
```

要完全移除旧数据并回收磁盘空间，请连续两次调用相应的程序。

为单可用区数据库配置 Aurora MySQL 二进制日志记录

二进制日志是一组日志文件，其中包含对 Aurora MySQL 服务器实例所做的数据修改的信息。二进制日志包含以下信息：

- 描述数据库更改的事件，例如表创建或行修改
- 有关更新数据的各语句的持续时间的信息
- 本应更新但未更新数据的语句的事件

复制过程中发送的二进制日志记录语句。一些恢复操作也需要用到这些语句。有关更多信息，请参阅 MySQL 文档中的二进制日志。

只能从主数据库实例访问二进制日志，而不能从副本访问。

Amazon Aurora 上的 MariaDB 支持基于行、基于语句和混合二进制日志记录格式。我们建议混合使用，除非您需要特定的二进制日志格式。有关不同的 Aurora MySQL 二进制日志格式的详细信息，请参阅 MySQL 文档中的 Binary Logging Formats。

如果您计划使用复制，则二进制日志记录格式很重要，因为它确定了在源中记录和发送到复制目标的数据更改记录。有关用于复制的不同二进制日志记录格式的优缺点的信息，请参阅 MySQL 文档中的基于语句和基于行的复制的优点和缺点。

> **Important**
>
> 在 MySQL 8.0.34 中，MySQL 弃用了 `binlog_format` 参数。在更高版本的 MySQL 中，MySQL 计划删除该参数，并且仅支持基于行的复制。因此，我们建议对新的 MySQL 复制设置使用基于行的日志记录。有关更多信息，请参阅 MySQL 文档中的 binlog_format。
> MySQL 版本 8.0 和 8.4 接受参数 `binlog_format`。使用此参数时，MySQL 会发出弃用警告。在将来的主要版本中，MySQL 将移除参数 `binlog_format`。
> 基于语句的复制可能在源数据库集群和只读副本之间导致不一致。有关详细信息，请参阅 MySQL 文档中的确定二进制日志记录中的安全和不安全语句。
> 启用二进制日志记录会增加数据库集群的写入磁盘 I/O 操作数。您可以使用 `VolumeWriteIOPs` CloudWatch 指标监控 IOPS 使用情况。

设置 MySQL 二进制日志记录格式

1. 通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择参数组。
3. 选择要修改的与数据库集群关联的数据库集群参数组。

   您无法修改默认参数组。如果数据库集群使用默认参数组，则创建新的参数组并将其与数据库集群关联。

   有关参数组的更多信息，请参阅Amazon Aurora 的参数组。
4. 从操作中，选择编辑。
5. 将 `binlog_format` 参数设置为您选择的二进制日志记录格式（`ROW`、`STATEMENT` 或 `MIXED`）。您也可以使用值 `OFF` 关闭二进制日志记录。

   > **Note**
   >
   > 在数据库集群参数组中将 `binlog_format` 设置为 `OFF` 会禁用 `log_bin` 会话变量。这将禁用 Aurora MySQL 数据库集群上的二进制日志记录，这反过来又会将数据库中的 `binlog_format` 会话变量重置为默认值 `ROW`。
6. 选择保存更改以保存对数据库集群参数组的更新。

执行这些步骤后，必须重启数据库集群中的写入器实例，才能应用更改。在 Aurora MySQL 版本 2.09 及更低版本中，当您重启写入器实例时，数据库集群中的所有读取器实例也将重启。在 Aurora MySQL 版本 2.10 及更高版本中，您必须手动重启所有读取器实例。有关更多信息，请参阅 重启 Amazon Aurora 数据库集群或 Amazon Aurora 数据库实例。

> **Important**
>
> 更改数据库集群参数组会影响使用该参数组的所有数据库集群。如果要为AWS区域中的不同 Aurora MySQL 数据库集群指定不同的二进制日志记录格式，数据库集群必须使用不同的数据库集群参数组。这些参数组标识不同的日志记录格式。为每个数据库集群分配相应的数据库集群参数组。有关 Aurora MySQL 参数的更多信息，请参阅 Aurora MySQL 配置参数。

访问 MySQL 二进制日志

可使用 mysqlbinlog 实用程序从 RDS for MySQL 数据库实例下载或流式传输二进制日志。二进制日志下载到本地计算机，可以执行一些操作，例如使用 mysql 实用程序执行重放日志。有关使用 mysqlbinlog 实用程序的更多信息，请参阅 MySQL 文档中的使用 mysqlbinlog 备份二进制日志文件。

要针对 Amazon RDS 实例运行 mysqlbinlog 实用工具，请使用下列选项：

- `--read-from-remote-server` – 必需。
- `--host` – 来自实例的端点的 DNS 名称。
- `--port` – 实例使用的端口。
- `--user` – 已获得 `REPLICATION SLAVE` 权限的 MySQL 用户。
- `--password` – MySQL 用户的密码，或忽略密码值以让实用程序提示您输入密码。
- `--raw` – 以二进制格式下载文件。
- `--result-file` – 用于接收原始输出的本地文件。
- `--stop-never` – 流式传输二进制日志文件。
- `--verbose` – 使用 `ROW` 二进制日志格式时，包括此选项以将行事件视为伪 SQL 语句。有关 `--verbose` 选项的更多信息，请参阅 MySQL 文档中的 mysqlbinlog 行事件显示。
- 指定一个或多个二进制日志文件的名称。要获取可用日志的列表，请使用 SQL 命令 `SHOW BINARY LOGS`。

有关 mysqlbinlog 选项的更多信息，请参阅 MySQL 文档中的 mysqlbinlog - 处理二进制日志文件的实用程序。

以下示例显示如何使用 mysqlbinlog 实用程序。

对于 Linux、macOS 或 Unix：

```
mysqlbinlog \
    --read-from-remote-server \
    --host=MySQLInstance1.cg034hpkmmjt.region.rds.amazonaws.com \
    --port=3306  \
    --user ReplUser \
    --password \
    --raw \
    --verbose \
    --result-file=/tmp/ \
    binlog.00098
```

对于 Windows：

```
mysqlbinlog ^
    --read-from-remote-server ^
    --host=MySQLInstance1.cg034hpkmmjt.region.rds.amazonaws.com ^
    --port=3306  ^
    --user ReplUser ^
    --password ^
    --raw ^
    --verbose ^
    --result-file=/tmp/ ^
    binlog.00098
```

数据库实例上的二进制日志必须保持可用，以备 mysqlbinlog 实用程序访问。要确保这些日志的可用性，请使用 mysql.rds_set_configuration 存储过程并指定一个时段，以便您有足够的时间来下载日志。如果未设置此配置，Amazon RDS 会尽可能快地清除二进制日志，导致 mysqlbinlog 实用程序检索的二进制日志出现空白。

以下示例将保留期设置为 1 天。

```
call mysql.rds_set_configuration('binlog retention hours', 24);
```

要显示当前设置，请使用 mysql.rds_show_configuration 存储过程。

```
call mysql.rds_show_configuration;
```

Aurora PostgreSQL 数据库日志文件

您可以监控以下类型的 Aurora PostgreSQL 日志文件：

- PostgreSQL 日志
- 实例日志
- IAM 数据库身份验证错误日志

> **Note**
>
> 要启用 IAM 数据库身份验证错误日志，您必须先为 Aurora PostgreSQL 数据库集群启用 IAM 数据库身份验证。有关启用 IAM 数据库身份验证的更多信息，请参阅启用和禁用 IAM 数据库身份验证。

Aurora PostgreSQL 将数据库活动记录到原定设置 PostgreSQL 日志文件中。对于本地 PostgreSQL 数据库实例，这些消息本地存储在 `log/postgresql.log` 中。对于 Aurora PostgreSQL 数据库集群 ，日志文件可用于 Aurora 集群上。也可以通过 AWS Management Console访问这些日志，您可以在此处查看或下载它们。原定设置日志记录级别可捕获登录失败、严重服务器错误、死锁和查询失败。

有关如何查看、下载和监视基于文件的数据库日志的更多信息，请参阅 监控 Amazon Aurora 日志文件。要了解有关 PostgreSQL 日志的更多信息，请参阅 Working with Amazon RDS and Aurora PostgreSQL logs: Part 1 和 Working with Amazon RDS and Aurora PostgreSQL logs: Part 2。

除了本主题中讨论的标准 PostgreSQL 日志外，Aurora PostgreSQL 还支持 PostgreSQL Audit 扩展（`pgAudit`）。大多数受监管行业和政府机构需要保留对数据所做更改的审计日志或审计跟踪，以符合法律要求。有关安装和使用 pgAudit 的更多信息，请参阅 使用 pgAudit 记录数据库活动。

Aurora 为启用了自动暂停的数据库实例创建一个单独的日志文件。此 instance.log 文件记录无法按预期暂停这些数据库实例的所有原因。有关实例日志文件行为和 Aurora 自动暂停功能的更多信息，请参阅监控 Aurora Serverless v2 暂停和恢复活动。

主题

- Aurora PostgreSQL 中的日志记录参数
- 为您的 Aurora PostgreSQL 数据库集群开启查询日志记录

Aurora PostgreSQL 中的日志记录参数

您可以通过修改各种参数来自定义 Aurora PostgreSQL 数据库集群 的日志记录行为。在下表中，您可以找到影响日志存储时间长度、何时轮换日志和是否以 CSV（逗号分隔值）格式输出日志的参数。您还可以找到发送到 STDERR 的文本输出以及其他设置。要更改可修改的参数的设置，请为以下项目使用自定义数据库集群参数组：Aurora PostgreSQL 数据库集群 。有关更多信息，请参阅Amazon Aurora 的参数组。

参数	默认值	描述
log_destination	stderr	设置日志的输出格式。原定设置为 `stderr`，但您也可以通过向设置中添加 `csvlog` 来指定逗号分隔值（CSV）。有关更多信息，请参阅 设置日志目标（stderr、csvlog）。
log_filename	postgresql.log.%Y-%m-%d-%H%M	指定日志文件名的模式。除原定设置外，此参数还支持使用 `postgresql.log.%Y-%m-%d` 和 `postgresql.log.%Y-%m-%d-%H` 作为文件名模式。对于 Aurora PostgreSQL 版本 17.4 及更高版本，您无法修改此参数。
log_line_prefix	%t:%r:%u@%d:[%p]:	为写入 `stderr` 的每个日志行定义前缀，以记录时间（%t）、远程主机（%r）、用户（%u）、数据库（%d）和进程 ID（%p）。
log_rotation_age	60	此分钟数后，将自动轮换日志文件。您可以在 1 到 1440 分钟的范围内更改此值。有关更多信息，请参阅设置日志文件轮换。
log_rotation_size	–	日志自动轮换的大小（kB）。您可以在 50000 到 1000000 千字节的范围内更改此值。要了解更多信息，请参阅设置日志文件轮换。
rds.log_retention_period	4320	超过指定分钟数的 PostgreSQL 日志将被删除。默认值为 4320 分钟，表示系统将在 3 天后删除日志文件。有关更多信息，请参阅 设置日志保留期。

要识别应用程序问题，您可以在日志中查找查询失败、登录失败、死锁和致命的服务器错误。例如，假设您将旧版应用程序从 Oracle 转换为 Aurora PostgreSQL，但部分查询可能未正确转换。这些格式不正确的查询会生成错误消息，您可以在日志中找到这些错误消息以帮助识别问题。有关日志记录查询的更多信息，请参阅 为您的 Aurora PostgreSQL 数据库集群开启查询日志记录。

在以下主题中，您可以找到有关如何设置各种参数（用于控制 PostgreSQL 日志的基本详细信息）的信息。

主题

- 设置日志保留期
- 设置日志文件轮换
- 设置日志目标（stderr、csvlog）
- 了解 log_line_prefix 参数

设置日志保留期

`rds.log_retention_period` 参数指定 Aurora PostgreSQL 数据库集群将其日志文件保留多长时间。原定设置为 3 天（4320 分钟），但您可以将此值设置为 1 天（1440 分钟）到 7 天（10080 分钟）之间的任意值。确保您的 Aurora PostgreSQL 数据库集群具有足够的存储空间，可将日志文件保留一段时间。

建议您将日志定期发布到 Amazon CloudWatch Logs，以便您可以在日志从 Aurora PostgreSQL 数据库集群 中删除很久之后查看和分析系统数据。有关更多信息，请参阅 将 Aurora PostgreSQL 日志发布到 Amazon CloudWatch Logs。设置 CloudWatch 发布后，Aurora 在日志发布到 CloudWatch Logs 后才会删除日志。

在数据库实例的存储达到阈值时，Amazon Aurora 会压缩较旧的 PostgreSQL 日志。Aurora 使用 gzip 压缩实用程序来压缩文件。有关更多信息，请参阅 gzip 网站。

当数据库实例的存储空间不足且所有可用日志都被压缩时，您会收到如下所示的警告：

```
Warning: local storage for PostgreSQL log files is critically low for
this Aurora PostgreSQL instance, and could lead to a database outage.
```

如果存储空间不足，Aurora 可能会在指定的保留期结束前删除压缩的 PostgreSQL 日志。如果发生这种情况，您将看到类似如下的消息：

```
The oldest PostgreSQL log files were deleted due to local storage constraints.
```

设置日志文件轮换

原定设置情况下，Aurora 每小时都会创建新的日志文件。计时由 log_rotation_age 参数控制。此参数的原定设置值为 60（分钟），但您可以将其设置为 1 分钟到 24 小时（1440 分钟）之间的任意值。轮换时，系统将创建一个新的不同日志文件。该文件根据 log_filename 参数指定的模式命名。

也可以根据日志文件的大小进行轮换（如 log_rotation_size 参数指定）。此参数指定在日志达到指定的大小（以千字节为单位）时应轮换日志。对于 Aurora PostgreSQL 数据库集群，原定设置 log_rotation_size 为 100000KB（千字节），但您可以将此值设置为 50000 到 1000000 KB 之间的任意值。

日志文件名基于在 log_filename 参数中指定的文件名模式。此参数的可用设置如下：

- postgresql.log.%Y-%m-%d – 日志文件名的原定设置格式。在日志文件的名称中包括年、月和日。
- postgresql.log.%Y-%m-%d-%H – 在日志文件名格式中包含小时。
- postgresql.log.%Y-%m-%d-%H%M – 在日志文件名格式中包含小时:分钟。

如果您将 log_rotation_age 参数设置为小于 60 分钟，则将 log_filename 参数设置为分钟格式。

有关更多信息，请参阅 PostgreSQL 文档中的 log_rotation_age 和 log_rotation_size。

设置日志目标（**stderr**、**csvlog**）

默认情况下，Aurora PostgreSQL 以标准错误 (stderr) 格式生成日志。此格式是 log_destination 参数的原定设置。每条消息都使用在 log_line_prefix 参数中指定的模式作为前缀。有关更多信息，请参阅 了解 log_line_prefix 参数。

Aurora PostgreSQL 也可以按 csvlog 格式生成日志。csvlog 用于将日志数据作为逗号分隔值（CSV）数据进行分析。例如，假设您使用 log_fdw 扩展将日志作为外部表处理。在 stderr 日志文件上创建的外部表包含一个列，其中包含日志事件数据。通过向 csvlog 参数添加 log_destination，可以获得 CSV 格式的日志文件，其中对外部表的多个列进行了分界。您现在可以更轻松地对日志进行排序和分析。

如果您为此参数指定 csvlog，请注意将同时生成 stderr 和 csvlog 文件。确保监控日志所占用的存储空间，同时考虑 rds.log_retention_period 以及影响日志存储和周转的其他设置。使用 stderr 和 csvlog 会使日志占用的存储空间增加一倍以上。

如果将 csvlog 添加到 log_destination 并想单独恢复到 stderr，则需要重置参数。为此，请打开 Amazon RDS 控制台，然后为您的实例打开自定义数据库集群参数组。选择 log_destination 参数，选择 Edit parameter（编辑参数），然后选择 Reset（重置）。

有关配置日志记录的更多信息，请参阅使用 Amazon RDS 和 Aurora PostgreSQL 日志：第 1 部分。

了解 log_line_prefix 参数

stderr 日志格式将由 log_line_prefix 参数指定的详细信息作为每条日志消息的前缀。默认值为：

```
%t:%r:%u@%d:[%p]:t
```

从 Aurora PostgreSQL 版本 16 开始，还可以选择：

```
%m:%r:%u@%d:[%p]:%l:%e:%s:%v:%x:%c:%q%a
```

发送到 stderr 的每个日志条目都包含以下基于所选值的信息：

- %t：日志条目的时间，不带毫秒
- %m：日志条目的时间，带毫秒
- %r – 远程主机地址
- %u@%d – 用户名 @ 数据库名称
- [%p] – 进程 ID（如果有）
- %l：每个会话的日志行号
- %e：SQL 错误代码
- %s：进程开始时间戳
- %v：虚拟事务 ID
- %x：事务 ID
- %c：会话 ID
- %q：非会话终止器
- %a：应用程序名称

为您的 Aurora PostgreSQL 数据库集群开启查询日志记录

通过设置下表中列出的一些参数，您可以收集有关数据库活动的更多详细信息，包括查询、等待锁定的查询、检查点和许多其他详细信息。本主题重点介绍日志记录查询。

参数	默认值	描述
log_connections	–	记录每个成功的连接。要了解如何将此参数与 `log_disconnections` 结合使用来检测连接流失，请参阅 使用池管理 Aurora PostgreSQL 连接流失 。
log_disconnections	–	记录每个会话的结束及其持续时间。要了解如何将此参数与 `log_connections` 结合使用来检测连接流失，请参阅 使用池管理 Aurora PostgreSQL 连接流失 。
log_checkpoints	–	不适用于 Aurora PostgreSQL
log_lock_waits	–	记录长锁定等待次数。默认情况下，不设置此参数。
log_min_duration_sample	–	设置如超出则记录语句示例的最短执行时间（ms）。示例数量使用 `log_statement_sample_rate` 参数进行设置。
log_min_duration_statement	–	任何至少运行指定时间或更长时间的 SQL 语句都会被记录下来。默认情况下，不设置此参数。开启该参数可帮助查找未优化的查询。
log_statement	–	设置所记录的语句类型。默认情况下，未设置此参数，但您可以将其更改为 `all`、`ddl` 或 `mod`，以指定要记录的 SQL 语句的类型。如果您为此参数指定了与 `none` 不同的任何值，还应采取其他措施来防止日志文件中的密码泄露。有关更多信息，请参阅 降低使用查询日志记录时泄露密码的风险。

参数	默认值	描述
log_statement_sample_rate	–	超过在 log_min_duration_sample 中指定的时间的语句百分比，以介于 0.0 和 1.0 之间的浮点值表示。
log_statement_stats	–	向服务器日志写入累计性能统计数据。

使用日志记录查找执行缓慢的查询

您可以记录 SQL 语句和查询，以帮助查找执行缓慢的查询。您可以通过修改本节中概述的 log_statement 和 log_min_duration 参数的设置来开启此功能。在为 Aurora PostgreSQL 数据库集群 开启查询日志记录之前，您应该了解日志中可能存在的密码泄露以及如何降低风险。有关更多信息，请参阅 降低使用查询日志记录时泄露密码的风险。

接下来，您可以了解有关 log_statement 和 log_min_duration 参数的参考信息。

log_statement

此参数指定应发送到日志的 SQL 语句的类型。默认值为 none。如果您将此参数更改为 all、ddl 或 mod，请务必采取建议的操作来降低在日志中泄露密码的风险。有关更多信息，请参阅 降低使用查询日志记录时泄露密码的风险。

全部

记录所有语句。建议将此设置用于调试目的。

ddl

记录所有数据定义语言（DDL）语句，例如 CREATE、ALTER、DROP 等。

mod

记录所有可修改数据的 DDL 语句和数据操作语言（DML）语句，例如 INSERT、UPDATE 和 DELETE。

none

不记录任何 SQL 语句。我们建议使用此设置，以避免在日志中泄露密码的风险。

log_min_duration_statement

任何至少运行指定时间或更长时间的 SQL 语句都会被记录下来。默认情况下，不设置此参数。开启该参数可帮助查找未优化的查询。

–1–2147483647

如超过即记录语句的运行时间的毫秒数（ms）。

设置查询日志记录

这些步骤假设您的 Aurora PostgreSQL 数据库集群使用自定义数据库集群参数组。

1. 将 log_statement 参数设置为 all。以下示例显示了使用此参数设置写入 postgresql.log 文件的信息。

```
2022-10-05 22:05:52 UTC:52.95.4.1(11335):postgres@labdb:[3639]:LOG: statement:
 SELECT feedback, s.sentiment,s.confidence
FROM support,aws_comprehend.detect_sentiment(feedback, 'en') s
ORDER BY s.confidence DESC;
2022-10-05 22:05:52 UTC:52.95.4.1(11335):postgres@labdb:[3639]:LOG: QUERY
 STATISTICS
2022-10-05 22:05:52 UTC:52.95.4.1(11335):postgres@labdb:[3639]:DETAIL: ! system
 usage stats:
! 0.017355 s user, 0.000000 s system, 0.168593 s elapsed
! [0.025146 s user, 0.000000 s system total]
! 36644 kB max resident size
! 0/8 [0/8] filesystem blocks in/out
! 0/733 [0/1364] page faults/reclaims, 0 [0] swaps
! 0 [0] signals rcvd, 0/0 [0/0] messages rcvd/sent
! 19/0 [27/0] voluntary/involuntary context switches
2022-10-05 22:05:52 UTC:52.95.4.1(11335):postgres@labdb:[3639]:STATEMENT: SELECT
 feedback, s.sentiment,s.confidence
FROM support,aws_comprehend.detect_sentiment(feedback, 'en') s
ORDER BY s.confidence DESC;
2022-10-05 22:05:56 UTC:52.95.4.1(11335):postgres@labdb:[3639]:ERROR: syntax error
 at or near "ORDER" at character 1
2022-10-05 22:05:56 UTC:52.95.4.1(11335):postgres@labdb:[3639]:STATEMENT: ORDER BY
 s.confidence DESC;
---------------------- END OF LOG ----------------------
```

2. 设置 log_min_duration_statement 参数。以下示例显示了参数设置为 postgresql.log 时写入 1 文件的信息。

将记录超过在 log_min_duration_statement 参数中指定的持续时间的查询。下面是一个示例。您可以在 Amazon RDS 控制台中查看 Aurora PostgreSQL 数据库集群的日志文件。

```
2022-10-05 19:05:19 UTC:52.95.4.1(6461):postgres@labdb:[6144]:LOG: statement: DROP
 table comments;
2022-10-05 19:05:19 UTC:52.95.4.1(6461):postgres@labdb:[6144]:LOG: duration:
 167.754 ms
2022-10-05 19:08:07 UTC::@:[355]:LOG: checkpoint starting: time
2022-10-05 19:08:08 UTC::@:[355]:LOG: checkpoint complete: wrote 11 buffers
 (0.0%); 0 WAL file(s) added, 0 removed, 0 recycled; write=1.013 s, sync=0.006 s,
 total=1.033 s; sync files=8, longest=0.004 s, average=0.001 s; distance=131028 kB,
 estimate=131028 kB
---------------------- END OF LOG ----------------------
```

降低使用查询日志记录时泄露密码的风险

我们建议您将 log_statement 保持设置为 none，以避免泄露密码。如果您将 log_statement 设置为 all、ddl 或 mod，我们建议您采取以下一个或多个步骤。

- 对于客户端，加密敏感信息。有关更多信息，请参阅 PostgreSQL 文档中的加密选项。使用 ENCRYPTED 和 UNENCRYPTED 语句的 CREATE（和 ALTER）选项。有关更多信息，请参阅 PostgreSQL 文档中的 CREATE USER。
- 对于您的 Aurora PostgreSQL 数据库集群，请设置并使用 PostgreSQL Auditing（pgAudit）扩展。此扩展编辑发送到日志的 CREATE 和 ALTER 语句中的敏感信息。有关更多信息，请参阅 使用 pgAudit 记录数据库活动。
- 限制对 CloudWatch Logs 的访问。
- 使用更强的身份验证机制，如 IAM。

监控 AWS CloudTrail 中的 Amazon Aurora API 调用

AWS CloudTrail 是一项可帮助您审计 AWS 账户的 AWS 服务。在您创建 AWS CloudTrail 账户时，对您的账户开启 AWS。有关 CloudTrail 的更多信息，请参阅《AWS CloudTrail 用户指南》https://docs.aws.amazon.com/awscloudtrail/latest/userguide/。

主题

- CloudTrail 与 Amazon Aurora 集成
- Amazon Aurora 日志文件条目

CloudTrail 与 Amazon Aurora 集成

所有 Amazon Aurora 操作均由 CloudTrail 记录。CloudTrail 提供了用户、角色或 AWS 服务在 Amazon Aurora 中所执行操作的记录。

CloudTrail 事件

CloudTrail 将 Amazon Aurora 的 API 调用作为事件捕获。一个事件表示来自任何源的一个请求，包括有关所请求的操作、操作的日期和时间、请求参数等方面的信息。事件包含来自 Amazon RDS 控制台的调用和对 Amazon RDS API 操作的代码调用。

Amazon Aurora 活动记录在 Event history（事件历史记录）中的 CloudTrail 事件中。您可以使用 CloudTrail 控制台查看AWS区域中过去 90 天内记录的 API 活动和事件。有关更多信息，请参阅使用 CloudTrail 事件历史记录查看事件。

CloudTrail 跟踪

要持续记录 AWS 账户中的事件（包括 Amazon Aurora 的事件），请创建跟踪记录。跟踪是一种配置，可用于将事件传送到指定的 Amazon S3 存储桶。CloudTrail 通常会在账户活动发生后的 15 分钟内传送日志文件。

> **Note**
>
> 如果您不配置跟踪，则仍可在 CloudTrail 控制台中的事件历史记录中查看最新事件。

您可以为 AWS 账户创建两种类型的跟踪：应用于所有区域的跟踪，或应用于一个区域的跟踪。预设情况下，在控制台中创建跟踪时，此跟踪应用于所有 区域。

此外，您可以配置其他 AWS 服务，进一步分析在 CloudTrail 日志中收集的事件数据并采取行动。有关更多信息，请参阅：

- 创建跟踪概览
- CloudTrail 支持的服务和集成
- 为 CloudTrail 配置 Amazon SNS 通知
- 从多个区域接收 CloudTrail 日志文件和从多个账户接收 CloudTrail 日志文件

Amazon Aurora 日志文件条目

CloudTrail 日志文件包含一个或多个日志条目。CloudTrail 日志文件不是公用 API 调用的有序堆栈跟踪，因此它们不会按任何特定顺序显示。

下面的示例显示了一个 CloudTrail 日志条目，该条目说明了 `CreateDBInstance` 操作。

```
{
    "eventVersion": "1.04",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AKIAIOSFODNN7EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/johndoe",
        "accountId": "123456789012",
        "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
        "userName": "johndoe"
    },
    "eventTime": "2018-07-30T22:14:06Z",
    "eventSource": "rds.amazonaws.com",
    "eventName": "CreateDBInstance",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "aws-cli/1.15.42 Python/3.6.1 Darwin/17.7.0 botocore/1.10.42",
    "requestParameters": {
        "enableCloudwatchLogsExports": [
            "audit",
            "error",
            "general",
            "slowquery"
        ],
        "dBInstanceIdentifier": "test-instance",
        "engine": "mysql",
```

```
        "masterUsername": "myawsuser",
        "allocatedStorage": 20,
        "dBInstanceClass": "db.m1.small",
        "masterUserPassword": "****"
    },
    "responseElements": {
        "dBInstanceArn": "arn:aws:rds:us-east-1:123456789012:db:test-instance",
        "storageEncrypted": false,
        "preferredBackupWindow": "10:27-10:57",
        "preferredMaintenanceWindow": "sat:05:47-sat:06:17",
        "backupRetentionPeriod": 1,
        "allocatedStorage": 20,
        "storageType": "standard",
        "engineVersion": "8.0.28",
        "dbInstancePort": 0,
        "optionGroupMemberships": [
            {
                "status": "in-sync",
                "optionGroupName": "default:mysql-8-0"
            }
        ],
        "dBParameterGroups": [
            {
                "dBParameterGroupName": "default.mysql8.0",
                "parameterApplyStatus": "in-sync"
            }
        ],
        "monitoringInterval": 0,
        "dBInstanceClass": "db.m1.small",
        "readReplicaDBInstanceIdentifiers": [],
        "dBSubnetGroup": {
            "dBSubnetGroupName": "default",
            "dBSubnetGroupDescription": "default",
            "subnets": [
                {
                    "subnetAvailabilityZone": {"name": "us-east-1b"},
                    "subnetIdentifier": "subnet-cbfff283",
                    "subnetStatus": "Active"
                },
                {
                    "subnetAvailabilityZone": {"name": "us-east-1e"},
                    "subnetIdentifier": "subnet-d7c825e8",
                    "subnetStatus": "Active"
                },
```

```
                {
                    "subnetAvailabilityZone": {"name": "us-east-1f"},
                    "subnetIdentifier": "subnet-6746046b",
                    "subnetStatus": "Active"
                },
                {
                    "subnetAvailabilityZone": {"name": "us-east-1c"},
                    "subnetIdentifier": "subnet-bac383e0",
                    "subnetStatus": "Active"
                },
                {
                    "subnetAvailabilityZone": {"name": "us-east-1d"},
                    "subnetIdentifier": "subnet-42599426",
                    "subnetStatus": "Active"
                },
                {
                    "subnetAvailabilityZone": {"name": "us-east-1a"},
                    "subnetIdentifier": "subnet-da327bf6",
                    "subnetStatus": "Active"
                }
            ],
            "vpcId": "vpc-136a4c6a",
            "subnetGroupStatus": "Complete"
        },
        "masterUsername": "myawsuser",
        "multiAZ": false,
        "autoMinorVersionUpgrade": true,
        "engine": "mysql",
        "cACertificateIdentifier": "rds-ca-2015",
        "dbiResourceId": "db-ETDZIIXHEWY5N7GXVC4SH7H5IA",
        "dBSecurityGroups": [],
        "pendingModifiedValues": {
            "masterUserPassword": "****",
            "pendingCloudwatchLogsExports": {
                "logTypesToEnable": [
                    "audit",
                    "error",
                    "general",
                    "slowquery"
                ]
            }
        },
        "dBInstanceStatus": "creating",
        "publiclyAccessible": true,
```

```
        "domainMemberships": [],
        "copyTagsToSnapshot": false,
        "dBInstanceIdentifier": "test-instance",
        "licenseModel": "general-public-license",
        "iAMDatabaseAuthenticationEnabled": false,
        "performanceInsightsEnabled": false,
        "vpcSecurityGroups": [
            {
                "status": "active",
                "vpcSecurityGroupId": "sg-f839b688"
            }
        ]
    },
    "requestID": "daf2e3f5-96a3-4df7-a026-863f96db793e",
    "eventID": "797163d3-5726-441d-80a7-6eeb7464acd4",
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```

如前面示例中的 `userIdentity` 元素所示，每个事件或日志条目都包含有关生成请求的人员的信息。身份信息可帮助您确定以下内容：

- 请求是使用根用户凭证还是 IAM 用户凭证发出的。
- 请求是使用角色还是联合用户的临时安全凭证发出的。
- 请求是否由其他 AWS 服务发出。

有关 `userIdentity` 的更多信息，请参阅 CloudTrail userIdentity 元素。有关 `CreateDBInstance` 和其他 Amazon Aurora 操作的更多信息，请参阅 Amazon RDS API 参考。

使用数据库活动流监控 Amazon Aurora

通过使用数据库活动流，您可以监控近乎实时的数据库活动流。

主题

- 数据库活动流概览
- Aurora MySQL 数据库活动流的网络先决条件
- 启动数据库活动流
- 获取数据库活动流的状态
- 停止数据库活动流
- 监控数据库活动流
- 数据库活动流的 IAM 策略示例

数据库活动流概览

作为 Amazon Aurora 数据库管理员，您需要保障数据库的安全，并满足合规性和法规要求。一种策略是集成数据库活动流与监控工具。通过这种方式，您可以在 Amazon Aurora 集群中监控审计活动并相应设置警报。

安全威胁既可以来自外部，也可以来自内部。要防范内部威胁，您可以通过配置数据库活动流功能控制管理员对数据流的访问。 数据管理员无权收集、传输、存储和处理流。

目录

- 数据库活动流的工作原理
- 数据库活动流的异步和同步模式
- 数据库活动流的要求和限制
- 区域和版本可用性
- 数据库活动流支持的数据库实例类

数据库活动流的工作原理

在 Amazon Aurora 中，您可以在集群级别启动数据库活动流。集群中的所有数据库实例都启用了数据库活动流。

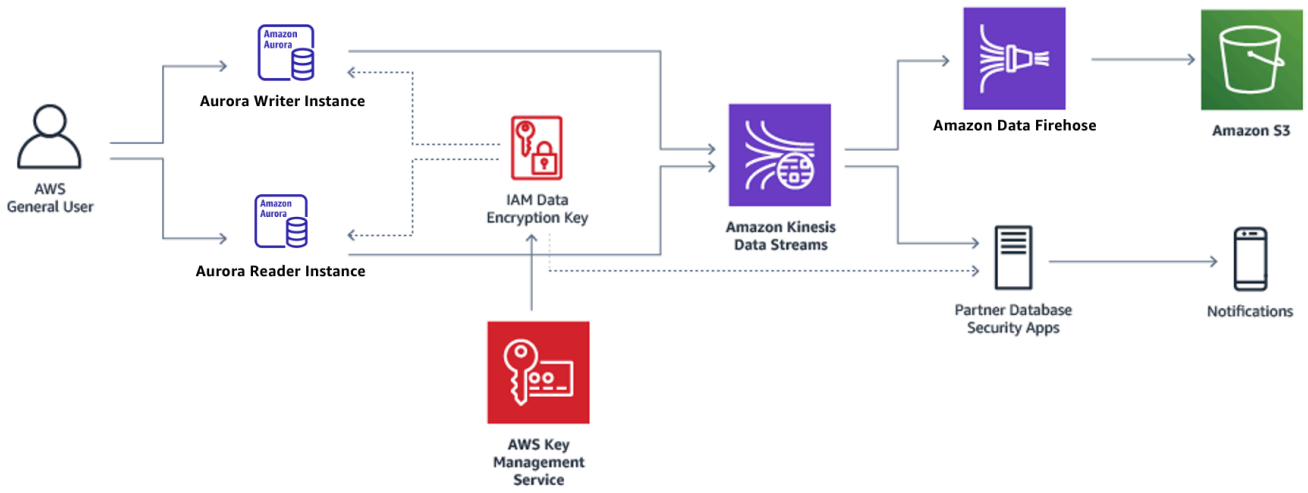
您的 Aurora 数据库集群会近乎实时地将活动推送到 Amazon Kinesis 数据流。系统将自动创建 Kinesis 流。在 Kinesis 中，您可以配置 AWS 服务（如 Amazon Data Firehose）和 AWS Lambda 来使用 Kinesis 流并存储数据。

> **Important**
>
> 使用 中的 Amazon Aurora数据库活动流功能是免费的，但 Amazon Kinesis 会针对数据流收费。有关更多信息，请参阅 Amazon Kinesis Data Streams 定价。

如果您使用 Aurora 全局数据库，请分别在每个数据库集群上启动数据库活动流。每个集群在其自己的 AWS 区域中将审计数据传送到其自己的 Kinesis 流。活动流在失效转移期间不会以不同方式运行。它们继续像往常一样审计您的全局数据库。

您可以为合规性管理配置应用程序，以使用数据库活动流。对于 Aurora PostgreSQL，合规性应用程序包括 IBM 的 Security Guardium 和 Imperva 的 SecureSphere Database Audit and Protection。这些应用程序可以使用流生成警报，并审计 Aurora 数据库集群上的活动。

下图显示了使用 Amazon Data Firehose 配置的 Aurora 数据库集群。



数据库活动流的异步和同步模式

您可以选择让数据库会话按以下任一模式处理数据库活动事件：

- 异步模式 – 当数据库会话生成活动流事件时，会话将立即返回到正常活动。在后台，活动流事件将成为持久记录。如果后台任务出错，则将发送 RDS 事件。此事件指示活动流事件记录可能已丢失的任何时间段的开始和结束时间。

  异步模式可提高数据库性能，而不是活动流的准确性。

  > **Note**
  >
  > 异步模式适用于 Aurora PostgreSQL 和 Aurora MySQL。

- 同步模式 – 当数据库会话生成活动流事件时，会话将阻止其他活动，直到该事件变为持久事件。如果因某个原因无法使事件持久，数据库会话将返回到正常活动。但会发送一个 RDS 事件来指示活动流记录可能会丢失一段时间。在系统恢复到正常运行状态后发送第二个 RDS 事件。

  同步模式可提高活动流的准确性，而不是数据库性能。

  > **Note**
  >
  > 同步模式适用于 Aurora PostgreSQL。您不能将同步模式用于 Aurora MySQL。

数据库活动流的要求和限制

在 Aurora 中，数据库活动流具有以下要求和限制：

- 数据库活动流需要使用 Amazon Kinesis。
- 数据库活动流需要使用 AWS Key Management Service (AWS KMS)，因为这些活动流始终是加密的。
- 对 Amazon Kinesis 数据流应用额外加密与数据库活动流不兼容，因为数据库活动流已使用 AWS KMS 密钥进行了加密。
- 在数据库集群级别启动数据库活动流。如果您向集群添加数据库实例，则无需在该实例上启动活动流：它会自动进行审计。
- 在 Aurora Global Database 中，请确保分别在每个数据库集群上启动活动流。每个集群在其自己的 AWS 区域中将审计数据传送到其自己的 Kinesis 流。
- 在 Aurora PostgreSQL 中，确保在主要版本升级之前停止数据库活动流。升级完成后，您可以启动数据库活动流。

区域和版本可用性

功能可用性和支持因每个 Aurora 数据库引擎的特定版本以及 AWS 区域而异。有关适用于 Aurora 和数据库活动流的版本和区域可用性的更多信息，请参阅 支持数据库活动流的区域和 Aurora 数据库引擎。

数据库活动流支持的数据库实例类

对于 Aurora MySQL，您可以将数据库活动流与以下数据库实例类一起使用：

- db.r8g.*large
- db.r7g.*large
- db.r7i.*large
- db.r6g.*large
- db.r6i.*large
- db.r5.*large
- db.x2g.*

对于 Aurora PostgreSQL，您可以将数据库活动流与以下数据库实例类一起使用：

- db.r8g.*large
- db.r7i.*large
- db.r7g.*large
- db.r6g.*large
- db.r6i.*large
- db.r6id.*large
- db.r5.*large
- db.r4.*large
- db.x2g.*

Aurora MySQL 数据库活动流的网络先决条件

在以下部分中，您可以了解如何将 Virtual Private Cloud (VPC) 配置为与数据库活动流结合使用。

> **Note**
>
> Aurora MySQL 网络先决条件适用于以下引擎版本：
>
> - Aurora MySQL 版本 2，最高 2.11.3
> - Aurora MySQL 版本 2.12.0
> - Aurora MySQL 版本 3，最高 3.04.2

主题

- AWS KMS 端点的先决条件
- 公开提供的先决条件
- 私有提供的先决条件

AWS KMS 端点的先决条件

Aurora MySQL 集群中使用活动流的实例必须能够访问 AWS KMS 端点。在为 Aurora MySQL 集群启用数据库活动流之前，请确保满足此要求。如果 Aurora 集群公开可用，则会自动满足此要求。

> **Important**
>
> 如果 Aurora MySQL 数据库集群无法访问 AWS KMS 端点，活动流将停止运行。在这种情况下，Aurora 会使用 RDS 事件通知您此问题。

公开提供的先决条件

对于要设为公有的 Aurora 数据库集群，必须满足以下要求：

- 在 AWS Management Console 集群详细信息页面中，公开访问为是。
- 数据库集群在 Amazon VPC 公有子网中。有关可公开访问的数据库实例的更多信息，请参阅 在 VPC 中使用数据库集群。有关公有 Amazon VPC 子网的更多信息，请参阅您的 VPC 和子网。

私有提供的先决条件

如果您的 Aurora 数据库集群位于 VPC 公有子网中且不可公开访问，说明它是私有的。要保持集群是私有的，并将其与数据库活动流结合使用，您可使用以下选项：

- 在 VPC 中配置网络地址转换 (NAT)。有关更多信息，请参阅 NAT 网关。
- 在 VPC 中创建 AWS KMS 端点。建议使用此选项，因为它更易于配置。

在 VPC 中创建 AWS KMS 端点

1. 通过以下网址打开 Amazon VPC 控制台：https://console.aws.amazon.com/vpc/。
2. 在导航窗格中，选择 Endpoints（端点）。
3. 选择 Create Endpoint（创建端点）。

   创建端点页面显示。
4. 执行以下操作：
   - 在服务类别中，选择 AWS 服务。
   - 在 Service Name（服务名称）中，选择 com.amazonaws.***region***.kms，其中 *region* 是集群所在的 AWS 区域。
   - 对于 VPC，选择集群所在的 VPC。
5. 选择 Create Endpoint（创建端点）。

有关配置 VPC 端点的更多信息，请参阅 VPC 端点。

启动数据库活动流

要监控 Aurora 数据库集群所有实例的数据库活动，请在集群级别启动活动流。还将自动监控添加到集群的任何数据库实例。如果您使用 Aurora 全局数据库，请分别在每个数据库集群上启动数据库活动流。每个集群在其自己的 AWS 区域中将审计数据传送到其自己的 Kinesis 流。

在启动活动流时，在审计策略中配置的每个数据库活动事件都会生成一个活动流事件。SQL 命令（例如 `CONNECT` 和 `SELECT`）可生成访问事件。SQL 命令（例如 `CREATE` 和 `INSERT`）可生成更改事件。

Console

要启动数据库活动流

1. 通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Databases (数据库)。
3. 选择要对其启动活动流的数据库集群。

4. 对于 Actions (操作)，选择 Start activity stream (启动活动流)。

   启动数据库活动流：##窗口出现，其中##是您的 数据库实例。

5. 输入以下设置：

   - 对于 AWS KMS key，从 AWS KMS keys 列表中选择一个密钥。

     > **Note**
     >
     > 如果您的 Aurora MySQL 集群无法访问 KMS 密钥，请先按照 Aurora MySQL 数据库活动流的网络先决条件 中的说明启用此类访问。

     Aurora 使用 KMS 密钥加密密钥，从而加密数据库活动。请选择原定设置密钥以外的 KMS 密钥。有关加密密钥和 AWS KMS 的更多信息，请参阅 AWS Key Management Service 开发人员指南中的什么是 AWS Key Management Service？。

   - 对于 Database activity stream mode (数据库活动流模式)，选择 Asynchronous (异步) 或 Synchronous (同步)。

     > **Note**
     >
     > 此选项仅适用于 Aurora PostgreSQL。对于 Aurora MySQL，您只能使用异步模式。

   - 选择 Immediately (立即)。

     当您选择 Immediately (立即) 时，数据库集群会立即重新启动。如果选择 During the next maintenance window (在下一维护时段内)，数据库集群不会立即重新启动。在这种情况下，数据库活动流不会启动，直到下一个维护时段。

6. 选择 Start database activity stream（启动数据库活动流）。

   数据库集群的状态显示活动流正在启动。

   > **Note**
   >
   > 如果您收到错误 `You can't start a database activity stream in this configuration`，请检查 数据库活动流支持的数据库实例类，以了解您的数据库集群是否正在使用受支持的实例类。

AWS CLI

要为数据库集群启动数据库活动流，请使用 AWS CLI 命令 start-activity-stream 配置数据库。

- `--resource-arn` *arn* – 指定数据库集群的 Amazon Resource Name (ARN)。
- `--mode` *sync-or-async* – 指定同步 (sync) 或异步 (async) 模式。对于 Aurora PostgreSQL，您可以选择任一值。对于 Aurora MySQL，请指定 `async`。
- `--kms-key-id` *key* – 指定用于加密数据库活动流中的消息的 KMS 密钥标识符。AWS KMS 密钥标识符是密钥 ARN、密钥 ID、别名 ARN 或者 AWS KMS key 的别名。

以下示例以异步模式启动数据库集群的数据库活动流。

对于 Linux、macOS 或 Unix：

```
aws rds start-activity-stream \
    --mode async \
    --kms-key-id my-kms-key-arn \
    --resource-arn my-cluster-arn \
    --apply-immediately
```

对于 Windows：

```
aws rds start-activity-stream ^
    --mode async ^
    --kms-key-id my-kms-key-arn ^
    --resource-arn my-cluster-arn ^
    --apply-immediately
```

Amazon RDS API

要为数据库集群启动数据库活动流，请使用 StartActivityStream 操作配置集群。

使用以下参数调用操作：

- `Region`
- `KmsKeyId`
- `ResourceArn`
- `Mode`

> **Note**
>
> 如果您收到一条错误，指出您无法使用当前版本的 Aurora PostgreSQL 数据库启动数据库活动流，请先应用 Aurora PostgreSQL 的最新补丁，然后再启动数据库活动流。有关升级 Aurora PostgreSQL 数据库的信息，请参阅升级 Amazon Aurora 数据库集群。
> 以下是用于使用 Aurora PostgreSQL 启动数据库活动流的最低补丁版本。
>
> - 3.4.15 (11.9.15) 、11.21.10
> - 12.9.15、12.15.9、12.16.10、12.17.7、12.18.5、12.19.4、12.20.3、12.22.3
> - 13.9.12、13.11.9、13.12.10、13.13.7、13.14.5、13.15.4、13.16.3、13.18.3
> - 14.6.12、14.8.9、14.9.10、14.10.7、14.11.5、14.12.4、14.13.3、14.15.3
> - 15.3.9、15.4.10、15.5.7、15.6.5、15.7.4、15.8.3、15.10.3
> - 16.1.7、16.2.5、16.3.4、16.4.3、16.6.3

获取数据库活动流的状态

您可以使用控制台或 AWS CLI 获取活动流的状态。

控制台

要获取数据库活动流的状态，请执行以下操作

1. 通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Databases (数据库)，然后选择数据库集群链接。
3. 选择 Configuration (配置) 选项卡，并选择 Database activity stream (数据库活动流) 以查看状态。

AWS CLI

您可以获取数据库集群的活动流配置，作为对 describe-db-clusters CLI 请求的响应。

以下示例描述 *my-cluster*。

```
aws rds --region my-region describe-db-clusters --db-cluster-identifier my-cluster
```

下面的示例介绍一个 JSON 响应。显示了以下字段：

- `ActivityStreamKinesisStreamName`

- ActivityStreamKmsKeyId
- ActivityStreamStatus
- ActivityStreamMode
-

这些字段对于 Aurora PostgreSQL 和 Aurora MySQL 是相同的，只是对于 Aurora MySQL，ActivityStreamMode 总是 async，而对于 Aurora PostgreSQL，它可能是 sync 或 async。

```
{
    "DBClusters": [
        {
      "DBClusterIdentifier": "my-cluster",
            ...
            "ActivityStreamKinesisStreamName": "aws-rds-das-cluster-
A6TSYXITZCZXJHIRVFUBZ5LTWY",
            "ActivityStreamStatus": "starting",
            "ActivityStreamKmsKeyId": "12345678-abcd-efgh-ijkl-bd041f170262",
            "ActivityStreamMode": "async",
            "DbClusterResourceId": "cluster-ABCD123456"
            ...
        }
    ]
}
```

RDS API

您可以获取数据库集群的活动流配置，作为对 DescribeDBClusters 操作的响应。

停止数据库活动流

您可以使用控制台或 AWS CLI 停止活动流。

如果您删除数据库集群，活动流将停止，并且会自动删除底层 Amazon Kinesis 流。

控制台

关闭活动流

1. 通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Databases (数据库)。

3. 选择要为其停止数据库活动流的数据库集群。
4. 对于 Actions (操作)，选择 Stop activity stream (停止活动流)。此时将显示 Database Activity Stream (数据库活动流) 窗口。

   a. 选择 Immediately (立即)。

      当您选择 Immediately (立即) 时，数据库集群会立即重新启动。如果选择 During the next maintenance window (在下一维护时段内)，数据库集群不会立即重新启动。在这种情况下，数据库活动流直到下一个维护时段才会停止。

   b. 选择 Continue (继续)。

AWS CLI

要为数据库集群停止数据库活动流，请使用 AWS CLI 命令 stop-activity-stream 配置数据库集群。使用 `--region` 参数标识数据库集群的AWS区域。`--apply-immediately` 参数是可选的。

对于 Linux、macOS 或 Unix：

```
aws rds --region MY_REGION \
    stop-activity-stream \
    --resource-arn MY_CLUSTER_ARN \
    --apply-immediately
```

对于 Windows：

```
aws rds --region MY_REGION ^
    stop-activity-stream ^
    --resource-arn MY_CLUSTER_ARN ^
    --apply-immediately
```

RDS API

要为数据库集群停止数据库活动流，请使用 StopActivityStream 操作配置集群。使用 `Region` 参数标识数据库集群的AWS区域。`ApplyImmediately` 参数是可选的。

监控数据库活动流

数据库活动流监控并报告活动。收集活动流并将其传输到 Amazon Kinesis。从 Kinesis 中，您可以监控活动流，或者其他服务和应用程序可以使用活动流进行进一步分析。您可以使用 AWS CLI 命令 `describe-db-clusters` 或 RDS API `DescribeDBClusters` 操作查找底层 Kinesis 流名称。

Aurora 为您管理 Kinesis 流，如下所示：

- Aurora 自动创建具有 24 小时保留期的 Kinesis 流。
- 如有必要，Aurora 会扩展 Kinesis 流。
- 如果停止了数据库活动流或删除了数据库集群，Aurora 将删除 Kinesis 流。

监控以下类别的活动并将其放入活动流审核日志中：

- SQL 命令 – 将审计所有 SQL 命令，以及预编译语句、内置函数和采用 PL/SQL 的函数。将审核对存储过程的调用。还将审核在存储过程或函数中发出的任何 SQL 语句。
- 其他数据库信息 – 受监控的活动包括完整的 SQL 语句、来自 DML 命令的受影响行的行数、访问的对象以及唯一的数据库名称。对于 Aurora PostgreSQL，数据库活动流还监控绑定变量和存储过程参数。

  > **Important**
  >
  > 每个语句的完整 SQL 文本在活动流审核日志中可见，包括任何敏感数据。但是，如果 Aurora 可以从上下文中确定数据库用户密码，则会对该密码进行修订，如下面的 SQL 语句所示。
  >
  > ```
  > ALTER ROLE role-name WITH password
  > ```

- 连接信息 – 受监控的活动包括会话和网络信息、服务器进程 ID 和退出代码。

如果在监控数据库实例时活动流发生故障，则会使用 RDS 事件通知您。

在以下各节中，您可以访问、审计和处理数据库活动流。

主题

- 从 Amazon Kinesis 访问活动流
- 数据库活动流的审计日志内容和示例
- 数据库活动流的 databaseActivityEventList JSON 数组
- 使用 AWS 开发工具包处理数据库活动流

从 Amazon Kinesis 访问活动流

在为数据库集群启用活动流时，将会为您创建一个 Kinesis 流。您可以从 Kinesis 实时监控数据库活动。要进一步分析数据库活动，您可以将 Kinesis 流连接到使用者应用程序。您还可以将流连接到合规性管理应用程序，例如 IBM 的 Security Guardium 或 Imperva 的 SecureSphere Database Audit and Protection。

您可以从 RDS 控制台或 Kinesis 控制台访问您的 Kinesis 流。

使用 RDS 控制台从 Kinesis 中访问活动流

1. 通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Databases（数据库）。
3. 选择已启动活动流的数据库集群。
4. 选择配置。
5. 在 Database activity stream（数据库活动流）下，选择 Kinesis stream（Kinesis 流）下的链接。
6. 在 Kinesis 控制台中，选择 Monitoring（监控）以开始观察数据库活动。

使用 Kinesis 控制台从 Kinesis 中访问活动流

1. 打开 Kinesis 控制台，网址为：https://console.aws.amazon.com/kinesis。
2. 从 Kinesis 流列表中选择您的活动流。

   活动流的名称包含前缀 `aws-rds-das-cluster-`，后跟数据库集群的资源 ID。以下是示例。

   ```
   aws-rds-das-cluster-NHVOV4PCLWHGF52NP
   ```

   要使用 Amazon RDS 控制台查找数据库集群的资源 ID，请从数据库列表中选择数据库集群，然后选择 Configuration（配置）选项卡。

   要使用 AWS CLI 查找活动流的完整 Kinesis 流名称，请使用 describe-db-clusters CLI 请求并记下响应中的 `ActivityStreamKinesisStreamName` 值。
3. 选择监控以开始观察数据库活动。

有关使用 Amazon Kinesis 的更多信息，请参阅什么是 Amazon Kinesis Data Streams？

数据库活动流的审计日志内容和示例

受监控的事件在数据库活动流中表示为 JSON 字符串。结构包含一个 JSON 对象，该对象包含一个 `DatabaseActivityMonitoringRecord`，后者反过来包含活动事件的 `databaseActivityEventList` 阵列。

> **Note**
>
> 对于数据库活动流，`paramList` JSON 数组不包含来自 Hibernate 应用程序的空值。

主题

- 活动流的审计日志示例
- DatabaseActivityMonitoringRecords JSON 对象
- databaseActivityEvents JSON 对象

活动流的审计日志示例

以下是活动事件记录的已解密 JSON 审核日志示例。

Example Aurora PostgreSQLCONNECT SQL 语句的活动事件记录

以下活动事件记录显示 psql 客户端（`clientApplication`）使用 `CONNECT` SQL 语句（`command`）进行的登录。

```
{
  "type":"DatabaseActivityMonitoringRecords",
  "version":"1.1",
  "databaseActivityEvents":
    {
      "type":"DatabaseActivityMonitoringRecord",
      "clusterId":"cluster-4HNY5V4RRNPKKYB7ICFKE5JBQQ",
      "instanceId":"db-FZJTMYKCXQBUUZ6VLU7NW3ITCM",
      "databaseActivityEventList":[
        {
          "startTime": "2019-10-30 00:39:49.940668+00",
          "logTime": "2019-10-30 00:39:49.990579+00",
          "statementId": 1,
          "substatementId": 1,
          "objectType": null,
          "command": "CONNECT",
```

```
          "objectName": null,
          "databaseName": "postgres",
          "dbUserName": "rdsadmin",
          "remoteHost": "172.31.3.195",
          "remotePort": "49804",
          "sessionId": "5ce5f7f0.474b",
          "rowCount": null,
          "commandText": null,
          "paramList": [],
          "pid": 18251,
          "clientApplication": "psql",
          "exitCode": null,
          "class": "MISC",
          "serverVersion": "2.3.1",
          "serverType": "PostgreSQL",
          "serviceName": "Amazon Aurora PostgreSQL-Compatible edition",
          "serverHost": "172.31.3.192",
          "netProtocol": "TCP",
          "dbProtocol": "Postgres 3.0",
          "type": "record",
          "errorMessage": null
        }
      ]
    },
   "key":"decryption-key"
}
```

Example Aurora MySQL CONNECT SQL 语句的活动事件记录

以下活动事件记录显示 mysql 客户端（`clientApplication`）使用 CONNECT SQL 语句（command）进行的登录。

```
{
  "type":"DatabaseActivityMonitoringRecord",
  "clusterId":"cluster-some_id",
  "instanceId":"db-some_id",
  "databaseActivityEventList":[
    {
      "logTime":"2020-05-22 18:07:13.267214+00",
      "type":"record",
      "clientApplication":null,
      "pid":2830,
      "dbUserName":"rdsadmin",
```

```
      "databaseName":"",
      "remoteHost":"localhost",
      "remotePort":"11053",
      "command":"CONNECT",
      "commandText":"",
      "paramList":null,
      "objectType":"TABLE",
      "objectName":"",
      "statementId":0,
      "substatementId":1,
      "exitCode":"0",
      "sessionId":"725121",
      "rowCount":0,
      "serverHost":"master",
      "serverType":"MySQL",
      "serviceName":"Amazon Aurora MySQL",
      "serverVersion":"MySQL 5.7.12",
      "startTime":"2020-05-22 18:07:13.267207+00",
      "endTime":"2020-05-22 18:07:13.267213+00",
      "transactionId":"0",
      "dbProtocol":"MySQL",
      "netProtocol":"TCP",
      "errorMessage":"",
      "class":"MAIN"
    }
  ]
}
```

Example Aurora PostgreSQL的活动事件记录

以下示例显示 Aurora PostgreSQL的 CREATE TABLE 事件。

```
{
  "type":"DatabaseActivityMonitoringRecords",
  "version":"1.1",
  "databaseActivityEvents":
    {
      "type":"DatabaseActivityMonitoringRecord",
      "clusterId":"cluster-4HNY5V4RRNPKKYB7ICFKE5JBQQ",
      "instanceId":"db-FZJTMYKCXQBUUZ6VLU7NW3ITCM",
      "databaseActivityEventList":[
        {
          "startTime": "2019-05-24 00:36:54.403455+00",
          "logTime": "2019-05-24 00:36:54.494235+00",
```

```
          "statementId": 2,
          "substatementId": 1,
          "objectType": null,
          "command": "CREATE TABLE",
          "objectName": null,
          "databaseName": "postgres",
          "dbUserName": "rdsadmin",
          "remoteHost": "172.31.3.195",
          "remotePort": "34534",
          "sessionId": "5ce73c6f.7e64",
          "rowCount": null,
          "commandText": "create table my_table (id serial primary key, name
 varchar(32));",
          "paramList": [],
          "pid": 32356,
          "clientApplication": "psql",
          "exitCode": null,
          "class": "DDL",
          "serverVersion": "2.3.1",
          "serverType": "PostgreSQL",
          "serviceName": "Amazon Aurora PostgreSQL-Compatible edition",
          "serverHost": "172.31.3.192",
          "netProtocol": "TCP",
          "dbProtocol": "Postgres 3.0",
          "type": "record",
          "errorMessage": null
        }
      ]
    },
   "key":"decryption-key"
}
```

Example Aurora MySQL CREATE TABLE 语句的活动事件记录

以下示例显示 Aurora MySQL 的 CREATE TABLE 语句。操作表示为两个单独的事件记录。一个事件具有 "class":"MAIN"。另一个事件具有 "class":"AUX"。消息可能按任何顺序到达。logTime 事件的 MAIN 字段始终早于任何对应 logTime 事件的 AUX 字段。

以下示例显示 class 值为 MAIN 的事件。

```
{
  "type":"DatabaseActivityMonitoringRecord",
  "clusterId":"cluster-some_id",
```

```
  "instanceId":"db-some_id",
  "databaseActivityEventList":[
    {
      "logTime":"2020-05-22 18:07:12.250221+00",
      "type":"record",
      "clientApplication":null,
      "pid":2830,
      "dbUserName":"master",
      "databaseName":"test",
      "remoteHost":"localhost",
      "remotePort":"11054",
      "command":"QUERY",
      "commandText":"CREATE TABLE test1 (id INT)",
      "paramList":null,
      "objectType":"TABLE",
      "objectName":"test1",
      "statementId":65459278,
      "substatementId":1,
      "exitCode":"0",
      "sessionId":"725118",
      "rowCount":0,
      "serverHost":"master",
      "serverType":"MySQL",
      "serviceName":"Amazon Aurora MySQL",
      "serverVersion":"MySQL 5.7.12",
      "startTime":"2020-05-22 18:07:12.226384+00",
      "endTime":"2020-05-22 18:07:12.250222+00",
      "transactionId":"0",
      "dbProtocol":"MySQL",
      "netProtocol":"TCP",
      "errorMessage":"",
      "class":"MAIN"
    }
  ]
}
```

以下示例显示 `class` 值为 AUX 的相应事件。

```
{
  "type":"DatabaseActivityMonitoringRecord",
  "clusterId":"cluster-some_id",
  "instanceId":"db-some_id",
  "databaseActivityEventList":[
```

```
    {
      "logTime":"2020-05-22 18:07:12.247182+00",
      "type":"record",
      "clientApplication":null,
      "pid":2830,
      "dbUserName":"master",
      "databaseName":"test",
      "remoteHost":"localhost",
      "remotePort":"11054",
      "command":"CREATE",
      "commandText":"test1",
      "paramList":null,
      "objectType":"TABLE",
      "objectName":"test1",
      "statementId":65459278,
      "substatementId":2,
      "exitCode":"",
      "sessionId":"725118",
      "rowCount":0,
      "serverHost":"master",
      "serverType":"MySQL",
      "serviceName":"Amazon Aurora MySQL",
      "serverVersion":"MySQL 5.7.12",
      "startTime":"2020-05-22 18:07:12.226384+00",
      "endTime":"2020-05-22 18:07:12.247182+00",
      "transactionId":"0",
      "dbProtocol":"MySQL",
      "netProtocol":"TCP",
      "errorMessage":"",
      "class":"AUX"
    }
  ]
}
```

Example Aurora PostgreSQL的活动事件记录

以下示例显示 SELECT 事件。

```
{
  "type":"DatabaseActivityMonitoringRecords",
  "version":"1.1",
  "databaseActivityEvents":
    {
      "type":"DatabaseActivityMonitoringRecord",
```

```
      "clusterId":"cluster-4HNY5V4RRNPKKYB7ICFKE5JBQQ",
      "instanceId":"db-FZJTMYKCXQBUUZ6VLU7NW3ITCM",
      "databaseActivityEventList":[
        {
          "startTime": "2019-05-24 00:39:49.920564+00",
          "logTime": "2019-05-24 00:39:49.940668+00",
          "statementId": 6,
          "substatementId": 1,
          "objectType": "TABLE",
          "command": "SELECT",
          "objectName": "public.my_table",
          "databaseName": "postgres",
          "dbUserName": "rdsadmin",
          "remoteHost": "172.31.3.195",
          "remotePort": "34534",
          "sessionId": "5ce73c6f.7e64",
          "rowCount": 10,
          "commandText": "select * from my_table;",
          "paramList": [],
          "pid": 32356,
          "clientApplication": "psql",
          "exitCode": null,
          "class": "READ",
          "serverVersion": "2.3.1",
          "serverType": "PostgreSQL",
          "serviceName": "Amazon Aurora PostgreSQL-Compatible edition",
          "serverHost": "172.31.3.192",
          "netProtocol": "TCP",
          "dbProtocol": "Postgres 3.0",
          "type": "record",
          "errorMessage": null
        }
      ]
    },
   "key":"decryption-key"
}
```



```
{
    "type": "DatabaseActivityMonitoringRecord",
    "clusterId": "",
    "instanceId": "db-4JCWQLUZVFYP7DIWP6JVQ77O3Q",
    "databaseActivityEventList": [
        {
```

```
            "class": "TABLE",
            "clientApplication": "Microsoft SQL Server Management Studio - Query",
            "command": "SELECT",
            "commandText": "select * from [testDB].[dbo].[TestTable]",
            "databaseName": "testDB",
            "dbProtocol": "SQLSERVER",
            "dbUserName": "test",
            "endTime": null,
            "errorMessage": null,
            "exitCode": 1,
            "logTime": "2022-10-06 21:24:59.9422268+00",
            "netProtocol": null,
            "objectName": "TestTable",
            "objectType": "TABLE",
            "paramList": null,
            "pid": null,
            "remoteHost": "local machine",
            "remotePort": null,
            "rowCount": 0,
            "serverHost": "172.31.30.159",
            "serverType": "SQLSERVER",
            "serverVersion": "15.00.4073.23.v1.R1",
            "serviceName": "sqlserver-ee",
            "sessionId": 62,
            "startTime": null,
            "statementId": "0x03baed90412f564fad640ebe51f89b99",
            "substatementId": 1,
            "transactionId": "4532935",
            "type": "record",
            "engineNativeAuditFields": {
                "target_database_principal_id": 0,
                "target_server_principal_id": 0,
                "target_database_principal_name": "",
                "server_principal_id": 2,
                "user_defined_information": "",
                "response_rows": 0,
                "database_principal_name": "dbo",
                "target_server_principal_name": "",
                "schema_name": "dbo",
                "is_column_permission": true,
                "object_id": 581577110,
                "server_instance_name": "EC2AMAZ-NFUJJNO",
                "target_server_principal_sid": null,
                "additional_information": "",
```

```
                "duration_milliseconds": 0,
                "permission_bitmask": "0x00000000000000000000000000000001",
                "data_sensitivity_information": "",
                "session_server_principal_name": "test",
                "connection_id": "AD3A5084-FB83-45C1-8334-E923459A8109",
                "audit_schema_version": 1,
                "database_principal_id": 1,
                "server_principal_sid":
 "0x0105000000000005150000000bdc2795e2d0717901ba6998cf4010000",
                "user_defined_event_id": 0,
                "host_name": "EC2AMAZ-NFUJJNO"
            }
        }
    ]
}
```

Example Aurora MySQL SELECT 语句的活动事件记录

以下示例显示 SELECT 事件。

以下示例显示 `class` 值为 `MAIN` 的事件。

```
{
  "type":"DatabaseActivityMonitoringRecord",
  "clusterId":"cluster-some_id",
  "instanceId":"db-some_id",
  "databaseActivityEventList":[
    {
      "logTime":"2020-05-22 18:29:57.986467+00",
      "type":"record",
      "clientApplication":null,
      "pid":2830,
      "dbUserName":"master",
      "databaseName":"test",
      "remoteHost":"localhost",
      "remotePort":"11054",
      "command":"QUERY",
      "commandText":"SELECT * FROM test1 WHERE id < 28",
      "paramList":null,
      "objectType":"TABLE",
      "objectName":"test1",
      "statementId":65469218,
      "substatementId":1,
      "exitCode":"0",
```

```
      "sessionId":"726571",
      "rowCount":2,
      "serverHost":"master",
      "serverType":"MySQL",
      "serviceName":"Amazon Aurora MySQL",
      "serverVersion":"MySQL 5.7.12",
      "startTime":"2020-05-22 18:29:57.986364+00",
      "endTime":"2020-05-22 18:29:57.986467+00",
      "transactionId":"0",
      "dbProtocol":"MySQL",
      "netProtocol":"TCP",
      "errorMessage":"",
      "class":"MAIN"
    }
  ]
}
```

以下示例显示 class 值为 AUX 的相应事件。

```
{
  "type":"DatabaseActivityMonitoringRecord",
  "instanceId":"db-some_id",
  "databaseActivityEventList":[
    {
      "logTime":"2020-05-22 18:29:57.986399+00",
      "type":"record",
      "clientApplication":null,
      "pid":2830,
      "dbUserName":"master",
      "databaseName":"test",
      "remoteHost":"localhost",
      "remotePort":"11054",
      "command":"READ",
      "commandText":"test1",
      "paramList":null,
      "objectType":"TABLE",
      "objectName":"test1",
      "statementId":65469218,
      "substatementId":2,
      "exitCode":"",
      "sessionId":"726571",
      "rowCount":0,
      "serverHost":"master",
```

```
      "serverType":"MySQL",
      "serviceName":"Amazon Aurora MySQL",
      "serverVersion":"MySQL 5.7.12",
      "startTime":"2020-05-22 18:29:57.986364+00",
      "endTime":"2020-05-22 18:29:57.986399+00",
      "transactionId":"0",
      "dbProtocol":"MySQL",
      "netProtocol":"TCP",
      "errorMessage":"",
      "class":"AUX"
    }
  ]
}
```

DatabaseActivityMonitoringRecords JSON 对象

数据库活动事件记录位于包含以下信息的 JSON 对象中。

JSON 字段	数据类型	描述
type	字符串	JSON 记录的类型。值为 DatabaseActivityMonitoringRecords 。
version	字符串	数据库活动监控记录的版本。 所生成数据库活动记录的版本取决于数据库集群的引擎版本： • 对于运行引擎版本 10.10 和更高次要版本以及引擎版本 11.5 和更高版本的 Aurora PostgreSQL 数据库集群，生成版本 1.1 的数据库活动记录。 • 对于运行引擎版本 10.7 和 11.4 的 Aurora PostgreSQL 数据库集群，生成版本 1.0 的数据库活动记录。 除非另有注明，否则版本 1.0 和版本 1.1 中均包含以下所有字段。

JSON 字段	数据类型	描述
databaseActivityEvents	字符串	包含活动事件的 JSON 对象。
键	字符串	用于解密 databaseActivityEventList JSON 数组 的加密密钥

databaseActivityEvents JSON 对象

`databaseActivityEvents` JSON 对象包含以下信息。

JSON 记录中的顶级字段

审核日志中的每个事件都包装在 JSON 格式的记录中。此记录包含以下字段。

类型

此字段始终具有值 `DatabaseActivityMonitoringRecords`。

版本

此字段表示数据库活动流数据协议或合同的版本。它定义了哪些字段可用。

版本 1.0 表示对 Aurora PostgreSQL 10.7 和 11.4 版的原始数据活动流支持。版本 1.1 表示对 Aurora PostgreSQL 10.10 版及更高版本和 Aurora PostgreSQL 11.5 版及更高版本的数据活动流支持。版本 1.1 包括其他字段 `errorMessage` 和 `startTime`。版本 1.2 表示对 Aurora MySQL 2.08 及更高版本的数据活动流支持。版本 1.2 包括其他字段 `endTime` 和 `transactionId`。

databaseActivityEvents

表示一个或多个活动事件的加密字符串。它表示为 base64 字节数组。解密字符串时，结果是 JSON 格式的记录，其中包含字段，如本节中的示例所示。

key

用于加密 `databaseActivityEvents` 字符串的加密数据密钥。这与您在启动数据库活动流时提供的 AWS KMS key 密钥相同。

以下示例显示了此记录的格式。

```
{
```

```
  "type":"DatabaseActivityMonitoringRecords",
  "version":"1.1",
  "databaseActivityEvents":"encrypted audit records",
  "key":"encrypted key"
}
```

执行以下步骤来解密 databaseActivityEvents 字段的内容：

1. 使用您在启动数据库活动流时提供的 KMS 密钥解密 key JSON 字段中的值。这样做将以明文形式返回数据加密密钥。
2. 对 databaseActivityEvents JSON 字段中的值进行 Base64 解码，以获取审核负载的二进制格式的密文。
3. 使用您在第一步中解码的数据加密密钥解密二进制密文。
4. 解压解已解密的负载。
   - 已加密的负载在 databaseActivityEvents 字段中。
   - databaseActivityEventList 字段包含审核记录数组。此数组中的 type 字段可以是 record 或 heartbeat。

审核日志活动事件记录是包含以下信息的 JSON 对象。

JSON 字段	数据类型	描述
type	字符串	JSON 记录的类型。值为 DatabaseActivityMonitoringRecord 。
clusterId	字符串	数据库集群资源标识符。它对应于数据库集群属性 DbClusterResourceId 。
instanceId	字符串	数据库实例资源标识符。它对应于数据库实例属性 DbiResourceId 。
databaseActivityEventList JSON 数组	字符串	活动审核记录或检测信号消息的数组。

数据库活动流的 databaseActivityEventList JSON 数组

审核日志负载是解密的 `databaseActivityEventList` JSON 数组。以下表列表按字母顺序列出了审计日志的解密 `DatabaseActivityEventList` 数组中每个活动事件的字段。这些字段会因您使用的是 Aurora PostgreSQL 还是 Aurora MySQL 而有所不同。请参阅适用于您的数据库引擎的表。

> **Important**
>
> 事件结构可能会发生变化。Aurora 可能会将新字段添加到未来的活动事件中。在解析 JSON 数据的应用程序中，请确保您的代码可以忽略未知字段名称或对其采取适当操作。

Aurora PostgreSQL 的 databaseActivityEventList 字段

以下是 Aurora PostgreSQL 的 `databaseActivityEventList` 字段。

字段	数据类型	描述
`class`	字符串	活动事件的类。Aurora PostgreSQL 的有效值如下所示： • `ALL` • `CONNECT` – 连接或断开连接事件。 • `DDL` – 未包含在 `ROLE` 类的语句列表中的 DDL 语句。 • `FUNCTION` – 函数调用或 `DO` 块。 • `MISC` – 其他命令，例如 `DISCARD`、`FETCH`、`CHECKPOINT` 或 `VACUUM`。 • `NONE` • `READ` – `SELECT` 或 `COPY` 语句（当源为关系或查询时）。 • `ROLE` – 与角色或权限关联的语句，包括 `GRANT`、`REVOKE` 和 `CREATE/ALTER/DROP ROLE`。 • `WRITE` – `INSERT`、`UPDATE`、`DELETE`、`TRUNCATE` 或 `COPY` 语句（当目标为关系时）。
`clientApplication`	字符串	客户端报告的其用于连接的应用程序。由于客户端不必提供此信息，因此值可以为 null。
`command`	字符串	不带任何命令详细信息的 SQL 命令的名称。

字段	数据类型	描述
commandText	字符串	用户传入的实际 SQL 语句。对于 Aurora PostgreSQL，该值与原始 SQL 语句相同。此字段用于除连接或断开连接记录之外的所有类型的记录，在这种情况下，该值为 null。 **Important** 每个语句的完整 SQL 文本在活动流审核日志中可见，包括任何敏感数据。但是，如果 Aurora 可以从上下文中确定数据库用户密码，则会对该密码进行修订，如下面的 SQL 语句所示。 `ALTER ROLE role-name WITH password`
databaseName	字符串	用户连接到的数据库。
dbProtocol	字符串	数据库协议，例如 Postgres 3.0。
dbUserName	字符串	客户端对其进行身份验证的数据库用户。
errorMessage （仅版本 1.1 数据库活动记录）	字符串	如果出现任何错误，则使用数据库服务器生成的错误消息填充此字段。对于未导致错误的普通语句，errorMessage 值为 null。 错误定义为生成客户端可见 PostgreSQL 错误日志事件（其严重性级别为 ERROR 或更高）的任何活动。有关更多信息，请参阅 PostgreSQL 消息严重性级别。例如，语法错误和查询取消会生成错误消息。 内部 PostgreSQL 服务器错误（例如后台检查指针进程错误）不会生成错误消息。但是，无论如何设置日志严重性级别，此类事件的记录仍会发出。这样可防止攻击者关闭日志记录以尝试避开检测。 另请参阅 exitCode 字段。

字段	数据类型	描述
exitCode	int	用于会话退出记录的值。在干净的出口，这包含退出代码。在某些故障场景中，无法始终获得退出代码。例如，如果 PostgreSQL 执行 exit() 或操作者执行 kill -9 等命令。 如果存在任何错误，则 exitCode 字段将显示 SQL 错误代码 SQLSTATE，如 PostgreSQL 错误代码中所列。 另请参阅 errorMessage 字段。
logTime	字符串	审核代码路径中记录的时间戳。这表示 SQL 语句执行结束时间。另请参阅 startTime 字段。
netProtocol	字符串	网络通信协议。
objectName	字符串	数据库对象的名称（如果正在对一个数据库对象运行 SQL 语句）。此字段仅在对数据库对象运行 SQL 语句时使用。如果未对一个对象运行 SQL 语句，则此值为 null。
objectType	字符串	数据库对象类型，例如表、索引、视图等。此字段仅在对数据库对象运行 SQL 语句时使用。如果未对一个对象运行 SQL 语句，则此值为 null。包括下列有效值： • COMPOSITE TYPE • FOREIGN TABLE • FUNCTION • INDEX • MATERIALIZED VIEW • SEQUENCE • TABLE • TOAST TABLE • VIEW • UNKNOWN
paramList	字符串	传递给 SQL 语句的逗号分隔的参数数组。如果 SQL 语句没有参数，则此值为空数组。

字段	数据类型	描述
pid	int	为向客户端连接提供服务而分配的后端进程的进程 ID。
remoteHost	字符串	客户端 IP 地址或主机名。对于 Aurora PostgreSQL，使用哪一个取决于数据库的 log_hostname 参数设置。remoteHost 值还包括指示活动来自 rdsadmin 用户的 [local] 和 localhost 。
remotePort	字符串	客户端的端口号。
rowCount	int	SQL 语句所影响或检索的表行的数目。此字段仅用于作为数据操作语言 (DML) 语句的 SQL 语句。如果 SQL 语句不是 DML 语句，则此值为 null。
serverHost	字符串	数据库服务器主机 IP 地址。serverHost 值还包括指示活动来自 rdsadmin 用户的 [local] 和 localhost 。
serverType	字符串	数据库服务器类型，例如 PostgreSQL 。
serverVersion	字符串	数据库服务器版本，例如对于 Aurora PostgreSQL 为 2.3.1。
serviceName	字符串	服务名称，例如 Amazon Aurora PostgreSQL-Compatible edition 。
sessionId	int	伪唯一会话标识符。
sessionId	int	伪唯一会话标识符。
startTime （仅版本 1.1 数据库活动记录）	字符串	SQL 语句开始执行的时间。 要计算 SQL 语句的近似执行时间，请使用 logTime - startTime 。另请参阅 logTime 字段。
statementId	int	客户端的 SQL 语句的标识符。计数器处于会话级别，并随客户端输入的每个 SQL 语句递增。
substatementId	int	SQL 子语句的标识符。此值计算 statementId 字段标识的每个 SQL 语句的包含的子语句。

字段	数据类型	描述
type	字符串	事件类型。有效值为 record 或 heartbeat 。

Aurora MySQL 的 databaseActivityEventList 字段

以下是 Aurora MySQL 的 databaseActivityEventList 字段。

字段	数据类型	描述
class	字符串	活动事件的类。 Aurora MySQL 的有效值如下所示： • MAIN – 表示 SQL 语句的主事件。 • AUX – 包含其他详细信息的补充事件。例如，重命名对象的语句可能具有反映新名称的类为 AUX 的事件。 要查找对应于同一语句的 MAIN 和 AUX 事件，请检查具有相同的 pid 字段值和 statementId 字段值的不同事件。
clientApplication	字符串	客户端报告的其用于连接的应用程序。由于客户端不必提供此信息，因此值可以为 null。
command	字符串	SQL 语句的常规类别。此字段的值取决于 class 的值。 class 为 MAIN 时的值包括以下值： • CONNECT – 连接客户端会话时。 • QUERY – SQL 语句。附带一个或多个 class 值为 AUX 的事件。 • DISCONNECT – 客户端会话断开连接时。 • FAILED_CONNECT – 客户端尝试连接但无法连接时。 • CHANGEUSER – 属于 MySQL 网络协议的一部分但不是来自您发出的语句的状态更改。 class 为 AUX 时的值包括以下值：

字段	数据类型	描述
		• READ – SELECT 或 COPY 语句（当源为关系或查询时）。 • WRITE – INSERT、UPDATE、DELETE、TRUNCATE 或 COPY 语句（当目标为关系时）。 • DROP – 删除对象。 • CREATE – 创建对象。 • RENAME – 重命名对象。 • ALTER – 更改对象的属性。
commandText	字符串	对于 class 值为 MAIN 的事件，此字段表示用户传入的实际 SQL 语句。此字段用于除连接或断开连接记录之外的所有类型的记录，在这种情况下，该值为 null。 对于 class 值为 AUX 的事件，此字段包含有关事件中涉及的对象的补充信息。 对于 Aurora MySQL，字符（如引号）前面有反斜杠，表示转义字符。 **Important** 每个语句的完整 SQL 文本在审核日志中可见，包括任何敏感数据。但是，如果 Aurora 可以从上下文中确定数据库用户密码，则会对该密码进行修订，如下面的 SQL 语句所示。 `mysql> SET PASSWORD = 'my-password ';` **Note** 作为安全最佳实践，请指定除此处所示提示以外的密码。
databaseName	字符串	用户连接到的数据库。

字段	数据类型	描述
dbProtocol	字符串	数据库协议。目前，对于 Aurora MySQL，此值始终为 MySQL。
dbUserName	字符串	客户端对其进行身份验证的数据库用户。
endTime （仅版本 1.2 数据库活动记录）	字符串	SQL 语句执行结束的时间。它以协调世界时 (UTC) 格式表示。 要计算 SQL 语句的执行时间，请使用 endTime - startTime 。另请参阅 startTime 字段。
errorMessage （仅版本 1.1 数据库活动记录）	字符串	如果出现任何错误，则使用数据库服务器生成的错误消息填充此字段。对于未导致错误的普通语句，errorMessage 值为 null。 错误定义为生成客户端可见 MySQL 错误日志事件（其严重性级别为 ERROR 或更高）的任何活动。有关更多信息，请参阅 MySQL 参考手册中的错误日志。例如，语法错误和查询取消会生成错误消息。 内部 MySQL 服务器错误（例如后台检查指针进程错误）不会生成错误消息。但是，无论如何设置日志严重性级别，此类事件的记录仍会发出。这样可防止攻击者关闭日志记录以尝试避开检测。 另请参阅 exitCode 字段。
exitCode	int	用于会话退出记录的值。在干净的出口，这包含退出代码。在某些故障场景中，无法始终获得退出代码。在此类情况下，此值可能为零，也可能为空。
logTime	字符串	审核代码路径中记录的时间戳。它以协调世界时 (UTC) 格式表示。有关计算语句持续时间的最准确方法，请参阅 startTime 和 endTime 字段。
netProtocol	字符串	网络通信协议。目前，对于 Aurora MySQL，此值始终为 TCP。

字段	数据类型	描述
objectName	字符串	数据库对象的名称（如果正在对一个数据库对象运行 SQL 语句）。此字段仅在对数据库对象运行 SQL 语句时使用。如果未对一个对象运行 SQL 语句，则此值为空。要构造对象的完全限定名称，请将 databaseName 和 objectName 组合起来。如果查询涉及多个对象，则此字段可以是名称的逗号分隔列表。
objectType	字符串	数据库对象类型，例如表、索引等。此字段仅在对数据库对象运行 SQL 语句时使用。如果未对一个对象运行 SQL 语句，则此值为 null。 Aurora MySQL 的有效值包括： • INDEX • TABLE • UNKNOWN
paramList	字符串	此字段不用于 Aurora MySQL 且始终为 null。
pid	int	为向客户端连接提供服务而分配的后端进程的进程 ID。当数据库服务器重新启动时，pid 会发生更改，并且 statement Id 字段的计数器会重新开始。
remoteHost	字符串	发出 SQL 语句的客户端的 IP 地址或主机名。对于 Aurora MySQL，使用哪一个取决于数据库的 skip_name _resolve 参数设置。值 localhost 指示来自 rdsadmin 特殊用户的活动。
remotePort	字符串	客户端的端口号。
rowCount	int	SQL 语句返回的行数。例如，如果 SELECT 语句返回 10 行，则 rowCount 为 10。对于 INSERT 或 UPDATE 语句，则 rowCount 为 0。
serverHost	字符串	数据库服务器实例标识符。
serverType	字符串	数据库服务器类型，例如 MySQL。

字段	数据类型	描述
serverVersion	字符串	数据库服务器版本。目前，对于 Aurora MySQL，此值始终为 MySQL 5.7.12。
serviceName	字符串	服务的名称。目前，对于 Aurora MySQL，此值始终为 Amazon Aurora MySQL。
sessionId	int	伪唯一会话标识符。
startTime （仅版本 1.1 数据库活动记录）	字符串	SQL 语句开始执行的时间。它以协调世界时 (UTC) 格式表示。 要计算 SQL 语句的执行时间，请使用 endTime - startTime 。另请参阅 endTime 字段。
statementId	int	客户端的 SQL 语句的标识符。计数器随客户端输入的每个 SQL 语句递增。在重新启动数据库实例时，将重置计数器。
substatementId	int	SQL 子语句的标识符。对于类为 MAIN 的事件，此值为 1；对于类为 AUX 的事件，此值为 2。使用 statementId 字段标识同一语句生成的所有事件。
transactionId （仅版本 1.2 数据库活动记录）	int	事务的标识符。
type	字符串	事件类型。有效值为 record 或 heartbeat 。

使用 AWS 开发工具包处理数据库活动流

您可以使用 AWS 开发工具包以编程方式处理活动流。以下功能完善的 Java 和 Python 示例是关于可能会如何处理 Kinesis 数据流。

Java

```
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
```

```
import java.io.IOException;
import java.net.InetAddress;
import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.security.NoSuchAlgorithmException;
import java.security.NoSuchProviderException;
import java.security.Security;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.UUID;
import java.util.zip.GZIPInputStream;

import javax.crypto.Cipher;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.spec.SecretKeySpec;

import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.encryptionsdk.AwsCrypto;
import com.amazonaws.encryptionsdk.CryptoInputStream;
import com.amazonaws.encryptionsdk.jce.JceMasterKey;
import
 com.amazonaws.services.kinesis.clientlibrary.exceptions.InvalidStateException;
import com.amazonaws.services.kinesis.clientlibrary.exceptions.ShutdownException;
import com.amazonaws.services.kinesis.clientlibrary.exceptions.ThrottlingException;
import com.amazonaws.services.kinesis.clientlibrary.interfaces.IRecordProcessor;
import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.IRecordProcessorCheckpointer;
import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.IRecordProcessorFactory;
import
 com.amazonaws.services.kinesis.clientlibrary.lib.worker.InitialPositionInStream;
import
 com.amazonaws.services.kinesis.clientlibrary.lib.worker.KinesisClientLibConfiguration;
import com.amazonaws.services.kinesis.clientlibrary.lib.worker.ShutdownReason;
import com.amazonaws.services.kinesis.clientlibrary.lib.worker.Worker;
import com.amazonaws.services.kinesis.clientlibrary.lib.worker.Worker.Builder;
import com.amazonaws.services.kinesis.model.Record;
import com.amazonaws.services.kms.AWSKMS;
import com.amazonaws.services.kms.AWSKMSClientBuilder;
import com.amazonaws.services.kms.model.DecryptRequest;
import com.amazonaws.services.kms.model.DecryptResult;
import com.amazonaws.util.Base64;
```

```
import com.amazonaws.util.IOUtils;
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import com.google.gson.annotations.SerializedName;
import org.bouncycastle.jce.provider.BouncyCastleProvider;

public class DemoConsumer {

    private static final String STREAM_NAME = "aws-rds-das-[cluster-external-
resource-id]";
    private static final String APPLICATION_NAME = "AnyApplication"; //unique
 application name for dynamo table generation that holds kinesis shard tracking
    private static final String AWS_ACCESS_KEY =
 "[AWS_ACCESS_KEY_TO_ACCESS_KINESIS]";
    private static final String AWS_SECRET_KEY =
 "[AWS_SECRET_KEY_TO_ACCESS_KINESIS]";
    private static final String DBC_RESOURCE_ID = "[cluster-external-resource-id]";
    private static final String REGION_NAME = "[region-name]"; //us-east-1, us-
east-2...
    private static final BasicAWSCredentials CREDENTIALS = new
 BasicAWSCredentials(AWS_ACCESS_KEY, AWS_SECRET_KEY);
    private static final AWSStaticCredentialsProvider CREDENTIALS_PROVIDER = new
 AWSStaticCredentialsProvider(CREDENTIALS);

    private static final AwsCrypto CRYPTO = new AwsCrypto();
    private static final AWSKMS KMS = AWSKMSClientBuilder.standard()
            .withRegion(REGION_NAME)
            .withCredentials(CREDENTIALS_PROVIDER).build();

    class Activity {
        String type;
        String version;
        String databaseActivityEvents;
        String key;
    }

    class ActivityEvent {
        @SerializedName("class") String _class;
        String clientApplication;
        String command;
        String commandText;
        String databaseName;
        String dbProtocol;
        String dbUserName;
```

```
        String endTime;
        String errorMessage;
        String exitCode;
        String logTime;
        String netProtocol;
        String objectName;
        String objectType;
        List<String> paramList;
        String pid;
        String remoteHost;
        String remotePort;
        String rowCount;
        String serverHost;
        String serverType;
        String serverVersion;
        String serviceName;
        String sessionId;
        String startTime;
        String statementId;
        String substatementId;
        String transactionId;
        String type;
    }

    class ActivityRecords {
        String type;
        String clusterId;
        String instanceId;
        List<ActivityEvent> databaseActivityEventList;
    }

    static class RecordProcessorFactory implements IRecordProcessorFactory {
        @Override
        public IRecordProcessor createProcessor() {
            return new RecordProcessor();
        }
    }

    static class RecordProcessor implements IRecordProcessor {

        private static final long BACKOFF_TIME_IN_MILLIS = 3000L;
        private static final int PROCESSING_RETRIES_MAX = 10;
        private static final long CHECKPOINT_INTERVAL_MILLIS = 60000L;
```

```
        private static final Gson GSON = new
GsonBuilder().serializeNulls().create();

        private static final Cipher CIPHER;
        static {
            Security.insertProviderAt(new BouncyCastleProvider(), 1);
            try {
                CIPHER = Cipher.getInstance("AES/GCM/NoPadding", "BC");
            } catch (NoSuchAlgorithmException | NoSuchPaddingException |
NoSuchProviderException e) {
                throw new ExceptionInInitializerError(e);
            }
        }

        private long nextCheckpointTimeInMillis;

        @Override
        public void initialize(String shardId) {
        }

        @Override
        public void processRecords(final List<Record> records, final
IRecordProcessorCheckpointer checkpointer) {
            for (final Record record : records) {
                processSingleBlob(record.getData());
            }

            if (System.currentTimeMillis() > nextCheckpointTimeInMillis) {
                checkpoint(checkpointer);
                nextCheckpointTimeInMillis = System.currentTimeMillis() +
CHECKPOINT_INTERVAL_MILLIS;
            }
        }

        @Override
        public void shutdown(IRecordProcessorCheckpointer checkpointer,
ShutdownReason reason) {
            if (reason == ShutdownReason.TERMINATE) {
                checkpoint(checkpointer);
            }
        }

        private void processSingleBlob(final ByteBuffer bytes) {
            try {
```

```
                // JSON $Activity
                final Activity activity = GSON.fromJson(new String(bytes.array(),
StandardCharsets.UTF_8), Activity.class);

                // Base64.Decode
                final byte[] decoded =
Base64.decode(activity.databaseActivityEvents);
                final byte[] decodedDataKey = Base64.decode(activity.key);

                Map<String, String> context = new HashMap<>();
                context.put("aws:rds:dbc-id", DBC_RESOURCE_ID);

                // Decrypt
                final DecryptRequest decryptRequest = new DecryptRequest()

.withCiphertextBlob(ByteBuffer.wrap(decodedDataKey)).withEncryptionContext(context);
                final DecryptResult decryptResult = KMS.decrypt(decryptRequest);
                final byte[] decrypted = decrypt(decoded,
getByteArray(decryptResult.getPlaintext()));

                // GZip Decompress
                final byte[] decompressed = decompress(decrypted);
                // JSON $ActivityRecords
                final ActivityRecords activityRecords = GSON.fromJson(new
String(decompressed, StandardCharsets.UTF_8), ActivityRecords.class);

                // Iterate throught $ActivityEvents
                for (final ActivityEvent event :
activityRecords.databaseActivityEventList) {
                    System.out.println(GSON.toJson(event));
                }
            } catch (Exception e) {
                // Handle error.
                e.printStackTrace();
            }
        }

        private static byte[] decompress(final byte[] src) throws IOException {
            ByteArrayInputStream byteArrayInputStream = new
ByteArrayInputStream(src);
            GZIPInputStream gzipInputStream = new
GZIPInputStream(byteArrayInputStream);
            return IOUtils.toByteArray(gzipInputStream);
        }
```

```
       private void checkpoint(IRecordProcessorCheckpointer checkpointer) {
           for (int i = 0; i < PROCESSING_RETRIES_MAX; i++) {
               try {
                   checkpointer.checkpoint();
                   break;
               } catch (ShutdownException se) {
                   // Ignore checkpoint if the processor instance has been shutdown
(fail over).
                   System.out.println("Caught shutdown exception, skipping
checkpoint." + se);
                   break;
               } catch (ThrottlingException e) {
                   // Backoff and re-attempt checkpoint upon transient failures
                   if (i >= (PROCESSING_RETRIES_MAX - 1)) {
                       System.out.println("Checkpoint failed after " + (i + 1) +
"attempts." + e);
                       break;
                   } else {
                       System.out.println("Transient issue when checkpointing -
attempt " + (i + 1) + " of " + PROCESSING_RETRIES_MAX + e);
                   }
               } catch (InvalidStateException e) {
                   // This indicates an issue with the DynamoDB table (check for
table, provisioned IOPS).
                   System.out.println("Cannot save checkpoint to the DynamoDB table
used by the Amazon Kinesis Client Library." + e);
                   break;
               }
               try {
                   Thread.sleep(BACKOFF_TIME_IN_MILLIS);
               } catch (InterruptedException e) {
                   System.out.println("Interrupted sleep" + e);
               }
           }
       }
   }

   private static byte[] decrypt(final byte[] decoded, final byte[] decodedDataKey)
throws IOException {
       // Create a JCE master key provider using the random key and an AES-GCM
encryption algorithm
       final JceMasterKey masterKey = JceMasterKey.getInstance(new
SecretKeySpec(decodedDataKey, "AES"),
```

```
                "BC", "DataKey", "AES/GCM/NoPadding");
        try (final CryptoInputStream<JceMasterKey> decryptingStream =
 CRYPTO.createDecryptingStream(masterKey, new ByteArrayInputStream(decoded));
             final ByteArrayOutputStream out = new ByteArrayOutputStream()) {
            IOUtils.copy(decryptingStream, out);
            return out.toByteArray();
        }
    }

    public static void main(String[] args) throws Exception {
        final String workerId = InetAddress.getLocalHost().getCanonicalHostName() +
 ":" + UUID.randomUUID();
        final KinesisClientLibConfiguration kinesisClientLibConfiguration =
                new KinesisClientLibConfiguration(APPLICATION_NAME, STREAM_NAME,
 CREDENTIALS_PROVIDER, workerId);

 kinesisClientLibConfiguration.withInitialPositionInStream(InitialPositionInStream.LATEST);
        kinesisClientLibConfiguration.withRegionName(REGION_NAME);
        final Worker worker = new Builder()
                .recordProcessorFactory(new RecordProcessorFactory())
                .config(kinesisClientLibConfiguration)
                .build();

        System.out.printf("Running %s to process stream %s as worker %s...\n",
 APPLICATION_NAME, STREAM_NAME, workerId);

        try {
            worker.run();
        } catch (Throwable t) {
            System.err.println("Caught throwable while processing data.");
            t.printStackTrace();
            System.exit(1);
        }
        System.exit(0);
    }

    private static byte[] getByteArray(final ByteBuffer b) {
        byte[] byteArray = new byte[b.remaining()];
        b.get(byteArray);
        return byteArray;
    }
}
```

Python

```
import base64
import json
import zlib
import aws_encryption_sdk
from aws_encryption_sdk import CommitmentPolicy
from aws_encryption_sdk.internal.crypto import WrappingKey
from aws_encryption_sdk.key_providers.raw import RawMasterKeyProvider
from aws_encryption_sdk.identifiers import WrappingAlgorithm, EncryptionKeyType
import boto3

REGION_NAME = '<region>'                    # us-east-1
RESOURCE_ID = '<external-resource-id>'      # cluster-ABCD123456
STREAM_NAME = 'aws-rds-das-' + RESOURCE_ID  # aws-rds-das-cluster-ABCD123456

enc_client =
 aws_encryption_sdk.EncryptionSDKClient(commitment_policy=CommitmentPolicy.FORBID_ENCRYPT_AL

class MyRawMasterKeyProvider(RawMasterKeyProvider):
    provider_id = "BC"

    def __new__(cls, *args, **kwargs):
        obj = super(RawMasterKeyProvider, cls).__new__(cls)
        return obj

    def __init__(self, plain_key):
        RawMasterKeyProvider.__init__(self)
        self.wrapping_key =
 WrappingKey(wrapping_algorithm=WrappingAlgorithm.AES_256_GCM_IV12_TAG16_NO_PADDING,
                                        wrapping_key=plain_key,
 wrapping_key_type=EncryptionKeyType.SYMMETRIC)

    def _get_raw_key(self, key_id):
        return self.wrapping_key


def decrypt_payload(payload, data_key):
    my_key_provider = MyRawMasterKeyProvider(data_key)
    my_key_provider.add_master_key("DataKey")
    decrypted_plaintext, header = enc_client.decrypt(
        source=payload,

 materials_manager=aws_encryption_sdk.materials_managers.default.DefaultCryptoMaterialsManag
```

```
    return decrypted_plaintext


def decrypt_decompress(payload, key):
    decrypted = decrypt_payload(payload, key)
    return zlib.decompress(decrypted, zlib.MAX_WBITS + 16)


def main():
    session = boto3.session.Session()
    kms = session.client('kms', region_name=REGION_NAME)
    kinesis = session.client('kinesis', region_name=REGION_NAME)

    response = kinesis.describe_stream(StreamName=STREAM_NAME)
    shard_iters = []
    for shard in response['StreamDescription']['Shards']:
        shard_iter_response = kinesis.get_shard_iterator(StreamName=STREAM_NAME,
 ShardId=shard['ShardId'],

 ShardIteratorType='LATEST')
        shard_iters.append(shard_iter_response['ShardIterator'])

    while len(shard_iters) > 0:
        next_shard_iters = []
        for shard_iter in shard_iters:
            response = kinesis.get_records(ShardIterator=shard_iter, Limit=10000)
            for record in response['Records']:
                record_data = record['Data']
                record_data = json.loads(record_data)
                payload_decoded =
 base64.b64decode(record_data['databaseActivityEvents'])
                data_key_decoded = base64.b64decode(record_data['key'])
                data_key_decrypt_result =
 kms.decrypt(CiphertextBlob=data_key_decoded,

 EncryptionContext={'aws:rds:dbc-id': RESOURCE_ID})
                print (decrypt_decompress(payload_decoded,
 data_key_decrypt_result['Plaintext']))
            if 'NextShardIterator' in response:
                next_shard_iters.append(response['NextShardIterator'])
        shard_iters = next_shard_iters


if __name__ == '__main__':
```

```
    main()
```

数据库活动流的 IAM 策略示例

具有数据库活动流的相应 AWS Identity and Access Management（IAM）角色权限的任何用户均可以创建、启动、停止和修改数据库集群的活动流设置。这些操作包含在流的审核日志中。对于最佳合规性实践，我们建议您不要向 DBA 提供这些权限。

您可以使用 IAM 策略设置对数据库活动流的访问权限。有关 Aurora 身份验证的更多信息，请参阅Amazon Aurora 的 Identity and Access Management。有关创建 IAM 策略的更多信息，请参阅 创建和使用适用于 IAM 数据库访问的 IAM 策略。

Example 允许配置数据库活动流的策略

要为用户提供精细访问权限以修改活动流，请在 IAM 策略中使用服务特定的操作上下文密钥 `rds:StartActivityStream` 和 `rds:StopActivityStream`。以下 IAM 策略示例允许用户或角色配置活动流。

```
{
   "Version":"2012-10-17",
   "Statement":[
        {
            "Sid":"ConfigureActivityStreams",
            "Effect":"Allow",
            "Action": [
                "rds:StartActivityStream",
                "rds:StopActivityStream"
            ],
            "Resource":"*",
          }
     ]
}
```

Example 允许启动数据库活动流的策略

以下 IAM 策略示例允许用户或角色启动活动流。

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
```

```
            "Sid":"AllowStartActivityStreams",
            "Effect":"Allow",
            "Action":"rds:StartActivityStream",
            "Resource":"*"
        }
    ]
}
```

Example 允许停止数据库活动流的策略

以下 IAM 策略示例允许用户或角色停止活动流。

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"AllowStopActivityStreams",
            "Effect":"Allow",
            "Action":"rds:StopActivityStream",
            "Resource":"*"
        }
     ]
}
```

Example 拒绝启动数据库活动流的策略

以下 IAM 策略示例阻止用户或角色启动活动流。

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"DenyStartActivityStreams",
            "Effect":"Deny",
            "Action":"rds:StartActivityStream",
            "Resource":"*"
        }
     ]
}
```

Example 拒绝停止数据库活动流的策略

以下 IAM 策略示例阻止用户或角色停止活动流。

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"DenyStopActivityStreams",
            "Effect":"Deny",
            "Action":"rds:StopActivityStream",
            "Resource":"*"
        }
    ]
}
```

使用适用于 Amazon Aurora 的 Amazon GuardDuty RDS 保护功能监控威胁

Amazon GuardDuty 是一项威胁检测服务，可帮助保护您的账户、容器、工作负载和 AWS 环境中的数据。GuardDuty 使用机器学习（ML）模型以及异常和威胁检测功能，持续监控不同的日志源和运行时活动，以识别环境中的潜在安全风险和恶意活动并确定其优先级。

GuardDuty RDS 保护功能会分析和剖析登录事件，找出对 Amazon Aurora 数据库的潜在访问威胁。当您开启 RDS 保护功能时，GuardDuty 会监控来自 Aurora 数据库的 RDS 登录事件。RDS 保护功能会监控这些事件并对其进行剖析，以了解潜在的内部威胁或外部行为者。

有关启用 GuardDuty RDS 保护功能的更多信息，请参阅《Amazon GuardDuty 用户指南》中的 GuardDuty RDS 保护。

当 RDS 保护功能检测到潜在威胁（例如成功或失败的登录尝试中的异常模式）时，GuardDuty 会生成新的调查结果，其中包含有关可能受损数据库的详细信息。您可以在 Amazon GuardDuty 控制台的调查发现摘要部分查看调查发现的详细信息。调查发现详细信息因调查发现类型而异。主要详细信息、资源类型和资源角色决定了任何调查发现中可用的信息类型。有关调查发现的常见详细信息和调查发现类型的更多信息，请分别参阅《Amazon GuardDuty 用户指南》中的调查发现详细信息和 GuardDuty RDS 保护调查发现类型。

您可以在提供 RDS 保护功能的任何 AWS 区域，为任何 AWS 账户开启或关闭此功能。未启用 RDS 保护功能时，GuardDuty 不会检测可能受损的 Aurora 数据库，也不会提供受损的详细信息。

现有 GuardDuty 账户可以启用 RDS 保护功能，试用期为 30 天。对于新的 GuardDuty 账户，RDS 保护功能已启用，并包含在 30 天免费试用期中。有关更多信息，请参阅《Amazon GuardDuty 用户指南》中的估计 GuardDuty 成本。

有关 GuardDuty 尚不支持 RDS 保护功能的 AWS 区域的信息，请参阅《Amazon GuardDuty User Guide》中的 Region-specific feature availability。

有关 GuardDuty RDS 保护支持的 Aurora 数据库版本的信息，请参阅《Amazon GuardDuty User Guide》中的 Supported Amazon Aurora, Amazon RDS, and Aurora Limitless databases。

使用 Amazon Aurora MySQL

Amazon Aurora MySQL 是一个完全托管式、兼容 MySQL 的关系数据库引擎，结合了高端商用数据库的速度和可靠性，同时还具有开源数据库的简单性和成本效益。Aurora MySQL 是 MySQL 的直接替代品，可以让您通过简单且经济高效的方式设置、运行和扩展新的和现有的 MySQL 部署，以使您腾出时间专注于业务和应用程序。Amazon RDS 能够掌管常规数据库任务（例如预置、补丁、备份、恢复、故障检测与修复），从而对 Aurora 进行管理。Amazon RDS 还提供一键迁移工具，帮助您将现有的 Amazon RDS for MySQL 应用程序转换至 Aurora MySQL。

主题

- Amazon Aurora MySQL 概述
- 使用 Amazon Aurora MySQL 实现高安全性
- 使用新的 TLS 证书更新应用程序，以连接到 Aurora MySQL 数据库集群
- 对 Aurora MySQL 使用 Kerberos 身份验证
- 将数据迁移到 Amazon Aurora MySQL 数据库集群
- 管理 Amazon Aurora MySQL
- 优化 Aurora MySQL
- Amazon Aurora MySQL 的并行查询
- 在 Amazon Aurora MySQL 数据库集群中使用高级审计
- 使用 Amazon Aurora MySQL 进行复制
- 在 Amazon Aurora MySQL 数据库集群中使用本地写入转发
- 将 Amazon Aurora MySQL 与其他AWS服务集成
- Amazon Aurora MySQL 实验室模式
- Amazon Aurora MySQL 的最佳实践
- 排除 Amazon Aurora MySQL 数据库性能故障
- Amazon Aurora MySQL 参考
- Amazon Aurora MySQL 的数据库引擎更新

Amazon Aurora MySQL 概述

以下几节提供了 Amazon Aurora MySQL 概述。

主题

- Amazon Aurora MySQL 性能增强
- Amazon Aurora MySQL 和空间数据
- 与 MySQL 8.0 兼容的 Aurora MySQL 版本 3
- 与 MySQL 5.7 兼容的 Aurora MySQL 版本 2

Amazon Aurora MySQL 性能增强

Amazon Aurora 包括用于支持高端商用数据库的不同需求的性能增强。

快速插入

快速插入加速了按主键排序的并行插入，特别适用于 `LOAD DATA` 和 `INSERT INTO ... SELECT ...` 语句。在执行语句时，快速插入将光标的位置缓存到索引遍历中。这可避免再次不必要地遍历索引。

在 Aurora MySQL 3.03.2 及更高版本中，仅对常规 InnoDB 表启用快速插入。这种优化不适用于 InnoDB 临时表。在所有 2.11 和 2.12 版本的 Aurora MySQL 版本 2 中禁用了该功能。仅在禁用自适应哈希索引优化功能时，快速插入优化才起作用。

您可监控下列指标来确定数据库集群的快速插入的有效性：

- `aurora_fast_insert_cache_hits`：在成功检索和验证缓存光标时递增的计数器。
- `aurora_fast_insert_cache_misses`：当缓存光标不再有效且 Aurora 执行常规索引遍历时递增的计数器。

您可以使用以下命令检索快速插入指标的当前值：

```
mysql> show global status like 'Aurora_fast_insert%';
```

您将获得与下内容类似的输出：

```
+---------------------------------+-----------+
| Variable_name                   | Value     |
+---------------------------------+-----------+
| Aurora_fast_insert_cache_hits   | 3598300   |
```

```
| Aurora_fast_insert_cache_misses | 436401336 |
+---------------------------------+-----------+
```

Amazon Aurora MySQL 和空间数据

下表汇总了主 Aurora MySQL 空间功能并说明了它们如何与 MySQL 中的空间功能对应：

- Aurora MySQL 版本 2 与 MySQL 5.7 支持相同的空间数据类型和空间关系函数。有关这些数据类型和函数的更多信息，请参阅 MySQL 5.7 文档中的空间数据类型和空间关系函数。
- Aurora MySQL 版本 3 与 MySQL 8.0 支持相同的空间数据类型和空间关系函数。有关这些数据类型和函数的更多信息，请参阅 MySQL 8.0 文档中的空间数据类型和空间关系函数。
- Aurora MySQL 支持 InnoDB 表上的空间索引。空间索引将提高空间数据查询在大型数据集上的查询性能。在 MySQL 中，InnoDB 表的空间索引可用于 MySQL 5.7 和 8.0 中。

  Aurora MySQL 使用与 MySQL 不同的空间索引策略，以便让空间查询实现高性能。Aurora 空间索引实施在 B 树上使用空间填充曲线，旨在提供比 R 树更高的空间范围扫描性能。

  > **Note**
  >
  > 在 Aurora MySQL 中，在具有空间参考标识符（SRID）的列上定义空间索引的表上的事务无法插入到其他事务选择进行更新的区域。

以下数据定义语言 (DDL) 语句可用于在使用空间数据类型的列上创建索引。

CREATE TABLE

您可以在 `SPATIAL INDEX` 语句中使用 `CREATE TABLE` 关键字来向新表中的列添加空间索引。以下是一个示例。

```
CREATE TABLE test (shape POLYGON NOT NULL, SPATIAL INDEX(shape));
```

ALTER TABLE

您可以在 `SPATIAL INDEX` 语句中使用 `ALTER TABLE` 关键字向现有表中的列添加空间索引。以下是一个示例。

```
ALTER TABLE test ADD SPATIAL INDEX(shape);
```

CREATE INDEX

您可以在 SPATIAL 语句中使用 CREATE INDEX 关键字向现有表中的列添加空间索引。以下为示例。

```
CREATE SPATIAL INDEX shape_index ON test (shape);
```

与 MySQL 8.0 兼容的 Aurora MySQL 版本 3

您可以使用 Aurora MySQL 版本 3 来获得最新的 MySQL 兼容功能、性能增强功能和错误修复。接下来，您可以了解与 MySQL 8.0 兼容的 Aurora MySQL 版本 3。您可以了解如何将集群和应用程序升级到 Aurora MySQL 版本 3。

一些 Aurora 功能（例如 Aurora Serverless v2）需要 Aurora MySQL 版本 3。

主题

- MySQL 8.0 社群版中的功能
- Aurora MySQL 版本 3 是 Aurora MySQL Serverless v2 的先决条件
- Aurora MySQL 版本 3 的发布说明
- 新的并行查询优化
- 进行优化以缩短数据库重启时间
- Aurora MySQL 版本 3 中的新临时表行为
- 比较 Aurora MySQL 版本 2 和 Aurora MySQL 版本 3
- 比较 Aurora MySQL 版本 3 和 MySQL 8.0 社群版
- 升级到 Aurora MySQL 版本 3

MySQL 8.0 社群版中的功能

Aurora MySQL 版本 3 的初始版本与 MySQL 8.0.23 社群版兼容。MySQL 8.0 引入了几项新功能，包括以下功能：

- 原子数据定义语言（DDL）支持。有关更多信息，请参阅 原子数据定义语言（DDL）支持。
- JSON 函数。有关使用信息，请参阅 MySQL 参考手册中的 JSON 函数。
- 窗口函数。有关使用信息，请参阅 MySQL 参考手册中的 Window 函数。
- 使用 WITH 子句的公用表表达式 (CTE)。有关使用信息，请参阅 MySQL 参考手册中的 WITH（公用表表达式）。

- `ALTER TABLE` 语句的优化 `ADD COLUMN` 和 `RENAME COLUMN` 子句。这些优化被称为“即时 DDL”。Aurora MySQL 版本 3 与社群 MySQL 即时 DDL 功能兼容。未使用以前的 Aurora 快速 DDL 功能。有关即时 DDL 的使用信息，请参阅 即时 DDL（Aurora MySQL 版本 3）。
- 降序、功能性和不可见的索引。有关使用信息，请参阅 MySQL 参考手册中的不可见索引、降序索引和 CREATE INDEX 语句。
- 通过 SQL 语句控制的基于角色的权限。有关更改权限模型的更多信息，请参阅 基于角色的权限模型。
- 带有 `SELECT ... FOR SHARE` 语句的 `NOWAIT` 和 `SKIP LOCKED` 子句。这些子句避免了等待其他事务释放行锁。有关使用信息，请参阅 MySQL 参考手册中的锁定读取。
- 改进二进制日志 (binlog) 复制。有关 Aurora MySQL 的详细信息，请参阅 二进制日志复制。特别是，您可以执行筛选的复制。有关筛选复制的使用信息，请参阅 MySQL 参考手册中的服务器如何评估复制筛选规则。
- 提示。一些 MySQL 8.0 兼容的提示已被向后移植到 Aurora MySQL 版本 2。有关将提示用于 Aurora MySQL 的信息，请参阅 Aurora MySQL 提示。有关社群 MySQL 8.0 中的完整提示列表，请参阅 MySQL 参考手册中的优化程序提示。

有关添加到 MySQL 8.0 社群版的功能的完整列表，请参阅博客文章 MySQL 8.0 中的完整新功能列表。

Aurora MySQL 版本 3 还包括对包容性语言的关键字的更改，从社群 MySQL 8.0.26 向后移植。有关更改的详细信息，请参阅 Aurora MySQL 版本 3 的包容性语言更改。

Aurora MySQL 版本 3 是 Aurora MySQL Serverless v2 的先决条件

Aurora MySQL 版本 3 是 Aurora MySQL Serverless v2 集群中所有数据库实例的先决条件。Aurora MySQL Serverless v2 包括对数据库集群中的读取器实例的支持，并包括对 Aurora MySQL Serverless v1 不可用的其他 Aurora 功能。与 Aurora MySQL Serverless v1 相比，它还具有更快、更细粒度的扩展。

Aurora MySQL 版本 3 的发布说明

有关所有 Aurora MySQL 版本 3 发布的发布说明，请参阅《Aurora MySQL 发布说明》中的 Amazon Aurora MySQL 版本 3 的数据库引擎更新。

新的并行查询优化

Aurora 并行查询优化现在适用于更多的 SQL 操作：

- 并行查询现在适用于包含数据类型 TEXT、BLOB、JSON、GEOMETRY 和 VARCHAR 以及超过 768 个字节的 CHAR 的表。
- 并行查询可以优化涉及分区表的查询。
- 并行查询可以优化涉及选择列表中聚合函数调用的查询和 HAVING 子句。

有关这些增强功能的更多信息，请参阅 将并行查询集群升级到 Aurora MySQL 版本 3。有关 Aurora 并行查询的一般信息，请参阅 Amazon Aurora MySQL 的并行查询。

进行优化以缩短数据库重启时间

在计划内和计划外停机期间，您的 Aurora MySQL 数据库集群都必须具有高可用性。

数据库管理员需要偶尔进行数据库维护。此维护包括数据库修补、升级、需要手动重启的数据库参数修改、执行故障转移以缩短实例类更改所花费的时间等。这些计划内操作需要停机。

但是，停机也可能由计划外操作引起，例如由于底层硬件故障或数据库资源限制而导致的意外故障转移。所有这些计划内和计划外操作都会导致数据库重启。

在 Aurora MySQL 3.05 及更高版本中，我们引入了缩短数据库重启时间的优化措施。与未进行优化相比，这些优化措施可减少多达 65% 的停机时间，并且在重启后数据库工作负载的中断也会减少。

在数据库启动期间，会初始化许多内部内存组件。其中最大的是 InnoDB 缓冲池，在 Aurora MySQL 中，该缓冲池默认为实例内存大小的 75%。我们经测试发现，初始化时间与 InnoDB 缓冲池的大小成正比，因此会随着数据库实例类的大小而扩展。在此初始化阶段，数据库无法接受连接，这会导致重启期间的停机时间更长。Aurora MySQL 快速重启的第一阶段优化了缓冲池初始化，这将会缩短数据库初始化的时间，从而缩短总体重启时间。

有关更多详细信息，请参阅博客借助 Amazon Aurora MySQL 数据库重启时间优化减少停机时间。

Aurora MySQL 版本 3 中的新临时表行为

Aurora MySQL 版本 3 处理临时表的方式与早期的 Aurora MySQL 版本不同。这种新行为继承自 MySQL 8.0 社群版。使用 Aurora MySQL 版本 3 可以创建两种类型的临时表：

- 内部（或隐式）临时表 – 由 Aurora MySQL 引擎创建，以处理诸如对聚合、派生表或公用表表达式 (CTE) 进行排序等操作。
- 用户创建的（或显式）临时表 – 使用 CREATE TEMPORARY TABLE 语句时由 Aurora MySQL 引擎创建。

对于 Aurora 读取器数据库实例上的内部和用户创建的临时表，还有其他注意事项。我们将在以下各节中讨论这些变化。

主题

- 内部（隐式）临时表的存储引擎
- 限制内部内存中临时表的大小
- 缓解 Aurora 副本上内部临时表的完整性问题
- 优化 Aurora MySQL 数据库实例上的 temptable_max_mmap 参数
- 读取器数据库实例上用户创建的（显式）临时表
- 临时表创建错误和缓解

内部（隐式）临时表的存储引擎

在生成中间结果集时，Aurora MySQL 最初尝试写入内存中临时表。这可能不成功，原因是数据类型不兼容或配置了限制。如果是这样，临时表将转换为磁盘上的临时表，而不是保留在内存中。有关这方面的更多信息，请参阅 MySQL 文档中的在 MySQL 中使用内部临时表。

在 Aurora MySQL 版本 3 中，内部临时表的工作方式与早期的 Aurora MySQL 版本不同。现在，您不必针对此类临时表在 InnoDB 与 MyISAM 存储引擎之间进行选择，而是在 `TempTable` 与 `MEMORY` 存储引擎之间进行选择。

使用 `TempTable` 存储引擎，您可以另外选择如何处理某些数据。受影响的数据溢出了保存数据库实例的所有内部临时表的内存池。

这些选择可能会影响生成大量临时数据的查询的性能，例如，在大型表上执行诸如 `GROUP BY` 之类的聚合操作。

> **Tip**
>
> 如果您的工作负载包括生成内部临时表的查询，请通过运行基准测试和监控与性能相关的指标来确认应用程序如何执行此更改。
>
> 在某些情况下，临时数据量适合 `TempTable` 内存池或者只有少量溢出内存池。在这些情况下，我们建议将 `TempTable` 设置用于内部临时表和内存映射文件以保存任何溢出数据。此设置是原定设置。

`TempTable` 存储引擎为默认引擎。`TempTable` 对使用此引擎的所有临时表使用公用内存池，而不是每个表的最大内存限制。此内存池的大小由 temptable_max_ram 参数指定。对于具有 16 GiB 或更多

内存的数据库实例，其默认设置为 1 GiB；而对于内存小于 16 GiB 的数据库实例，其默认设置为 16 MB。内存池的大小会影响会话级别的内存消耗。

在某些情况下，当您使用 TempTable 存储引擎时，临时数据可能会超过内存池的大小。如果是这样，Aurora MySQL 会使用辅助机制存储溢出数据。

您可以设置 temptable_max_mmap 参数来选择数据是溢出到内存映射的临时文件，还是磁盘上的 InnoDB 内部临时表。这些溢出机制的不同数据格式和溢出标准可能会影响查询性能。他们通过影响写入磁盘的数据量和对磁盘存储吞吐量的需求来实现这一目标。

Aurora MySQL 版本 3 通过以下方式存储溢出数据：

- 在写入器数据库实例上，溢出到 InnoDB 内部临时表或内存映射临时文件的数据驻留在实例上的本地存储中。
- 在读取器数据库实例上，溢出数据始终驻留在本地存储上的内存映射临时文件中。

  只读实例无法在 Aurora 集群卷上存储任何数据。

与内部临时表相关的配置参数对集群中的写入器和读取器实例的应用方式不同：

- 在读取器实例上，Aurora MySQL 始终使用 TempTable 存储引擎。
- 对于写入器和读取器实例，temptable_max_mmap 的大小都默认设置为 1GiB，无论数据库实例内存大小如何。可以在写入器实例和读取器实例上调整此值。
- 将 temptable_max_mmap 设置为 0 可禁止在写入器实例上使用内存映射的临时文件。
- 您无法在读取器实例上将 temptable_max_mmap 设置为 0。

> **Note**
>
> 建议您不要使用 temptable_use_mmap 参数。它已被弃用，预计将在将来的 MySQL 版本中删除对它的支持。

限制内部内存中临时表的大小

如内部（隐式）临时表的存储引擎中所述，可以通过使用 temptable_max_ram 和 temptable_max_mmap 设置来全局控制临时表资源。

还可以使用 tmp_table_size 数据库参数，来限制任何单个内部内存中临时表的大小。此限制旨在防止各个查询消耗过量的全局临时表资源，这可能会影响需要这些资源的并发查询的性能。

tmp_table_size 参数定义了 Aurora MySQL 版本 3 中由 MEMORY 存储引擎创建的临时表的最大大小。

在 Aurora MySQL 版本 3.04 及更高版本中，tmp_table_size 还定义了当 aurora_tmptable_enable_per_table_limit 数据库参数设置为 ON 时，TempTable 存储引擎创建的临时表的最大大小。默认情况下，此行为处于禁用状态（OFF），这与 Aurora MySQL 版本 3.03 及更低版本中的行为相同。

- 当 aurora_tmptable_enable_per_table_limit 为 OFF 时，不考虑将 tmp_table_size 用于由 TempTable 存储引擎创建的内部内存中临时表。

  但是，全局 TempTable 资源限制仍然适用。当达到全局 TempTable 资源限制时，Aurora MySQL 会出现以下行为：

  - 写入器数据库实例 – Aurora MySQL 自动将内存中临时表转换为 InnoDB 磁盘上临时表。
  - 读取器数据库实例 - 查询结束时出现错误。

  ```
  ERROR 1114 (HY000): The table '/rdsdbdata/tmp/#sqlxx_xxx' is full
  ```

- 当 aurora_tmptable_enable_per_table_limit 为 ON 时，如果达到 tmp_table_size 限制，Aurora MySQL 会出现以下行为：

  - 写入器数据库实例 – Aurora MySQL 自动将内存中临时表转换为 InnoDB 磁盘上临时表。
  - 读取器数据库实例 - 查询结束时出现错误。

  ```
  ERROR 1114 (HY000): The table '/rdsdbdata/tmp/#sqlxx_xxx' is full
  ```

  全局 TempTable 资源限制和每个表的限制都适用于这种情况。

> **Note**
>
> 当 internal_tmp_mem_storage_engine 设置为 MEMORY 时，aurora_tmptable_enable_per_table_limit 参数不起作用。在这种情况下，内存中临时表的最大大小由 tmp_table_size 或 max_heap_table_size 值定义，以较小者为准。

以下示例显示了写入器和读取器数据库实例的 aurora_tmptable_enable_per_table_limit 参数的行为。

Example **aurora_tmptable_enable_per_table_limit** 设置为 **OFF** 的写入器数据库实例

内存中临时表未转换为 InnoDB 磁盘上临时表。

```
mysql> set aurora_tmptable_enable_per_table_limit=0;
Query OK, 0 rows affected (0.00 sec)

mysql> select
 @@innodb_read_only,@@aurora_version,@@aurora_tmptable_enable_per_table_limit,@@temptable_max_r
+-------------------+-----------------+-----------------------------------------
+---------------------+----------------------+
| @@innodb_read_only | @@aurora_version | @@aurora_tmptable_enable_per_table_limit |
 @@temptable_max_ram | @@temptable_max_mmap |
+-------------------+-----------------+-----------------------------------------
+---------------------+----------------------+
|                  0 | 3.04.0          |                                        0 |
       1073741824 |          1073741824 |
+-------------------+-----------------+-----------------------------------------
+---------------------+----------------------+
1 row in set (0.00 sec)

mysql> show status like '%created_tmp_disk%';
+-------------------------+-------+
| Variable_name           | Value |
+-------------------------+-------+
| Created_tmp_disk_tables | 0     |
+-------------------------+-------+
1 row in set (0.00 sec)

mysql> set cte_max_recursion_depth=4294967295;
Query OK, 0 rows affected (0.00 sec)

mysql> WITH RECURSIVE cte (n) AS (SELECT 1 UNION ALL SELECT n + 1 FROM cte WHERE n <
 60000000) SELECT max(n) FROM cte;
+----------+
| max(n)   |
+----------+
| 60000000 |
+----------+
1 row in set (13.99 sec)
```

```
mysql> show status like '%created_tmp_disk%';
+-------------------------+-------+
| Variable_name           | Value |
+-------------------------+-------+
| Created_tmp_disk_tables | 0     |
+-------------------------+-------+
1 row in set (0.00 sec)
```

Example **aurora_tmptable_enable_per_table_limit** 设置为 **ON** 的写入器数据库实例

内存中临时表转换为 InnoDB 磁盘上临时表。

```
mysql> set aurora_tmptable_enable_per_table_limit=1;
Query OK, 0 rows affected (0.00 sec)

mysql> select
 @@innodb_read_only,@@aurora_version,@@aurora_tmptable_enable_per_table_limit,@@tmp_table_size;
+--------------------+------------------+-----------------------------------------
+------------------+
| @@innodb_read_only | @@aurora_version | @@aurora_tmptable_enable_per_table_limit |
 @@tmp_table_size |
+--------------------+------------------+-----------------------------------------
+------------------+
|                  0 | 3.04.0           |                                        1 |
      16777216 |
+--------------------+------------------+-----------------------------------------
+------------------+
1 row in set (0.00 sec)

mysql> set cte_max_recursion_depth=4294967295;
Query OK, 0 rows affected (0.00 sec)

mysql> show status like '%created_tmp_disk%';
+-------------------------+-------+
| Variable_name           | Value |
+-------------------------+-------+
| Created_tmp_disk_tables | 0     |
+-------------------------+-------+
1 row in set (0.00 sec)

mysql> WITH RECURSIVE cte (n) AS (SELECT 1 UNION ALL SELECT n + 1 FROM cte WHERE n <
 6000000) SELECT max(n) FROM cte;
+---------+
| max(n)  |
```

```
+---------+
| 6000000 |
+---------+
1 row in set (4.10 sec)

mysql> show status like '%created_tmp_disk%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| Created_tmp_disk_tables | 1     |
+--------------------------+-------+
1 row in set (0.00 sec)
```

Example **aurora_tmptable_enable_per_table_limit** 设置为 **OFF** 的读取器数据库实例

查询完成时没有出现错误，因为 tmp_table_size 不适用，并且尚未达到全局 TempTable 资源限制。

```
mysql> set aurora_tmptable_enable_per_table_limit=0;
Query OK, 0 rows affected (0.00 sec)

mysql> select
 @@innodb_read_only,@@aurora_version,@@aurora_tmptable_enable_per_table_limit,@@temptable_max_r
+--------------------+------------------+-----------------------------------------
+----------------------+----------------------+
| @@innodb_read_only | @@aurora_version | @@aurora_tmptable_enable_per_table_limit |
 @@temptable_max_ram | @@temptable_max_mmap |
+--------------------+------------------+-----------------------------------------
+----------------------+----------------------+
|                  1 | 3.04.0           |                                        0 |
       1073741824 |           1073741824 |
+--------------------+------------------+-----------------------------------------
+----------------------+----------------------+
1 row in set (0.00 sec)

mysql> set cte_max_recursion_depth=4294967295;
Query OK, 0 rows affected (0.00 sec)

mysql> WITH RECURSIVE cte (n) AS (SELECT 1 UNION ALL SELECT n + 1 FROM cte WHERE n <
 60000000) SELECT max(n) FROM cte;
+----------+
| max(n)   |
+----------+
```

```
| 60000000 |
+----------+
1 row in set (14.05 sec)
```

Example **aurora_tmptable_enable_per_table_limit** 设置为 **OFF** 的读取器数据库实例

此查询达到全局 TempTable 资源限制，aurora_tmptable_enable_per_table_limit 设置为 OFF。查询结束时读取器实例上出现错误。

```
mysql> set aurora_tmptable_enable_per_table_limit=0;
Query OK, 0 rows affected (0.00 sec)

mysql> select
 @@innodb_read_only,@@aurora_version,@@aurora_tmptable_enable_per_table_limit,@@temptable_max_r
+--------------------+------------------+-----------------------------------------
+---------------------+----------------------+
| @@innodb_read_only | @@aurora_version | @@aurora_tmptable_enable_per_table_limit |
 @@temptable_max_ram | @@temptable_max_mmap |
+--------------------+------------------+-----------------------------------------
+---------------------+----------------------+
|                  1 | 3.04.0           |                                        0 |
       1073741824 |           1073741824 |
+--------------------+------------------+-----------------------------------------
+---------------------+----------------------+
1 row in set (0.00 sec)

mysql> set cte_max_recursion_depth=4294967295;
Query OK, 0 rows affected (0.01 sec)

mysql> WITH RECURSIVE cte (n) AS (SELECT 1 UNION ALL SELECT n + 1 FROM cte WHERE n <
 120000000) SELECT max(n) FROM cte;
ERROR 1114 (HY000): The table '/rdsdbdata/tmp/#sqlfd_1586_2' is full
```

Example **aurora_tmptable_enable_per_table_limit** 设置为 **ON** 的读取器数据库实例

如果达到 tmp_table_size 限制，查询结束时出现错误。

```
mysql> set aurora_tmptable_enable_per_table_limit=1;
Query OK, 0 rows affected (0.00 sec)

mysql> select
 @@innodb_read_only,@@aurora_version,@@aurora_tmptable_enable_per_table_limit,@@tmp_table_size;
```

```
+--------------------+------------------+-----------------------------------------
+------------------+
| @@innodb_read_only | @@aurora_version | @@aurora_tmptable_enable_per_table_limit |
 @@tmp_table_size |
+--------------------+------------------+-----------------------------------------
+------------------+
|                  1 | 3.04.0           |                                        1 |
      16777216 |
+--------------------+------------------+-----------------------------------------
+------------------+
1 row in set (0.00 sec)

mysql> set cte_max_recursion_depth=4294967295;
Query OK, 0 rows affected (0.00 sec)

mysql> WITH RECURSIVE cte (n) AS (SELECT 1 UNION ALL SELECT n + 1 FROM cte WHERE n <
 6000000) SELECT max(n) FROM cte;
ERROR 1114 (HY000): The table '/rdsdbdata/tmp/#sqlfd_8_2' is full
```

缓解 Aurora 副本上内部临时表的完整性问题

为避免临时表的大小限制问题，请将 temptable_max_ram 和 temptable_max_mmap 参数设置为可以满足工作负载要求的组合值。

设置 temptable_max_ram 参数的值时要小心。将该值设置为过高会减少数据库实例上的可用内存，这可能会导致内存不足的情况。监控数据库实例上的平均可用内存。然后，确定 temptable_max_ram 的适当值，以便实例上仍剩余合理的可用内存量。有关更多信息，请参阅 Amazon Aurora 中的可用内存问题。

监控本地存储的大小和临时表空间占用情况也很重要。可以使用 FreeLocalStorage Amazon CloudWatch 指标监控特定数据库实例可用的临时存储，如 Amazon Aurora 的 Amazon CloudWatch 指标中所述。

> **Note**
>
> 当 aurora_tmptable_enable_per_table_limit 参数设置为 ON 时，此过程不起作用。有关更多信息，请参阅限制内部内存中临时表的大小。

Example 1

您知道临时表增长到 20 GiB 的累计大小。您希望将内存中的临时表设置为 2 GiB，而在磁盘上增长到最大 20 GiB。

将 `temptable_max_ram` 设置为 **`2,147,483,648`**，将 `temptable_max_mmap` 设置为 **`21,474,836,480`**。这些值以字节为单位。

这些参数设置可确保临时表可以增长到累计 22 GiB 的总大小。

Example 2

您当前的实例大小为 16xlarge 或更大。您不知道可能需要的临时表的总大小。您希望能够在内存中使用多达 4 GiB，并在磁盘上使用最大的可用存储大小。

将 `temptable_max_ram` 设置为 **`4,294,967,296`**，将 `temptable_max_mmap` 设置为 **`1,099,511,627,776`**。这些值以字节为单位。

在此，您将 `temptable_max_mmap` 设置为 1 TiB，这小于 16xlarge Aurora 数据库实例上的最大本地存储 1.2 TiB。

在较小的实例大小上，调整 `temptable_max_mmap` 的值，以使它不会填满可用的本地存储。例如，2xlarge 实例只有 160 GiB 的本地存储可用。因此，我们建议将该值设置为小于 160 GiB。有关数据库实例大小的可用本地存储的更多信息，请参阅 Aurora MySQL 的临时存储限制。

优化 Aurora MySQL 数据库实例上的 temptable_max_mmap 参数

Aurora MySQL 中的 `temptable_max_mmap` 参数控制内存映射文件可以使用的最大本地磁盘空间量，超出此空间量之后，将溢出到磁盘上的 InnoDB 临时表（在写入器数据库实例上）或导致错误（在读取器数据库实例上）。正确设置此数据库实例参数有助于优化数据库实例的性能。

先决条件

1. 确保 Performance Schema 已启用。可以通过运行以下 SQL 命令进行验证：

   ```
   SELECT @@performance_schema;
   ```

   输出值为 1 表示已启用。
2. 确认临时表内存检测已启用。可以通过运行以下 SQL 命令进行验证：

   ```
   SELECT name, enabled FROM performance_schema.setup_instruments WHERE name LIKE
    '%memory%temptable%';
   ```

对于相关的临时表内存检测条目，enabled 列显示 YES。

监控临时表使用情况

建议您在设置 temptable_max_mmap 初始值时，一开始先设置为正在使用的数据库实例类的本地存储大小的 80%。这样可以确保临时表有足够的磁盘空间来高效运行，同时为实例上的其他磁盘使用留出空间。

要查找您的数据库实例类的本地存储大小，请参阅 Aurora MySQL 的临时存储限制。

例如，如果您使用的是 db.r5.large 数据库实例类，则本地存储大小为 32 GiB。在本例中，最初应将 temptable_max_mmap 参数设置为 32 GiB 的 80%，即 25.6 GiB。

设置初始 temptable_max_mmap 值后，在 Aurora MySQL 实例上运行您的峰值工作负载。使用以下 SQL 查询监控当前的和较高的临时表磁盘使用率：

```
SELECT event_name, current_count, current_alloc, current_avg_alloc, high_count,
 high_alloc, high_avg_alloc
FROM sys.memory_global_by_current_bytes WHERE event_name LIKE 'memory/temptable/%';
```

此查询检索以下信息：

- event_name – 临时表内存或磁盘使用事件的名称。
- current_count – 当前分配的临时表内存或磁盘块的数量。
- current_alloc – 当前为临时表分配的内存或磁盘量。
- current_avg_alloc – 临时表内存或磁盘块的当前平均大小。
- high_count – 分配的临时表内存或磁盘块的最大数量。
- high_alloc – 为临时表分配的最大内存或磁盘量。
- high_avg_alloc – 临时表内存或磁盘块的最大平均大小。

如果使用此设置时查询失败并出现表已满错误，则表示您的工作负载需要更多磁盘空间才能完成临时表操作。在这种情况下，可以考虑增加数据库实例大小，以便有更多本地存储空间。

设置最佳 **temptable_max_mmap** 值

使用以下步骤监控 temptable_max_mmap 参数并为其设置适当的大小。

1. 查看上一个查询的输出，并确定临时表磁盘使用率峰值，如 high_alloc 列所示。
2. 根据临时表磁盘使用峰值，调整 Aurora MySQL 数据库实例的数据库参数组中的参数 temptable_max_mmap。

将值设置为比临时表磁盘使用量峰值略高一点，以适应未来的增长。

3. 将参数组更改应用于您的数据库实例。
4. 在工作负载高峰期再次监控临时表磁盘使用情况，以确保新 temptable_max_mmap 值合适。
5. 根据需要重复前面的步骤来微调 temptable_max_mmap 参数。

读取器数据库实例上用户创建的（显式）临时表

您可以在 CREATE TABLE 语句中使用 TEMPORARY 关键字创建显式临时表。Aurora 数据库集群中的写入器数据库实例支持显式临时表。您还可以在读取器数据库实例上使用显式临时表，但这些表无法强制使用 InnoDB 存储引擎。

为避免在 Aurora MySQL 读取器数据库实例上创建显式临时表时出现错误，请确保按以下任一或两种方式运行所有 CREATE TEMPORARY TABLE 语句：

- 不要指定 ENGINE=InnoDB 子句。
- 请勿将 SQL 模式设置为 NO_ENGINE_SUBSTITUTION。

临时表创建错误和缓解

您收到的错误会有所不同，具体取决于您使用的是简单 CREATE TEMPORARY TABLE 语句还是变体 CREATE TEMPORARY TABLE AS SELECT。以下示例显示不同类型的错误。

此临时表行为仅适用于只读实例。第一个示例证实了这是会话连接到的实例类型。

```
mysql> select @@innodb_read_only;
+--------------------+
| @@innodb_read_only |
+--------------------+
|                  1 |
+--------------------+
```

对于简单的 CREATE TEMPORARY TABLE 语句，语句会在 NO_ENGINE_SUBSTITUTION SQL 模式开启时失败。当 NO_ENGINE_SUBSTITUTION 处于关闭状态（默认设置）时，将进行适当的引擎替换，并成功创建临时表。

```
mysql> set sql_mode = 'NO_ENGINE_SUBSTITUTION';

mysql>  CREATE TEMPORARY TABLE tt2 (id int) ENGINE=InnoDB;
```

```
ERROR 3161 (HY000): Storage engine InnoDB is disabled (Table creation is disallowed).

mysql> SET sql_mode = '';

mysql> CREATE TEMPORARY TABLE tt4 (id int) ENGINE=InnoDB;

mysql> SHOW CREATE TABLE tt4\G
*************************** 1. row ***************************
       Table: tt4
Create Table: CREATE TEMPORARY TABLE `tt4` (
  `id` int DEFAULT NULL
) ENGINE=MyISAM DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

对于 CREATE TEMPORARY TABLE AS SELECT 语句，语句会在 NO_ENGINE_SUBSTITUTION SQL 模式开启时失败。当 NO_ENGINE_SUBSTITUTION 处于关闭状态（默认设置）时，将进行适当的引擎替换，并成功创建临时表。

```
mysql> set sql_mode = 'NO_ENGINE_SUBSTITUTION';

mysql> CREATE TEMPORARY TABLE tt1 ENGINE=InnoDB AS SELECT * FROM t1;
ERROR 3161 (HY000): Storage engine InnoDB is disabled (Table creation is disallowed).

mysql> SET sql_mode = '';

mysql> show create table tt3;
+-------+-----------------------------------------------------------------+
| Table | Create Table                                                    |
+-------+-----------------------------------------------------------------+
| tt3   | CREATE TEMPORARY TABLE `tt3` (
  `id` int DEFAULT NULL
) ENGINE=MyISAM DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci |
+-------+-----------------------------------------------------------------+
1 row in set (0.00 sec)
```

有关 Aurora MySQL 版本 3 中临时表的存储方面和性能影响的更多信息，请参阅博客文章在 Amazon RDS for MySQL 和 Amazon Aurora MySQL 上使用 TempTable 存储引擎。

比较 Aurora MySQL 版本 2 和 Aurora MySQL 版本 3

使用以下内容了解在将 Aurora MySQL 版本 2 集群升级到版本 3 时需要注意的更改。

主题

- 原子数据定义语言（DDL）支持
- Aurora MySQL 版本 2 和 3 之间的功能区别
- 实例类支持
- Aurora MySQL 版本 3 的参数更改
- 状态变量
- Aurora MySQL 版本 3 的包容性语言更改
- AUTO_INCREMENT 值
- 二进制日志复制

原子数据定义语言（DDL）支持

从 MySQL 5.7 到 8.0 的最大变化之一是引入了 Atomic Data Dictionary。在 MySQL 8.0 之前，MySQL 数据字典使用基于文件的方法，来将表定义 (.frm)、触发器 (.trg) 和函数等元数据与存储引擎的元数据（例如 InnoDB 的元数据）分开存储。这会有一些问题，包括以下这类情况：如果在 DDL 操作期间发生意外情况，导致基于文件的元数据和存储引擎元数据不同步，表就有变为“orphaned”的风险。

为了解决这个问题，MySQL 8.0 引入了原子数据字典，它将所有元数据存储在架构 `mysql` 内的一组内部 InnoDB 表中。这种新架构提供了一种事务性、符合 ACID 的方法来管理数据库元数据，解决了基于文件的旧方法中的“原子 DDL”问题。有关原子数据字典的更多信息，请参阅《MySQL 参考手册》中的 Removal of file-based metadata storage 和 Atomic data definition statement support。

由于这种架构更改，从 Aurora MySQL 版本 2 升级到版本 3 时，必须考虑以下因素：

- 在升级到版本 3 的过程中，必须将版本 2 中基于文件的元数据迁移到新的数据字典表中。这可能需要花一些时间，具体取决于要迁移的数据库对象数量。
- 这些更改还引入了一些新的不兼容性，在从 MySQL 5.7 升级到 8.0 之前，可能需要解决这类问题。例如，8.0 有一些新的保留关键字，它们可能与现有的数据库对象名称冲突。

为了协助您在升级引擎之前确定这些不兼容性，在执行数据字典升级之前，Aurora MySQL 会运行一系列升级兼容性检查（预检查），来确定数据库字典中是否存在任何不兼容的对象。有关预检查的更多信息，请参阅 Aurora MySQL 的主要版本升级预检查。

Aurora MySQL 版本 2 和 3 之间的功能区别

在 Aurora MySQL for MySQL 5.7 中支持以下 Amazon Aurora MySQL 功能，但在 Aurora MySQL for MySQL 8.0 中不支持这些功能。

- 您不能将 Aurora MySQL 版本 3 用于 Aurora Serverless v1 集群。Aurora MySQL 版本 3 可与 Aurora Serverless v2 一起使用。
- 实验室模式不适用于 Aurora MySQL 版本 3。Aurora MySQL 版本 3 中没有任何实验室模式功能。即时 DDL 取代了以前在实验室模式下可用的快速线上 DDL 功能。有关示例，请参阅即时 DDL（Aurora MySQL 版本 3）。
- 查询缓存已从社区 MySQL 8.0 以及 Aurora MySQL 版本 3 中删除。
- Aurora MySQL 版本 3 与社区 MySQL 即时哈希联接功能兼容。未使用 Aurora MySQL 版本 2 中特定于 Aurora 的哈希联接实现。有关在 Aurora 并行查询中使用哈希联接的信息，请参阅 为并行查询集群开启哈希联接 和 Aurora MySQL 提示。有关哈希联接的一般用法信息，请参阅 MySQL 参考手册中的哈希联接优化。
- 在 Aurora MySQL 版本 2 中弃用的 `mysql.lambda_async` 存储过程将在版本 3 中删除。对于版本 3，请使用异步函数 `lambda_async`。
- Aurora MySQL 版本 3 中的原定设置字符集是 `utf8mb4`。Aurora MySQL 版本 2 中的原定设置字符集是 `latin1`。有关此字符集的信息，请参阅 MySQL 参考手册中的 utf8mb4 字符集（4 字节 UTF-8 Unicode 编码）。

一些 Aurora MySQL 功能可用于 AWS 区域和数据库引擎版本的特定组合。有关详细信息，请参阅Amazon Aurora 中受 AWS 区域 和 Aurora 数据库引擎支持的功能。

实例类支持

Aurora MySQL 版本 3 支持与 Aurora MySQL 版本 2 不同的一组实例类：

- 对于较大的实例，您可以使用现代实例类，例如 `db.r5`、`db.r6g` 和 `db.x2g`。
- 对于较大的实例，您可以使用现代实例类，例如 `db.t3` 和 `db.t4g`。

> **Note**
>
> 建议仅将 T 数据库实例类用于开发和测试服务器，或其他非生产服务器。有关 T 实例类的更多详细信息，请参阅使用 T 实例类进行开发和测试。

Aurora MySQL 版本 2 中的以下实例类不适用于 Aurora MySQL 版本 3：

- `db.r4`
- `db.r3`

- db.t3.small
- db.t2

检查您的管理脚本中是否有任何创建 Aurora MySQL 数据库实例的 CLI 语句。Aurora MySQL 版本 3 不可用的硬编码实例类名称。如有必要，请将实例类名称修改为 Aurora MySQL 版本 3 支持的名称。

> **Tip**
>
> 检查可用于 Aurora MySQL 版本和 AWS 区域特定组合的实例类，请使用 `describe-orderable-db-instance-options` AWS CLI 命令。

有关 Aurora 实例类的完整详细信息，请参阅 Amazon Aurora 数据库实例类。

Aurora MySQL 版本 3 的参数更改

Aurora MySQL 版本 3 包括新的集群级和实例级配置参数。Aurora MySQL 版本 3 还删除了 Aurora MySQL 版本 2 中存在的一些参数。由于包容性语言的倡议，一些参数名称发生了变化。为了向后兼容，您仍可以使用旧名称或使用新名称检索参数值。但是，您必须使用新名称在自定义参数组中指定参数值。

在 Aurora MySQL 版本 3 中，`lower_case_table_names` 参数的值在创建集群时永久设置。如果对此选项使用非原定设置值，请在升级之前设置 Aurora MySQL 版本 3 自定义参数组。然后，在创建集群或快照还原操作期间指定参数组。

> **Note**
>
> 使用基于 Aurora MySQL 的 Aurora 全局数据库时，如果开启了 `lower_case_table_names` 参数，则无法执行从 Aurora MySQL 版本 2 到版本 3 的就地升级。改用快照还原方法。

在 Aurora MySQL 版本 3 中，`init_connect` 和 `read_only` 参数不适用于具有 `CONNECTION_ADMIN` 权限的用户。这包括 Aurora 主用户。有关更多信息，请参阅 基于角色的权限模型。

有关 Aurora MySQL 集群参数的完整列表，请参阅 集群级别的参数。该表涵盖了 Aurora MySQL 版本 2 和 3 的所有参数。该表包括一些说明，显示了 Aurora MySQL 版本 3 中哪些参数是新增的，哪些参数已从 Aurora MySQL 版本 3 中删除。

有关 Aurora MySQL 实例参数的完整列表，请参阅 实例级参数。该表涵盖了 Aurora MySQL 版本 2 和 3 的所有参数。该表包括一些说明，显示了 Aurora MySQL 版本 3 中哪些参数是新增的，哪些参数已从 Aurora MySQL 版本 3 中删除。它还包括显示哪些参数在早期版本中可修改，但不能在 Aurora MySQL 版本 3 中修改的说明。

有关已更改的参数名称的信息，请参阅 Aurora MySQL 版本 3 的包容性语言更改。

状态变量

有关不适用于 Aurora MySQL 的状态变量的信息，请参阅 不适用于 Aurora MySQL 的 MySQL 状态变量。

Aurora MySQL 版本 3 的包容性语言更改

Aurora MySQL 版本 3 与 MySQL 社群版的 8.0.23 版本兼容。Aurora MySQL 版本 3 还包括 MySQL 8.0.26 中与包容性语言的关键字和系统架构相关的更改。例如，现在首选 `SHOW REPLICA STATUS` 命令，而不是 `SHOW SLAVE STATUS`。

以下 Amazon CloudWatch 指标在 Aurora MySQL 版本 3 中有新名称。

在 Aurora MySQL 版本 3 中，只有新的指标名称可用。升级到 Aurora MySQL 版本 3 时，请务必更新依赖指标名称的任何告警或其他自动化。

旧名称	新名称
`ForwardingMasterDMLLatency`	`ForwardingWriterDMLLatency`
`ForwardingMasterOpenSessions`	`ForwardingWriterOpenSessions`
`AuroraDMLRejectedMasterFull`	`AuroraDMLRejectedWriterFull`
`ForwardingMasterDMLThroughput`	`ForwardingWriterDMLThroughput`

以下状态变量在 Aurora MySQL 版本 3 中有新名称。

为了获得兼容性，您可以在初始的 Aurora MySQL 版本 3 中使用任何一个名称。未来版本将删除旧的状态变量名称。

要删除的名称	新名称或首选名称	
Aurora_fwd_master_dml_stmt_duration	Aurora_fwd_writer_dml_stmt_duration	
Aurora_fwd_master_dml_stmt_count	Aurora_fwd_writer_dml_stmt_count	
Aurora_fwd_master_select_stmt_duration	Aurora_fwd_writer_select_stmt_duration	
Aurora_fwd_master_select_stmt_count	Aurora_fwd_writer_select_stmt_count	
Aurora_fwd_master_errors_session_timeout	Aurora_fwd_writer_errors_session_timeout	
Aurora_fwd_master_open_sessions	Aurora_fwd_writer_open_sessions	
Aurora_fwd_master_errors_session_limit	Aurora_fwd_writer_errors_session_limit	
Aurora_fwd_master_errors_rpc_timeout	Aurora_fwd_writer_errors_rpc_timeout	

以下配置参数在 Aurora MySQL 版本 3 中有新名称。

为了获得兼容性，您可以在初始的 Aurora MySQL 版本 3 中使用任何一个名称以检查 `mysql` 客户端中的参数值。当修改自定义参数组中的值时，只能使用新名称。未来版本将删除旧的参数名称。

要删除的名称	新名称或首选名称	
aurora_fwd_master_idle_timeout	aurora_fwd_writer_idle_timeout	

要删除的名称	新名称或首选名称	
aurora_fwd_master_max_connections_pct	aurora_fwd_writer_max_connections_pct	
master_verify_checksum	source_verify_checksum	
sync_master_info	sync_source_info	
init_slave	init_replica	
rpl_stop_slave_timeout	rpl_stop_replica_timeout	
log_slow_slave_statements	log_slow_replica_statements	
slave_max_allowed_packet	replica_max_allowed_packet	
slave_compressed_protocol	replica_compressed_protocol	
slave_exec_mode	replica_exec_mode	
slave_type_conversions	replica_type_conversions	
slave_sql_verify_checksum	replica_sql_verify_checksum	
slave_parallel_type	replica_parallel_type	
slave_preserve_commit_order	replica_preserve_commit_order	
log_slave_updates	log_replica_updates	

要删除的名称	新名称或首选名称	
slave_allow_batching	replica_allow_batching	
slave_load_tmpdir	replica_load_tmpdir	
slave_net_timeout	replica_net_timeout	
sql_slave_skip_counter	sql_replica_skip_counter	
slave_skip_errors	replica_skip_errors	
slave_checkpoint_period	replica_checkpoint_period	
slave_checkpoint_group	replica_checkpoint_group	
slave_transaction_retries	replica_transaction_retries	
slave_parallel_workers	replica_parallel_workers	
slave_pending_jobs_size_max	replica_pending_jobs_size_max	
pseudo_slave_mode	pseudo_replica_mode	

以下存储过程在 Aurora MySQL 版本 3 中有新名称。

为了获得兼容性，您可以在初始的 Aurora MySQL 版本 3 中使用任何一个名称。未来版本将删除旧的过程名称。

要删除的名称	新名称或首选名称	
mysql.rds_set_master_auto_position	mysql.rds_set_source_auto_position	
mysql.rds_set_external_master	mysql.rds_set_external_source	
mysql.rds_set_external_master_with_auto_position	mysql.rds_set_external_source_with_auto_position	
mysql.rds_reset_external_master	mysql.rds_reset_external_source	
mysql.rds_next_master_log	mysql.rds_next_source_log	

AUTO_INCREMENT 值

在 Aurora MySQL 版本 3 中，Aurora 在重启每个数据库实例时保留每个表的 AUTO_INCREMENT 值。在 Aurora MySQL 版本 2 中，AUTO_INCREMENT 值在重启后不会保留。

当您通过从快照还原、执行时间点恢复和克隆集群来设置新集群时，不会保留 AUTO_INCREMENT 值。在这些情况下，AUTO_INCREMENT 值将根据创建快照时表中的最大列值初始化为该值。此行为与 RDS for MySQL 8.0 中的行为不同，其中 AUTO_INCREMENT 值在这些操作过程中将被保留。

二进制日志复制

在 MySQL 8.0 社群版中，预设情况下，二进制日志复制处于开启状态。预设情况下，在 Aurora MySQL 版本 3 中，二进制日志复制处于关闭状态。

> **Tip**
>
> 如果 Aurora 内置复制功能满足了您的高可用性要求，则可以关闭二进制日志复制。这样，您就可以避免二进制日志复制的性能开销。您还可以避免管理二进制日志复制所需的相关监控和故障排除过程。

Aurora 支持从兼容 MySQL 5.7 的源到 Aurora MySQL 版本 3 的二进制日志复制。源系统可以是 Aurora MySQL 数据库集群、RDS for MySQL 数据库实例或本地 MySQL 实例。

与社区 MySQL 一样，Aurora MySQL 支持从运行特定版本的源复制到运行相同主要版本或更高版本的目标。例如，不支持从兼容 MySQL 5.6 的系统复制到 Aurora MySQL 版本 3。不支持将 Aurora MySQL 版本 3 复制到兼容 MySQL 5.7 的系统或兼容 MySQL 5.6 的系统。有关使用二进制日志复制的详细信息，请参阅 Aurora 与 MySQL 之间或 Aurora 与其他 Aurora 数据库集群之间的复制（二进制日志复制）。

Aurora MySQL 版本 3 包括对社区 MySQL 8.0 中二进制日志复制的改进，例如筛选的复制。有关社区 MySQL 8.0 改进的详细信息，请参阅 MySQL 参考手册中的服务器如何评估复制筛选规则。

二进制日志复制的事务压缩

有关二进制日志压缩的使用信息，请参阅 MySQL 参考手册中的二进制日志事务压缩。

以下限制适用于 Aurora MySQL 版本 3 中的二进制日志压缩：

- 不会压缩其二进制日志数据大于允许的最大数据包大小的事务。无论是否开启 Aurora MySQL 二进制日志压缩设置，都是如此。此类事务在不被压缩的情况下被复制。
- 如果您使用尚不支持 MySQL 8.0 的更改数据捕获 (CDC) 的连接器，则无法使用此功能。我们建议您使用二进制日志压缩彻底测试任何第三方连接器。此外，我们建议您在对 CDC 使用二进制日志复制的系统上开启二进制日志压缩之前先这样做。

比较 Aurora MySQL 版本 3 和 MySQL 8.0 社群版

您可以使用以下信息了解从不同的 MySQL 8.0 兼容系统转换为 Aurora MySQL 版本 3 时需要注意的更改。

一般来说，Aurora MySQL 版本 3 支持社区 MySQL 8.0.23 的功能集。MySQL 8.0 社群版的一些新功能不适用于 Aurora MySQL。其中一些功能与 Aurora 的某些方面不兼容，例如 Aurora 存储架构。不需要其他功能，因为 Amazon RDS 管理服务提供了同等的功能。社区 MySQL 8.0 中的以下功能不受支持或者在 Aurora MySQL 版本 3 中的工作方式有所不同。

有关所有 Aurora MySQL 版本 3 发布的发布说明，请参阅《Aurora MySQL 发布说明》中的 Amazon Aurora MySQL 版本 3 的数据库引擎更新。

主题

- MySQL 8.0 功能在 Aurora MySQL 版本 3 中不可用

- 基于角色的权限模型
- 查找数据库服务器 ID
- 身份验证

MySQL 8.0 功能在 Aurora MySQL 版本 3 中不可用

社区 MySQL 8.0 中的以下功能未提供或者在 Aurora MySQL 版本 3 中的工作方式有所不同。

- Aurora MySQL 不支持资源组和相关的 SQL 语句。
- Aurora MySQL 不支持用户定义的撤消表空间和关联的 SQL 语句，例如 `CREATE UNDO TABLESPACE`、`ALTER UNDO TABLESPACE ... SET INACTIVE` 和 `DROP UNDO TABLESPACE`。
- 对于低于 3.06 的 Aurora MySQL 版本，Aurora MySQL 不支持撤消表空间截断。在 Aurora MySQL 3.06 及更高版本中，支持自动撤消表空间截断。
- 您无法修改任何 MySQL 插件的设置。
- 不支持 X 插件。
- 不支持多源复制。

基于角色的权限模型

使用 Aurora MySQL 版本 3，您无法直接修改 `mysql` 数据库中的表。特别是，您无法通过插入到 `mysql.user` 表来设置用户。相反，您可以使用 SQL 语句授予基于角色的权限。您也无法创建其他类型的对象，例如 `mysql` 数据库中的存储过程。您仍然可以查询 `mysql` 表。如果您使用二进制日志复制，则直接对源集群上的 `mysql` 表进行的更改不会复制到目标集群中。

在某些情况下，您的应用程序可能会使用快捷方式通过插入到 `mysql` 表来创建用户或其他对象。如果是这样，请更改应用程序代码以使用相应的语句，例如 `CREATE USER`。如果您的应用程序在 `mysql` 数据库中创建存储过程或其他对象，请改用其他数据库。

要在从外部 MySQL 数据库迁移期间导出数据库用户的元数据，您可以使用 MySQL Shell 命令而非 `mysqldump`。有关更多信息，请参阅 Instance Dump Utility, Schema Dump Utility, and Table Dump Utility。

为了简化对许多用户或应用程序的权限管理，您可以使用 `CREATE ROLE` 语句来创建具有一组权限的角色。然后，您可以使用 `GRANT` 和 `SET ROLE` 语句以及 `current_role` 函数将角色分配给用户或应用程序、切换当前角色以及检查哪些角色有效。有关 MySQL 8.0 中基于角色的权限系统的更多信息，请参阅 MySQL 参考手册中的使用角色。

> **Important**
>
> 我们强烈建议不要直接在应用程序中使用主用户。请遵守使用数据库用户的最佳实践，按照您的应用程序所需的最少权限创建用户。

主题

- rds_superuser_role
- 二进制日志复制的权限检查用户
- 用于访问其它 AWS 服务的角色

rds_superuser_role

Aurora MySQL 版本 3 包括一个具有以下所有权限的特殊角色。该角色命名为 `rds_superuser_role`。每个集群的主管理用户已经授予了此角色。`rds_superuser_role` 角色包括所有数据库对象的以下权限：

- `ALTER`
- `APPLICATION_PASSWORD_ADMIN`
- `ALTER ROUTINE`
- `CONNECTION_ADMIN`
- `CREATE`
- `CREATE ROLE`
- `CREATE ROUTINE`
- `CREATE TEMPORARY TABLES`
- `CREATE USER`
- `CREATE VIEW`
- `DELETE`
- `DROP`
- `DROP ROLE`
- `EVENT`
- `EXECUTE`

- FLUSH_OPTIMIZER_COSTS（Aurora MySQL 版本 3.09 及更高版本）
- FLUSH_STATUS（Aurora MySQL 版本 3.09 及更高版本）
- FLUSH_TABLES（Aurora MySQL 版本 3.09 及更高版本）
- FLUSH_USER_RESOURCES（Aurora MySQL 版本 3.09 及更高版本）
- INDEX
- INSERT
- LOCK TABLES
- PROCESS
- REFERENCES
- RELOAD
- REPLICATION CLIENT
- REPLICATION SLAVE
- ROLE_ADMIN
- SET_USER_ID
- SELECT
- SHOW DATABASES
- SHOW_ROUTINE（Aurora MySQL 版本 3.04 及更高版本）
- SHOW VIEW
- TRIGGER
- UPDATE
- XA_RECOVER_ADMIN

角色定义还包括 WITH GRANT OPTION，以便管理用户可以将该角色授予其他用户。特别是，管理员必须授予以 Aurora MySQL 集群作为目标执行二进制日志复制所需的任何权限。

> **Tip**
>
> 要查看权限的完整详细信息，请输入以下语句。
>
> ```
> SHOW GRANTS FOR rds_superuser_role@'%';
> SHOW GRANTS FOR name_of_administrative_user_for_your_cluster@'%';
> ```

二进制日志复制的权限检查用户

Aurora MySQL 版本 3 包括二进制日志（binlog）复制的权限检查用户，即 `rdsrepladmin_priv_checks_user`。除 `rds_superuser_role` 的权限之外，此用户还拥有 `replication_applier` 权限。

当您通过调用 `mysql.rds_start_replication` 存储过程来开启二进制日志复制时，即会创建 `rdsrepladmin_priv_checks_user`。

`rdsrepladmin_priv_checks_user@localhost` 用户是保留用户。请勿修改此用户。

用于访问其它 AWS 服务的角色

Aurora MySQL 版本 3 包括可用于访问其它 AWS 服务的角色。可以将其中许多角色设置为授予权限的替代。例如，您可以指定 `GRANT AWS_LAMBDA_ACCESS TO user` 而不是 `GRANT INVOKE LAMBDA ON *.* TO user`。对于访问其他 AWS 服务的程序，请参阅 将 Amazon Aurora MySQL 与其他AWS 服务集成。Aurora MySQL 版本 3 包括以下与访问其他 AWS 服务相关的角色：

- `AWS_LAMBDA_ACCESS` – `INVOKE LAMBDA` 权限的替代。有关使用信息，请参阅从 Amazon Aurora MySQL 数据库集群中调用 Lambda 函数。
- `AWS_LOAD_S3_ACCESS` – `LOAD FROM S3` 权限的替代。有关使用信息，请参阅将数据从 Amazon S3 存储桶中的文本文件加载到 Amazon Aurora MySQL 数据库集群。
- `AWS_SELECT_S3_ACCESS` – `SELECT INTO S3` 权限的替代。有关使用信息，请参阅将数据从 Amazon Aurora MySQL 数据库集群保存到 Amazon S3 存储桶中的文本文件。
- `AWS_COMPREHEND_ACCESS` – `INVOKE COMPREHEND` 权限的替代。有关使用信息，请参阅授予数据库用户访问 Aurora 机器学习的权限。
- `AWS_SAGEMAKER_ACCESS` – `INVOKE SAGEMAKER` 权限的替代。有关使用信息，请参阅授予数据库用户访问 Aurora 机器学习的权限。
- `AWS_BEDROCK_ACCESS` – Amazon Bedrock 没有类似的 `INVOKE` 权限。有关使用信息，请参阅授予数据库用户访问 Aurora 机器学习的权限。

当您使用 Aurora MySQL 版本 3 中的角色授予访问权限时，还可以通过使用 `SET ROLE role_name` 或 `SET ROLE ALL` 语句来激活角色。下面的示例演示如何操作。将适当的角色名称替换为 `AWS_SELECT_S3_ACCESS`。

```
# Grant role to user.

mysql> GRANT AWS_SELECT_S3_ACCESS TO 'user'@'domain-or-ip-address'
```

```
# Check the current roles for your user. In this case, the AWS_SELECT_S3_ACCESS role
 has not been activated.
# Only the rds_superuser_role is currently in effect.
mysql> SELECT CURRENT_ROLE();
+--------------------------+
| CURRENT_ROLE()           |
+--------------------------+
| `rds_superuser_role`@`%` |
+--------------------------+
1 row in set (0.00 sec)

# Activate all roles associated with this user using SET ROLE.
# You can activate specific roles or all roles.
# In this case, the user only has 2 roles, so we specify ALL.
mysql> SET ROLE ALL;
Query OK, 0 rows affected (0.00 sec)

# Verify role is now active
mysql> SELECT CURRENT_ROLE();
+-----------------------------------------------------+
| CURRENT_ROLE()                                      |
+-----------------------------------------------------+
| `AWS_SELECT_S3_ACCESS`@`%`,`rds_superuser_role`@`%` |
+-----------------------------------------------------+
```

查找数据库服务器 ID

数据库服务器 ID（`server_id`）是二进制日志（binlog）复制所必需的。在 Aurora MySQL 中查找服务器 ID 的方式与在 Community MySQL 中查找不同。

在 Community MySQL 中，服务器 ID 是一个数字，您可以在登录服务器时使用以下语法获得：

```
mysql> select @@server_id;

+-------------+
| @@server_id |
+-------------+
| 2           |
+-------------+
1 row in set (0.00 sec)
```

在 Aurora MySQL 中，服务器 ID 是数据库实例 ID，您可以在登录数据库实例时使用以下语法获得：

```
mysql> select @@aurora_server_id;

+-------------------------+
| @@aurora_server_id      |
+-------------------------+
| mydbcluster-instance-2 |
+-------------------------+
1 row in set (0.00 sec)
```

有关二进制日志复制的更多信息，请参阅 Aurora 与 MySQL 之间或 Aurora 与其他 Aurora 数据库集群之间的复制（二进制日志复制）。

身份验证

在社区 MySQL 8.0 中，默认的身份验证插件是 `caching_sha2_password`。Aurora MySQL 版本 3 仍然使用 `mysql_native_password` 插件。您将无法更改 `default_authentication_plugin` 设置。但是，您可以创建新用户和更改当前用户，并且这些用户各自的密码使用新的身份验证插件。以下为示例。

```
mysql> CREATE USER 'testnewsha'@'%' IDENTIFIED WITH caching_sha2_password BY
 'aNewShaPassword';
Query OK, 0 rows affected (0.74 sec)
```

升级到 Aurora MySQL 版本 3

有关将数据库从 Aurora MySQL 版本 2 升级到版本 3 的信息，请参阅升级 Amazon Aurora MySQL 数据库集群的主要版本。

与 MySQL 5.7 兼容的 Aurora MySQL 版本 2

本主题介绍 Aurora MySQL 版本 2 和 MySQL 5.7 社区版之间的区别。

> **Important**
>
> Aurora MySQL 版本 2 的标准支持已于 2024 年 10 月 31 日结束。有关更多信息，请参阅 为 Amazon Aurora MySQL 兼容版的版本 2 终止标准支持做好准备。

Aurora MySQL 版本 2 中不支持的功能

以下功能在 MySQL 5.7 中受支持，但目前在 Aurora MySQL 版本 2 中不受支持：

- CREATE TABLESPACE SQL 语句
- 组复制插件
- 增加的页面大小
- InnoDB 缓冲池启动时加载
- InnoDB 全文分析器插件
- 多源复制
- 在线缓冲池大小调整
- 密码验证插件 – 您可以安装插件，但不受支持。无法自定义插件。
- 查询重写插件
- 复制筛选
- X 协议

有关这些功能的更多信息，请参阅 MySQL 5.7 文档。

Aurora MySQL 版本 2 中的临时表空间行为

在 MySQL 5.7 中，临时表空间会自动扩展并根据需要增加大小，以容纳磁盘上的临时表。删除临时表时，释放的空间可以重新用于新的临时表，但临时表空间保持扩展的大小而不会缩小。当引擎重新启动时，将删除并重新创建临时表空间。

在 Aurora MySQL 版本 2 中，以下行为适用：

- 对于使用版本 2.10 及更高版本创建的新的 Aurora MySQL 数据库集群，在您重新启动数据库时将删除并重新创建临时表空间。这允许动态调整大小功能回收存储空间。
- 对于升级到以下版本的现有 Aurora MySQL 数据库集群：
  - 版本 2.10 或更高版本 - 当您重新启动数据库时，将删除并重新创建临时表空间。这允许动态调整大小功能回收存储空间。
  - 版本 2.09 - 重新启动数据库时不会删除临时表空间。

您可以使用以下查询检查 Aurora MySQL 版本 2 数据库集群上临时表空间的大小：

```
SELECT
    FILE_NAME,
    TABLESPACE_NAME,
    ROUND((TOTAL_EXTENTS * EXTENT_SIZE) / 1024 / 1024 / 1024, 4) AS SIZE
```

```
FROM
    INFORMATION_SCHEMA.FILES
WHERE
    TABLESPACE_NAME = 'innodb_temporary';
```

有关更多信息，请参阅 MySQL 文档中的临时表空间。

磁盘上的临时表的存储引擎

Aurora MySQL 版本 2 根据实例的角色对磁盘上的内部临时表使用不同的存储引擎。

- 在写入器实例上，在默认情况下，磁盘上的临时表使用 InnoDB 存储引擎。它们存储在 Aurora 集群卷的临时表空间中。

  您可以通过修改数据库参数 `internal_tmp_disk_storage_engine` 的值来更改写入器实例上的这一行为。有关更多信息，请参阅 实例级参数。
- 在读取器实例上，磁盘上的临时表使用 MyISAM 存储引擎，该引擎使用本地存储。这是因为只读实例无法在 Aurora 集群卷上存储任何数据。

使用 Amazon Aurora MySQL 实现高安全性

Amazon Aurora MySQL 的安全性在三个级别上进行管理：

- 要控制可对 Aurora MySQL 数据库集群和数据库实例执行 Amazon RDS 管理操作的人员，请使用 AWS Identity and Access Management (IAM)。使用 IAM 凭证连接到AWS时，您的AWS账户必须具有授予执行 Amazon RDS 管理操作所需的权限的 IAM 策略。有关更多信息，请参阅 Amazon Aurora 的 Identity and Access Management

  如果要使用 IAM 访问 Amazon RDS 控制台，请确保首先使用您的 IAM 用户凭证登录 AWS Management Console。然后通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
- 确保基于 Amazon VPC 服务在 Virtual Public Cloud (VPC) 中创建 Aurora MySQL 数据库集群。要控制哪些设备和 Amazon EC2 实例能够建立与 VPC 中 Aurora MySQL 数据库集群的数据库实例的端点和端口的连接，请使用 VPC 安全组。您可以使用传输层安全性协议（TLS）建立这些端点和端口连接。此外，公司的防火墙规则也可以控制公司中运行的哪些设备可以建立与数据库实例的连接。有关 VPC 的更多信息，请参阅 Amazon VPC 和 Amazon Aurora。

  支持的 VPC 租赁取决于 Aurora MySQL 数据库集群使用的数据库实例类。对于 `default` VPC 租赁，VPC 在共享硬件上运行。对于 `dedicated` VPC 租赁，VPC 在专用硬件实例上运行。可突增的性能数据库实例类仅支持原定设置 VPC 租赁。可突增的性能数据库实例类包括 db.t2、db.t3 和 db.t4g 数据库实例类。所有其他 Aurora MySQL 数据库实例类都支持原定设置和专用 VPC 租赁。

  > **Note**
  >
  > 建议仅将 T 数据库实例类用于开发和测试服务器，或其他非生产服务器。有关 T 实例类的更多详细信息，请参阅使用 T 实例类进行开发和测试。

  有关实例类的更多信息，请参阅 Amazon Aurora 数据库实例类。有关 `default` 和 `dedicated` VPC 租赁的更多信息，请参阅 Amazon Elastic Compute Cloud 用户指南 中的专用实例。
- 要对 Amazon Aurora MySQL 数据库集群的登录信息和权限进行身份验证，可单独或组合采用以下各种方式：
  - 您可采用与独立 MySQL 实例相同的方式。

CREATE USER、RENAME USER、GRANT、REVOKE 和 SET PASSWORD 等命令的作用与它们在本地数据库中的作用相同，就像直接修改数据库架构表。有关更多信息，请参阅 MySQL 文档中的访问控制和账户管理。

- 您还可以使用 IAM 数据库身份验证。

  如果采用 IAM 数据库身份验证方式，可使用 IAM 用户或 IAM 角色以及身份验证令牌对您的数据库集群进行身份验证。身份验证令牌是使用签名版本 4 签名流程生成的唯一值。通过使用 IAM 数据库身份验证，您可以使用相同的凭证来控制对 AWS 资源和数据库的访问。有关更多信息，请参阅“ IAM 数据库身份验证”。

> **Note**
>
> 有关更多信息，请参阅 Amazon Aurora 中的安全性。

在以下各节中，请参阅有关 Aurora MySQL 的用户权限以及与 Aurora MySQL 数据库集群的 TLS 连接的信息。

主题

- Amazon Aurora MySQL 中的主用户权限
- 与 Aurora MySQL 数据库集群的 TLS 连接

Amazon Aurora MySQL 中的主用户权限

当您创建 Amazon Aurora MySQL 数据库实例时，主用户具有 主用户账户权限 中列出的原定设置权限。

要为每个数据库集群提供管理服务，需要在创建数据库集群时创建 admin 和 rdsadmin 用户。如果试图删掉、重命名、修改 rdsadmin 账户的密码，或者修改该账户的权限，会导致出错。

在 Aurora MySQL 版本 2 数据库集群中，在创建数据库集群时系统会创建 admin 和 rdsadmin 用户。在 Aurora MySQL 版本 3 数据库集群中，系统会创建 admin、rdsadmin 和 rds_superuser_role 用户。

> **Important**
>
> 我们强烈建议不要直接在应用程序中使用主用户。请遵守使用数据库用户的最佳实践，按照您的应用程序所需的最少权限创建用户。

对于 Aurora MySQL 数据库集群管理，已限制标准 `kill` 和 `kill_query` 命令。应使用 Amazon RDS 命令 `rds_kill` 和 `rds_kill_query` 以终止 Aurora MySQL 数据库实例上的用户会话或查询。

> **Note**
>
> 中国 (宁夏) 区域不支持数据库实例和快照加密。

与 Aurora MySQL 数据库集群的 TLS 连接

Amazon Aurora MySQL 数据库集群通过使用与 RDS for MySQL 数据库实例相同的过程和公有密钥，支持从应用程序建立传输层安全性协议（TLS）连接。

在 Amazon RDS 预调配数据库实例时，Amazon RDS 创建 TLS 证书，并将该证书安装在数据库实例上。这些证书由证书颁发机构签署。TLS 证书会将数据库实例端点作为 TLS 证书的公用名（CN）包含在内以防止欺诈攻击。因此，只有在您的客户端支持主题备用名称（SAN）时，您才能使用数据库集群端点通过 TLS 连接到数据库集群。否则，您必须使用写入器实例的实例端点。

有关下载证书的信息，请参阅使用 SSL/TLS 加密与数据库集群的连接。

建议将 AWS JDBC 驱动程序作为支持使用 TLS 的 SAN 的客户端。有关 AWS JDBC 驱动程序的更多信息及其完整使用说明，请参阅 Amazon Web Services (AWS) JDBC Driver GitHub repository。

主题

- 需要与 Aurora MySQL 数据库集群建立 TLS 连接
- Aurora MySQL 的 TLS 版本
- 配置密码套件以连接到 Aurora MySQL 数据库集群
- 加密到 Aurora MySQL 数据库集群的连接

需要与 Aurora MySQL 数据库集群建立 TLS 连接

通过使用 `require_secure_transport` 数据库集群参数，您可以要求与您 Aurora MySQL 数据库集群建立的所有用户连接都使用 TLS。默认情况下，`require_secure_transport` 参数设置为 `OFF`。您可将 `require_secure_transport` 参数设置为 `ON` 以要求使用 TLS 连接到数据库集群。

您可通过更新数据库集群的数据库集群参数组来设置 `require_secure_transport` 参数值。您无需重新启动数据库集群即可使更改生效。有关参数组的更多信息，请参阅Amazon Aurora 的参数组。

> **Note**
>
> `require_secure_transport` 参数适用于 Aurora MySQL 版本 2 和 3。您可以在自定义数据库集群参数组中设置此参数。该参数在数据库实例参数组中不可用。

当数据库集群的 `require_secure_transport` 参数设置为 `ON` 时，如果数据库客户端能够建立加密连接，则可以连接到该数据库集群。否则，将向客户端返回类似于以下内容的错误消息：

```
MySQL Error 3159 (HY000): Connections using insecure transport are prohibited while --require_secure_transport=ON.
```

Aurora MySQL 的 TLS 版本

Aurora MySQL 支持传输层安全性协议（TLS）版本 1.0、1.1、1.2 和 1.3。从 Aurora MySQL 3.04.0 及更高版本开始，您可以使用 TLS 1.3 协议来保护您的连接。下表显示各个 Aurora MySQL 版本的 TLS 支持情况。

Aurora MySQL version	TLS 1.0	TLS 1.1	TLS 1.2	TLS 1.3	默认
Aurora MySQL 版本 2	支持	支持	支持	不支持	所有支持的 TLS 版本
Aurora MySQL 版本 3（低于 3.04.0）	支持	支持	支持	不支持	所有支持的 TLS 版本

Aurora MySQL version	TLS 1.0	TLS 1.1	TLS 1.2	TLS 1.3	默认
Aurora MySQL 版本 3（3.04.0 及更高版本）	不支持	不支持	支持	支持	所有支持的 TLS 版本

Important

如果您为版本 2 和版本低于 3.04.0 的 Aurora MySQL 集群使用自定义参数组，我们建议使用 TLS 1.2，因为 TLS 1.0 和 1.1 不太安全。MySQL 8.0.26 和 Aurora MySQL 3.03 的社区版及其次要版本已不再支持 TLS 版本 1.1 和 1.0。

MySQL 8.0.28 的社区版和兼容的 Aurora MySQL 版本 3.04.0 及更高版本不支持 TLS 1.1 和 TLS 1.0。如果您使用的是 Aurora MySQL 版本 3.04.0 及更高版本，请勿在自定义参数组中将 TLS 协议设置为 1.0 和 1.1。

对于 Aurora MySQL 版本 3.04.0 及更高版本，原定设置为 TLS 1.3 和 TLS 1.2。

您可以使用 `tls_version` 数据库集群参数来指示允许的协议版本。大多数客户端工具或数据库驱动程序都有类似的客户端参数。某些较旧的客户端可能不支持较新的 TLS 版本。默认情况下，数据库集群尝试使用服务器和客户端配置都允许的最高 TLS 协议版本。

将 `tls_version` 数据库集群参数设置为以下值之一：

- `TLSv1.3`
- `TLSv1.2`
- `TLSv1.1`
- `TLSv1`

也可以将 `tls_version` 参数设置为以逗号分隔的列表的字符串。如果您想同时使用 TLS 1.2 和 TLS 1.0 协议，`tls_version` 参数必须包括从最低协议到最高协议的所有协议。在这种情况下，`tls_version` 设置为：

```
tls_version=TLSv1,TLSv1.1,TLSv1.2
```

有关在数据库集群参数组中修改参数的信息，请参阅在 Amazon Aurora 中修改数据库集群参数组中的参数。如果您将 AWS CLI 用于修改 `tls_version` 数据库集群参数，`ApplyMethod` 必须设置为 `pending-reboot`。当应用方法为 `pending-reboot` 时，对参数的更改会在您停止并重新启动与参数组相关的数据库集群之后应用。

配置密码套件以连接到 Aurora MySQL 数据库集群

通过使用可配置的密码套件，您可以更好地控制数据库连接的安全性。您可以指定想要允许保护客户端与数据库的 TLS 连接的密码套件列表。使用可配置的密码套件，您现在可以控制数据库服务器接受的连接加密。这样做可防止使用不安全已或弃用的密码。

Aurora MySQL 版本 3 和 Aurora MySQL 版本 2 支持可配置的密码套件。要指定允许用于加密连接的 TLS 1.2、TLS 1.1、TLS 1.0 密码的列表，请修改 `ssl_cipher` 集群参数。使用 AWS Management Console、AWS CLI 或 RDS API 在集群参数组中设置 `ssl_cipher` 参数。

将 `ssl_cipher` 参数设置为 TLS 版本的以逗号分隔的密码值字符串。对于客户端应用程序，您可以在连接到数据库时使用 `--ssl-cipher` 选项指定用于加密连接的密码。有关连接到数据库的更多信息，请参阅 连接到 Amazon Aurora MySQL 数据库集群。

从 Aurora MySQL 版本 3.04.0 及更高版本开始，您可以指定 TLS 1.3 密码套件。要指定允许的 TLS 1.3 密码套件，请修改参数组中的 `tls_ciphersuites` 参数。TLS 1.3 减少了可用密码套件的数量，这是由于命名约定中进行了相关更改，删除了使用的密钥交换机制和证书。将 `tls_ciphersuites` 设置为 TLS 1.3 以逗号分隔的密码值字符串。

下表显示了支持的密码、TLS 加密协议以及每个密码的有效 Aurora MySQL 引擎版本。

密码	加密协议	支持的 Aurora MySQL 版本
DHE-RSA-AES128-SHA	TLS 1.0	3.01.0 及更高版本，均低于 2.11.0
DHE-RSA-AES128-SHA256	TLS 1.2	3.01.0 及更高版本，均低于 2.11.0
DHE-RSA-AES128-GCM-SHA256	TLS 1.2	3.01.0 及更高版本，均低于 2.11.0

密码	加密协议	支持的 Aurora MySQL 版本
DHE-RSA-AES256-SHA	TLS 1.0	3.03.0 及更低版本，所有低于 2.11.0 的版本
DHE-RSA-AES256-SHA256	TLS 1.2	3.01.0 及更高版本，均低于 2.11.0
DHE-RSA-AES256-GCM-SHA384	TLS 1.2	3.01.0 及更高版本，均低于 2.11.0
ECDHE-RSA-AES128-SHA	TLS 1.0	3.01.0 及更高版本、2.09.3 及更高版本、2.10.2 及更高版本
ECDHE-RSA-AES128-SHA256	TLS 1.2	3.01.0 及更高版本、2.09.3 及更高版本、2.10.2 及更高版本
ECDHE-RSA-AES128-GCM-SHA256	TLS 1.2	3.01.0 及更高版本、2.09.3 及更高版本、2.10.2 及更高版本
ECDHE-RSA-AES256-SHA	TLS 1.0	3.01.0 及更高版本、2.09.3 及更高版本、2.10.2 及更高版本
ECDHE-RSA-AES256-SHA384	TLS 1.2	3.01.0 及更高版本、2.09.3 及更高版本、2.10.2 及更高版本
ECDHE-RSA-AES256-GCM-SHA384	TLS 1.2	3.01.0 及更高版本、2.09.3 及更高版本、2.10.2 及更高版本
TLS_AES_128_GCM_SHA256	TLS 1.3	3.04.0 及更高版本
TLS_AES_256_GCM_SHA384	TLS 1.3	3.04.0 及更高版本
TLS_CHACHA20_POLY1305_SHA256	TLS 1.3	3.04.0 及更高版本

> **Note**
>
> 只有 2.11.0 之前的 Aurora MySQL 版本才支持 DHE-RSA 密码。2.11.0 及更高版本仅支持 ECDHE 密码。

有关在数据库集群参数组中修改参数的信息，请参阅在 Amazon Aurora 中修改数据库集群参数组中的参数。如果您使用 CLI 来修改 ssl_cipher 数据库集群参数，请务必将 ApplyMethod 设置为 pending-reboot。当应用方法为 pending-reboot 时，对参数的更改会在您停止并重新启动与参数组相关的数据库集群之后应用。

您也可以使用 describe-engine-default-cluster-parameters CLI 命令来确定特定参数组系列当前支持哪些密码套件。以下示例展示如何获取 Aurora MySQL 版本 2 的 ssl_cipher 集群参数允许的值。

```
aws rds describe-engine-default-cluster-parameters --db-parameter-group-family aurora-
mysql5.7

        ...some output truncated...
 {
  "ParameterName": "ssl_cipher",
  "ParameterValue": "DHE-RSA-AES128-SHA,DHE-RSA-AES128-SHA256,DHE-RSA-AES128-GCM-
SHA256,DHE-RSA-AES256-SHA,DHE-RSA-AES256-SHA256,DHE-RSA-AES256-GCM-SHA384,ECDHE-RSA-
AES128-SHA,ECDHE-RSA-AES128-SHA256,ECDHE-RSA-AES128-GCM-SHA256,ECDHE-RSA-AES256-
SHA,ECDHE-RSA-AES256-SHA384,ECDHE-RSA-AES256-GCM-SHA384",
  "Description": "The list of permissible ciphers for connection encryption.",
  "Source": "system",
  "ApplyType": "static",
  "DataType": "list",
  "AllowedValues": "DHE-RSA-AES128-SHA,DHE-RSA-AES128-SHA256,DHE-RSA-AES128-GCM-
SHA256,DHE-RSA-AES256-SHA,DHE-RSA-AES256-SHA256,DHE-RSA-AES256-GCM-SHA384,ECDHE-
RSA-AES128-SHA,ECDHE-RSA-AES128-SHA256,ECDHE-RSA-AES128-GCM-SHA256,ECDHE-RSA-AES256-
SHA,ECDHE-RSA-AES256-SHA384,ECDHE-RSA-AES256-GCM-SHA384",
  "IsModifiable": true,
  "SupportedEngineModes": [
   "provisioned"
  ]
 },
       ...some output truncated...
```

有关密码的更多信息，请参阅 MySQL 文档中的 ssl_cipher 变量。有关密码套件格式的更多信息，请参阅 OpenSSL 网站上的 openssl-ciphers list format 和 openssl-ciphers string format 文档。

加密到 Aurora MySQL 数据库集群的连接

要使用默认的 mysql 客户端对连接加密，需用 --ssl-ca 参数启动 mysql 客户端以便引用公有密钥，例如：

对于 MySQL 5.7 和 8.0：

```
mysql -h myinstance.123456789012.rds-us-east-1.amazonaws.com
--ssl-ca=full_path_to_CA_certificate --ssl-mode=VERIFY_IDENTITY
```

对于 MySQL 5.6：

```
mysql -h myinstance.123456789012.rds-us-east-1.amazonaws.com
--ssl-ca=full_path_to_CA_certificate --ssl-verify-server-cert
```

将 *full_path_to_CA_certificate* 替换为证书颁发机构 (CA) 证书的完整路径。有关下载证书的信息，请参阅 使用 SSL/TLS 加密与数据库集群的连接。

您可以要求特定用户账户建立 TLS 连接。例如，可以根据您的 MySQL 版本，使用以下语句之一来要求用户账户 encrypted_user 建立 TLS 连接。

对于 MySQL 5.7 和 8.0：

```
ALTER USER 'encrypted_user'@'%' REQUIRE SSL;
```

对于 MySQL 5.6：

```
GRANT USAGE ON *.* TO 'encrypted_user'@'%' REQUIRE SSL;
```

使用 RDS 代理时，您可以连接到代理端点，而不是通常的集群端点。对于到代理的连接，您可以像直接到 Aurora 数据库集群的连接那样，使 SSL/TLS 成为必需或可选的选项。有关使用 RDS 代理的信息，请参阅适用于 Aurora 的 Amazon RDS 代理。

> **Note**
>
> 有关与 MySQL 的 TLS 连接的更多信息，请参阅 MySQL 文档。

使用新的 TLS 证书更新应用程序，以连接到 Aurora MySQL 数据库集群

自 2023 年 1 月 13 日起，Amazon RDS 发布了新的证书颁发机构（CA）证书，以便使用传输层安全性协议（TLS）连接到 Aurora 数据库集群。接下来，您可以找到有关更新应用程序以使用新证书的信息。

本主题可帮助您确定是否有任何客户端应用程序使用 TLS 连接到您的数据库集群。如果是这样，您可以进一步检查这些应用程序是否需要证书验证才能连接。

> **Note**
>
> 某些应用程序被配置为仅在它们可以成功验证服务器上的证书时才连接到 Aurora MySQL 数据库集群。
> 对于此类应用程序，您必须更新客户端应用程序信任存储，以包括新的 CA 证书。

更新客户端应用程序信任存储中的 CA 证书后，可以在数据库集群上轮换这些证书。强烈建议在生产环境中实现这些过程之前，先在开发或测试环境中测试它们。

有关证书轮换的更多信息，请参阅轮换 SSL/TLS 证书。有关下载证书的更多信息，请参阅使用 SSL/TLS 加密与数据库集群的连接。有关将 TLS 用于 Aurora MySQL 数据库集群的信息，请参阅与 Aurora MySQL 数据库集群的 TLS 连接。

主题

- 确定是否有任何应用程序正使用 TLS 连接到 Aurora MySQL 数据库集群
- 确定客户端是否需要证书验证才能连接
- 更新应用程序信任存储
- 用于建立 TLS 连接的示例 Java 代码

确定是否有任何应用程序正使用 TLS 连接到 Aurora MySQL 数据库集群

如果您使用的是 Aurora MySQL 版本 2（与 MySQL 5.7 兼容）并且启用了性能架构，则运行以下查询以检查连接是否正在使用 TLS。有关启用性能架构的信息，请参阅 MySQL 文档中的性能架构快速入门。

```
mysql> SELECT id, user, host, connection_type
```

```
        FROM performance_schema.threads pst
        INNER JOIN information_schema.processlist isp
        ON pst.processlist_id = isp.id;
```

在该示例输出中，您可以看到自己的会话（admin）以及作为 webapp1 登录的应用程序均在使用 TLS。

```
+----+-----------------+------------------+-----------------+
| id | user            | host             | connection_type |
+----+-----------------+------------------+-----------------+
|  8 | admin           | 10.0.4.249:42590 | SSL/TLS         |
|  4 | event_scheduler | localhost        | NULL            |
| 10 | webapp1         | 159.28.1.1:42189 | SSL/TLS         |
+----+-----------------+------------------+-----------------+
3 rows in set (0.00 sec)
```

确定客户端是否需要证书验证才能连接

您可以检查 JDBC 客户端和 MySQL 客户端是否需要证书验证才能连接。

JDBC

以下使用 MySQL Connector/J 8.0 的示例显示了一种方法，用于检查应用程序的 JDBC 连接属性以确定成功的连接是否需要有效证书。有关 MySQL 的所有 JDBC 连接选项的更多信息，请参阅 MySQL 文档中的配置属性。

当使用 MySQL Connector/J 8.0 时，如果您的连接属性将 sslMode 设置为 VERIFY_CA 或 VERIFY_IDENTITY，则 TLS 连接需要对服务器 CA 证书进行验证，如以下示例所示。

```
Properties properties = new Properties();
properties.setProperty("sslMode", "VERIFY_IDENTITY");
properties.put("user", DB_USER);
properties.put("password", DB_PASSWORD);
```

> **Note**
>
> 如果您使用 MySQL Java Connector v5.1.38 或更高版本或者 MySQL Java Connector v8.0.9 或更高版本连接到数据库，即使您没有明确配置应用程序在连接到数据库时使用 TLS，这些客户端驱动程序仍默认为使用 TLS。此外，在使用 TLS 时，它们会执行部分证书验证，如果数据库服务器证书过期，则无法连接。

MySQL

以下使用 MySQL 客户端的示例显示了两种方法，用于检查脚本的 MySQL 连接以确定成功的连接是否需要有效证书。有关 MySQL 客户端的所有连接选项的更多信息，请参阅 MySQL 文档中的加密连接的客户端配置。

当使用 MySQL 5.7 或 MySQL 8.0 客户端时，如果对于 `--ssl-mode` 选项，您指定 `VERIFY_CA` 或 `VERIFY_IDENTITY`，则 TLS 连接需要对服务器 CA 证书进行验证，如以下示例所示。

```
mysql -h mysql-database.rds.amazonaws.com -uadmin -ppassword --ssl-ca=/tmp/ssl-cert.pem
 --ssl-mode=VERIFY_CA
```

当使用 MySQL 5.6 客户端时，如果您指定 `--ssl-verify-server-cert` 选项，则 SSL 连接需要对服务器 CA 证书进行验证，如以下示例所示。

```
mysql -h mysql-database.rds.amazonaws.com -uadmin -ppassword --ssl-ca=/tmp/ssl-cert.pem
 --ssl-verify-server-cert
```

更新应用程序信任存储

有关更新 MySQL 应用程序的信任存储的信息，请参阅 MySQL 文档中的安装 SSL 证书。

> **Note**
>
> 更新信任存储时，除了添加新证书外，还可以保留较旧证书。

更新 JDBC 应用程序信任存储

您可以更新使用 JDBC 的应用程序的信任存储以进行 TLS 连接。

有关下载根证书的信息，请参阅 使用 SSL/TLS 加密与数据库集群的连接。

有关导入证书的示例脚本，请参阅 将证书导入信任存储的示例脚本。

如果在应用程序中使用 mysql JDBC 驱动程序，请在该应用程序中设置以下属性。

```
System.setProperty("javax.net.ssl.trustStore", certs);
System.setProperty("javax.net.ssl.trustStorePassword", "password");
```

> **Note**
>
> 作为安全最佳实践，请指定除此处所示提示以外的密码。

启动应用程序时，请设置以下属性。

```
java -Djavax.net.ssl.trustStore=/path_to_truststore/MyTruststore.jks -
Djavax.net.ssl.trustStorePassword=my_truststore_password com.companyName.MyApplication
```

用于建立 TLS 连接的示例 Java 代码

以下代码示例展示了如何设置使用 JDBC 验证服务器证书的 SSL 连接。

```
public class MySQLSSLTest {

        private static final String DB_USER = "user name";
        private static final String DB_PASSWORD = "password";
        // This key store has only the prod root ca.
        private static final String KEY_STORE_FILE_PATH = "file-path-to-keystore";
        private static final String KEY_STORE_PASS = "keystore-password";

    public static void test(String[] args) throws Exception {
        Class.forName("com.mysql.jdbc.Driver");


        System.setProperty("javax.net.ssl.trustStore", KEY_STORE_FILE_PATH);
        System.setProperty("javax.net.ssl.trustStorePassword", KEY_STORE_PASS);

        Properties properties = new Properties();
        properties.setProperty("sslMode", "VERIFY_IDENTITY");
        properties.put("user", DB_USER);
        properties.put("password", DB_PASSWORD);

        Connection connection = DriverManager.getConnection("jdbc:mysql://jagdeeps-ssl-
test.cni62e2e7kwh.us-east-1.rds.amazonaws.com:3306",properties);
        Statement stmt=connection.createStatement();

        ResultSet rs=stmt.executeQuery("SELECT 1 from dual");

        return;
    }
```

```
}
```

Important

在确定了数据库连接使用 TLS 并更新了应用程序信任存储之后，可以更新数据库以使用 rds-ca-rsa2048-g1 证书。有关说明，请参阅通过修改数据库实例来更新 CA 证书中的步骤 3。

对 Aurora MySQL 使用 Kerberos 身份验证

当用户连接到 Aurora MySQL 数据库集群时，您可以使用 Kerberos 身份验证来验证用户的身份。为此，请将数据库集群配置为使用 AWS Directory Service for Microsoft Active Directory 进行 Kerberos 身份验证。AWS Directory Service for Microsoft Active Directory 也称为 AWS Managed Microsoft AD。这是 AWS Directory Service 提供的一项功能。要了解更多信息，请参阅《AWS Directory Service 管理指南》中的什么是 AWS Directory Service？

要开始操作，请创建一个 AWS Managed Microsoft AD 目录来存储用户凭证。然后，将 Active Directory 的域和其他信息提供给 Aurora MySQL 数据库集群。当用户向 Aurora MySQL 数据库集群验证身份时，身份验证请求将转发到 AWS Managed Microsoft AD 目录。

将所有凭证保存在同一目录中可以节省您的时间和精力。使用这种方法，您具有一个集中位置用于存储和管理多个数据库集群的凭证。使用目录还可以改善您的整体安全概要。

此外，还可以从自己的本地 Microsoft Active Directory 访问凭证。为此，请创建一个信任域关系，以便 AWS Managed Microsoft AD 目录信任您的本地 Microsoft Active Directory。通过这种方式，用户可以使用 Windows 单点登录（SSO）访问 Aurora MySQL 数据库集群，获得与访问本地网络中的工作负载相同的体验。

数据库可以使用 Kerberos、AWS Identity and Access Management (IAM)，或同时使用 Kerberos 和 IAM 身份验证。但是，由于 Kerberos 和 IAM 身份验证提供了不同的身份验证方法，因此，特定用户只能使用一种或另一种身份验证方法登录数据库，但不能同时使用这两种方法。有关 IAM 身份验证的更多信息，请参阅 IAM 数据库身份验证。

目录

- Aurora MySQL 数据库集群的 Kerberos 身份验证概述
- Aurora MySQL 的 Kerberos 身份验证限制
- 为 Aurora MySQL 数据库集群设置 Kerberos 身份验证
  - 步骤 1：使用 AWS Managed Microsoft AD 创建目录
  - 步骤 2：（可选）为本地 Active Directory 创建信任
  - 步骤 3：创建 IAM 角色以供 Amazon Aurora 使用
  - 步骤 4：创建和配置用户
  - 步骤 5：创建或修改 Aurora MySQL 数据库集群
  - 步骤 6：创建使用 Kerberos 身份验证的 Aurora MySQL 用户
    - 修改现有的 Aurora MySQL 登录名

- 步骤 7：配置 MySQL 客户端
- 步骤 8：（可选）配置不区分大小写的用户名比较
- 使用 Kerberos 身份验证连接到 Aurora MySQL
  - 使用 Aurora MySQL Kerberos 登录名连接到数据库集群
  - Aurora 全局数据库的 Kerberos 身份验证
  - 从 RDS for MySQL 迁移到 Aurora MySQL
  - 阻止票证缓存
  - 针对 Kerberos 身份验证的日志记录
- 在域中管理数据库集群
  - 了解域成员资格

Aurora MySQL 数据库集群的 Kerberos 身份验证概述

要为 Aurora MySQL 数据库集群设置 Kerberos 身份验证，请完成以下一般步骤。这些步骤将在后面进行更详细的描述。

1. 使用 AWS Managed Microsoft AD 创建 AWS Managed Microsoft AD 目录。您可以使用 AWS Management Console、AWS CLI 或 AWS Directory Service 创建目录。有关详细说明，请参阅《AWS Directory Service 管理指南》中的创建 AWS Managed Microsoft AD 目录。
2. 创建使用托管式 IAM 策略 `AmazonRDSDirectoryServiceAccess` 的 AWS Identity and Access Management（IAM）角色。此角色允许 Amazon Aurora 调用您的目录。

   为了让角色允许访问，AWS Security Token Service（AWS STS）端点必须在您的 AWS 账户的 AWS 区域中激活。AWS STS 端点原定设置为在所有 AWS 区域中保持活跃状态，且您无需任何进一步动作即可使用这些端点。有关更多信息，请参阅《IAM 用户指南》中的在 AWS 区域中激活和停用 AWS STS。
3. 使用 Microsoft Active Directory 工具在 AWS Managed Microsoft AD 目录中创建和配置用户。有关在 Active Directory 中创建用户的更多信息，请参阅 AWS 管理指南中的在 AWS Directory Service 托管式 Microsoft AD 中管理用户和组。
4. 创建或修改 Aurora MySQL 数据库集群。如果您在创建请求中使用 CLI 或 RDS API，请使用 `Domain` 参数指定域标识符。使用在创建目录时生成的 `d-*` 标识符和您创建的 IAM 角色的名称。

   如果您将现有 Aurora MySQL 数据库集群修改为使用 Kerberos 身份验证，请为数据库集群设置域和 IAM 角色参数。在与域目录相同的 VPC 中查找数据库集群。

5. 使用 Amazon RDS 主用户凭证连接到 Aurora MySQL 数据库集群。按照步骤 6：创建使用 Kerberos 身份验证的 Aurora MySQL 用户中的说明在 Aurora MySQL 中创建数据库用户。

   您以此方式创建的用户可以使用 Kerberos 身份验证登录到 Aurora MySQL 数据库集群。有关更多信息，请参阅 使用 Kerberos 身份验证连接到 Aurora MySQL。

要将 Kerberos 身份验证与本地或自托管式 Microsoft Active Directory 结合使用，请创建林信任。林信任是两组域之间的信任关系。信任可以是单向或双向的。有关使用 AWS Directory Service 设置林信任的更多信息，请参阅 AWS Directory Service 管理指南 中的何时创建信任关系。

Aurora MySQL 的 Kerberos 身份验证限制

以下限制适用于 Aurora MySQL 的 Kerberos 身份验证：

- Aurora MySQL 版本 3.03 及更高版本支持 Kerberos 身份验证。

  有关 AWS 区域支持的信息，请参阅适用于 Aurora MySQL 的 Kerberos 身份验证。
- 要将 Kerberos 身份验证与 Aurora MySQL 结合使用，MySQL 客户端或连接器必须在 Unix 平台上使用版本 8.0.26 或更高版本，在 Windows 上使用版本 8.0.27 或更高版本。否则，客户端 `authentication_kerberos_client` 插件不可用，您无法进行身份验证。
- 在 Aurora MySQL 上仅支持 AWS Managed Microsoft AD。但是，您可以将 Aurora MySQL 数据库集群加入到同一 AWS 区域中不同账户拥有的共享 Managed Microsoft AD 域。

  您还可以使用自己的本地 Active Directory。有关更多信息，请参阅 步骤 2：（可选）为本地 Active Directory 创建信任。
- 当使用 Kerberos 对从 MySQL 客户端或从 Windows 操作系统上的驱动程序连接到 Aurora MySQL 集群的用户进行身份验证时，原定设置情况下，数据库用户名的字符大小写必须与 Active Directory 中用户的大小写相匹配。例如，如果 Active Directory 中的用户显示为 `Admin`，则数据库用户名必须为 `Admin`。

  但是，您现在可以通过 `authentication_kerberos` 插件使用不区分大小写的用户名比较。有关更多信息，请参阅 步骤 8：（可选）配置不区分大小写的用户名比较。
- 开启该功能后，必须重启读取器数据库实例才能安装 `authentication_kerberos` 插件。
- 复制到不支持 `authentication_kerberos` 插件的数据库实例可能会导致复制失败。
- 要让 Aurora 全局数据库使用 Kerberos 身份验证，您必须为全局数据库中的每个数据库集群配置该身份验证。
- 域名长度必须小于 62 个字符。

- 开启 Kerberos 身份验证后，请勿修改数据库集群端口。如果您修改此端口，则 Kerberos 身份验证将不再起作用。

为 Aurora MySQL 数据库集群设置 Kerberos 身份验证

使用 AWS Managed Microsoft AD 为 Aurora MySQL 数据库集群设置 Kerberos 身份验证。要设置 Kerberos 身份验证，请执行以下步骤。

主题

- 步骤 1：使用 AWS Managed Microsoft AD 创建目录
- 步骤 2：（可选）为本地 Active Directory 创建信任
- 步骤 3：创建 IAM 角色以供 Amazon Aurora 使用
- 步骤 4：创建和配置用户
- 步骤 5：创建或修改 Aurora MySQL 数据库集群
- 步骤 6：创建使用 Kerberos 身份验证的 Aurora MySQL 用户
- 步骤 7：配置 MySQL 客户端
- 步骤 8：（可选）配置不区分大小写的用户名比较

步骤 1：使用 AWS Managed Microsoft AD 创建目录

AWS Directory Service 将在 AWS 云中创建一个完全托管的 Active Directory。创建 AWS Managed Microsoft AD 目录时，AWS Directory Service 将代表您创建两个域控制器和域名系统 (DNS) 服务器。目录服务器在 VPC 中的不同子网中创建。这种冗余有助于确保始终可以访问目录，即使发生了故障。

创建 AWS Managed Microsoft AD 目录时，AWS Directory Service 代表您执行以下任务：

- 在 VPC 中设置 Active Directory。
- 创建具有用户名 `Admin` 和指定密码的目录管理员账户。您可以使用此账户管理您的目录。

> **Note**
>
> 请务必保存此密码。AWS Directory Service 不会存储它。您可以重置它，但无法检索它。

- 为目录控制器创建安全组。

在启动 AWS Managed Microsoft AD 时，AWS 创建一个组织单位 (OU)，其中包含目录的所有对象。此 OU 具有您在创建目录时输入的 NetBIOS 名称。此 OU 位于域根目录中，由 AWS 拥有和管理。

使用 AWS Managed Microsoft AD 目录创建的 Admin 账户有权为您的 OU 执行最常见的管理活动，包括：

- 创建、更新或删除用户
- 将资源添加到域（如文件或打印服务器），然后为 OU 中的用户分配这些资源的权限
- 创建额外的 OU 和容器
- 委托授权
- 从 Active Directory 回收站还原删除的对象
- 在 Active Directory Web 服务上运行 AD 和 DNS Windows PowerShell 模块

Admin 账户还有权执行以下域范围的活动：

- 管理 DNS 配置（添加、删除或更新记录、区域和转发器）
- 查看 DNS 事件日志
- 查看安全事件日志

使用 AWS Managed Microsoft AD 创建目录

1. 登录 AWS Management Console，然后打开 AWS Directory Service 控制台，网址为：https://console.aws.amazon.com/directoryservicev2/。
2. 在导航窗格中，选择 Directories (目录)，然后选择 Set up Directory (设置目录)。
3. 选择 AWS Managed Microsoft AD。AWS Managed Microsoft AD 是当前唯一可以与 Amazon RDS 一起使用的选项。
4. 输入以下信息：

   目录 DNS 名称

   目录的完全限定名称，例如 **corp.example.com**。

   目录 NetBIOS 名称

   目录的短名称，如 **CORP**。

目录描述

（可选）目录的描述。

管理员密码

目录管理员的密码。目录创建过程将使用用户名 Admin 和此密码创建一个管理员账户。

目录管理员密码不能包含单词“admin”。此密码区分大小写，且长度必须介于 8 – 64 个字符之间。至少，它还必须包含下列四种类别中三种类别的一个字符：

- 小写字母 (a–z)
- 大写字母 (A–Z)
- 数字 (0–9)
- 非字母数字字符 (~!@#$%^&*_-+=`|\(){}[]:;"'<>,.?/)

确认密码

重新输入的管理员密码。

5. 选择下一步。
6. 在 Networking (网络) 部分中输入以下信息，然后选择 Next (下一步)：

   VPC

   目录的 VPC。在该同一 VPC 中创建 Aurora MySQL 数据库集群。

   子网

   目录服务器的子网。两个子网必须位于不同的可用区。
7. 查看目录信息并进行必要的更改。如果信息正确，请选择 Create directory (创建目录)。

创建目录需要几分钟时间。创建成功后，Status (状态) 值将更改为 Active (活动)。

要查看有关您的目录的信息，请在目录列表中选择目录名称。请记下目录 ID 值，因为您在创建或修改 Aurora MySQL 数据库集群时需要此值。

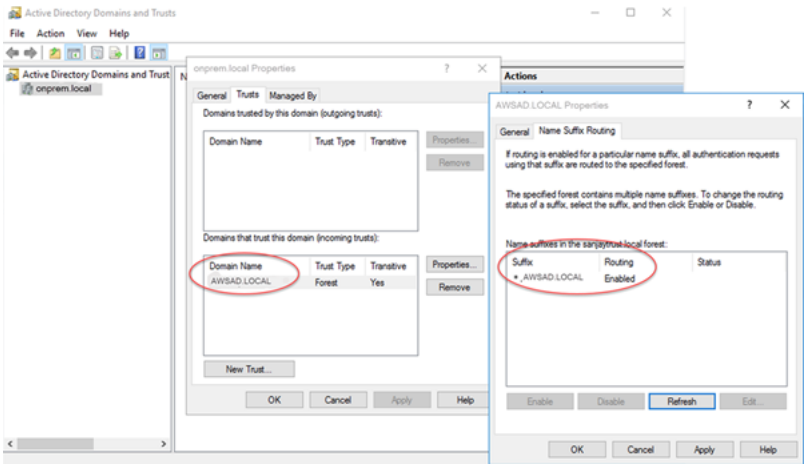
步骤 2：（可选）为本地 Active Directory 创建信任

如果您不打算使用自己的本地 Microsoft Active Directory，请跳转至 步骤 3：创建 IAM 角色以供 Amazon Aurora 使用。

要将 Kerberos 身份验证与本地 Active Directory 结合使用，您需要使用林信任在本地 Microsoft Active Directory 和 AWS Managed Microsoft AD 目录（在步骤 1：使用 AWS Managed Microsoft AD 创建目录中创建）之间创建信任域关系。信任可以是单向的，此时 AWS Managed Microsoft AD 目录信任本地 Microsoft Active Directory。信任也可以是双向的，此时两个 Active Directory 相互信任。有关使用 AWS Directory Service 设置信任的更多信息，请参阅 AWS Directory Service 管理指南中的何时创建信任关系。

> **Note**
>
> 如果使用本地 Microsoft Active Directory：
>
> - Windows 客户端无法使用 Aurora 自定义端点进行连接。要了解更多信息，请参阅Amazon Aurora 端点连接。
> - 对于全局数据库：
>   - Windows 客户端可以使用全局数据库的主 AWS 区域 中的实例端点或集群端点进行连接。
>   - Windows 客户端无法使用辅助 AWS 区域 中的集群端点进行连接。

请确保您的本地 Microsoft Active Directory 域名包含与新创建的信任关系对应的 DNS 后缀路由。以下屏幕截图显示一个示例。



步骤 3：创建 IAM 角色以供 Amazon Aurora 使用

要使 Amazon Aurora 为您调用 AWS Directory Service，您需要一个使用托管式 IAM policy `AmazonRDSDirectoryServiceAccess` 的 AWS Identity and Access Management（IAM）角色。该角色允许 Aurora 调用 AWS Directory Service。

使用 AWS Management Console创建数据库集群并且您具有 `iam:CreateRole` 权限时，控制台将自动创建此角色。在这种情况下，角色名为 `rds-directoryservice-kerberos-access-role`。否则，您必须手动创建 IAM 角色。在创建该 IAM 角色时，请选择 `Directory Service`，然后将 AWS 托管策略 `AmazonRDSDirectoryServiceAccess` 附加到该角色。

有关为服务创建 IAM 角色的更多信息，请参阅 IAM 用户指南中的创建向AWS服务委托权限的角色。

或者，您可以创建具有所需权限的角色，而不是使用托管 IAM 策略 `AmazonRDSDirectoryServiceAccess`。在这种情况下，IAM 角色必须具有以下 IAM 信任策略。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "directoryservice.rds.amazonaws.com",
          "rds.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

角色还必须具有以下 IAM 角色策略。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "ds:DescribeDirectories",
        "ds:AuthorizeApplication",
```

```
        "ds:UnauthorizeApplication",
        "ds:GetAuthorizedApplicationDetails"
      ],
    "Effect": "Allow",
    "Resource": "*"
    }
  ]
}
```

步骤 4：创建和配置用户

您可以使用“Active Directory 用户和计算机”工具创建用户。该工具是 Active Directory Domain Services 和 Active Directory Lightweight Directory Services 工具的一部分。用户表示有权访问您的目录的独立个人或实体。

要在 AWS Directory Service 目录中创建用户，请使用一个基于 Microsoft Windows 的本地或 Amazon EC2 实例，该实例已加入您的 AWS Directory Service 目录中。您必须以具有权限创建用户的用户身份登录此实例。有关更多信息，请参阅AWS Managed Microsoft AD Directory Service 管理指南中的管理 AWS 中的用户和组。

步骤 5：创建或修改 Aurora MySQL 数据库集群

创建或修改 Aurora MySQL 数据库集群，以便与目录一起使用。您可以使用控制台、AWS CLI 或 RDS API 将数据库集群与目录关联。您可以通过下列方式之一来执行该任务：

- 使用控制台、create-db-cluster CLI 命令或 CreateDBCluster RDS API 操作创建新的 Aurora MySQL 数据库集群。

  有关说明，请参阅创建 Amazon Aurora 数据库集群。
- 使用控制台、modify-db-cluster CLI 命令或 ModifyDBCluster RDS API 操作修改现有的 Aurora MySQL 数据库集群。

  有关说明，请参阅修改 Amazon Aurora 数据库集群。
- 使用控制台、restore-db-cluster-from-snapshot CLI 命令或 RestoreDBClusterFromSnapshot RDS API 操作，从数据库快照还原 Aurora MySQL 数据库集群。

  有关说明，请参阅从数据库集群快照还原。
- 使用控制台、restore-db-cluster-to-point-in-time CLI 命令或 RestoreDBClusterToPointInTime RDS API 操作，将 Aurora MySQL 数据库集群还原到某个时间点。

有关说明，请参阅将数据库集群还原到指定时间。

仅 VPC 中的 Aurora MySQL 数据库集群支持 Kerberos 身份验证。数据库集群可以与目录在同一 VPC 中或在不同 VPC 中。数据库集群的 VPC 必须具有允许与您的目录进行出站通信的 VPC 安全组。

控制台

在使用控制台创建、修改或还原数据库集群时，请选择数据库身份验证部分中的 Kerberos 身份验证。选择 Browse Directory (浏览目录)，然后选择目录或选择 Create a new directory (创建新目录)。

AWS CLI

使用 AWS CLI 或 RDS API 时，将数据库集群实例与目录关联。数据库集群需要以下参数才能使用您创建的域目录：

- 对于 `--domain` 参数，请使用创建目录时生成的域标识符（“d-*”标识符）。
- 对于 `--domain-iam-role-name` 参数，请使用您使用托管 IAM 策略 `AmazonRDSDirectoryServiceAccess` 创建的角色。

例如，以下 CLI 命令修改数据库集群以使用目录。

对于 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --domain d-ID \
    --domain-iam-role-name role-name
```

对于 Windows：

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --domain d-ID ^
    --domain-iam-role-name role-name
```

> **Important**
>
> 如果您修改数据库集群以开启 Kerberos 身份验证，请在进行更改之后重启读取器数据库实例。

步骤 6：创建使用 Kerberos 身份验证的 Aurora MySQL 用户

数据库集群加入到 AWS Managed Microsoft AD 域。这样，您可以从您域中的 Active Directory 用户创建 Aurora MySQL 用户。数据库权限是通过授予这些用户并从这些用户撤消的标准 Aurora MySQL 权限来管理的。

您可以允许 Active Directory 用户使用 Aurora MySQL 进行身份验证。为此，首先使用 Amazon RDS 主用户凭证连接到 Aurora MySQL 数据库集群，就像连接到任何其他数据库集群一样。登录后，在 Aurora MySQL 中创建一个具有 Kerberos 身份验证的经外部身份验证的用户，如下所示：

```
CREATE USER user_name@'host_name' IDENTIFIED WITH 'authentication_kerberos' BY
 'realm_name';
```

- 将 *user_name* 替换为用户名。您的域中的用户（人员和应用程序）现在可以使用 Kerberos 身份验证从加入域的客户端计算机连接到数据库集群。
- 将 *host_name* 替换为主机名。可以使用 % 作为通配符。您也可以使用特定的 IP 地址作为主机名。
- 将 *realm_name* 替换为域的目录领域名称。领域名称通常与使用大写字母的 DNS 域名相同，例如 `CORP.EXAMPLE.COM`。领域是一组使用相同 Kerberos 密钥分发中心的系统。

以下示例创建了一个名为 `Admin` 的数据库用户，该用户使用领域名称 `MYSQL.LOCAL` 针对 Active Directory 进行身份验证。

```
CREATE USER Admin@'%' IDENTIFIED WITH 'authentication_kerberos' BY 'MYSQL.LOCAL';
```

修改现有的 Aurora MySQL 登录名

还可以使用以下语法修改现有 Aurora MySQL 登录名以使用 Kerberos 身份验证：

```
ALTER USER user_name IDENTIFIED WITH 'authentication_kerberos' BY 'realm_name';
```

步骤 7：配置 MySQL 客户端

要配置 MySQL 客户端，请采取以下步骤：

1. 创建一个 `krb5.conf` 文件（或等效的文件）以指向该域。
2. 验证流量是否可以在客户端主机和 AWS Directory Service 之间流动。使用网络实用程序（如 Netcat）执行以下操作：
   - 验证端口 53 上通过 DNS 的流量。
   - 验证端口 53 上通过 TCP/UDP 的流量以及 Kerberos 的流量，包括用于 AWS Directory Service 的端口 88 和 464。
3. 验证流量是否可以通过数据库端口在客户端主机和数据库实例之间流动。例如，使用 `mysql` 连接和访问数据库。

以下是 AWS Managed Microsoft AD 的示例 `krb5.conf` 内容。

```
[libdefaults]
 default_realm = EXAMPLE.COM
[realms]
 EXAMPLE.COM = {
  kdc = example.com
  admin_server = example.com
 }
[domain_realm]
 .example.com = EXAMPLE.COM
 example.com = EXAMPLE.COM
```

以下是本地 Microsoft Active Directory 的示例 `krb5.conf` 内容。

```
[libdefaults]
 default_realm = EXAMPLE.COM
[realms]
 EXAMPLE.COM = {
  kdc = example.com
  admin_server = example.com
 }
 ONPREM.COM = {
  kdc = onprem.com
  admin_server = onprem.com
 }
```

```
[domain_realm]
 .example.com = EXAMPLE.COM
 example.com = EXAMPLE.COM
 .onprem.com = ONPREM.COM
 onprem.com = ONPREM.COM
 .rds.amazonaws.com = EXAMPLE.COM
 .amazonaws.com.cn = EXAMPLE.COM
 .amazon.com = EXAMPLE.COM
```

步骤 8：（可选）配置不区分大小写的用户名比较

原定设置情况下，MySQL 数据库用户名的字符大小写必须与 Active Directory 登录名的大小写匹配。但是，您现在可以通过 `authentication_kerberos` 插件使用不区分大小写的用户名比较。为此，可以将 `authentication_kerberos_caseins_cmp` 数据库集群参数设置为 `true`。

使用不区分大小写的用户名比较

1. 创建自定义数据库集群参数组。按照 在 Amazon Aurora 中创建数据库集群参数组 所述的过程操作。
2. 编辑新的参数组以将 `authentication_kerberos_caseins_cmp` 的值设置为 `true`。按照 在 Amazon Aurora 中修改数据库集群参数组中的参数 所述的过程操作。
3. 将数据库集群参数组与 Aurora MySQL 数据库集群关联。按照 在 Amazon Aurora 中将数据库集群参数组与数据库集群关联 所述的过程操作。
4. 重启数据库集群。

使用 Kerberos 身份验证连接到 Aurora MySQL

为避免错误，请在 Unix 平台上使用版本 8.0.26 或更高版本的 MySQL 客户端，在 Windows 上使用 8.0.27 或更高版本的 MySQL 客户端。

使用 Aurora MySQL Kerberos 登录名连接到数据库集群

要通过 Kerberos 身份验证连接到 Aurora MySQL，您需要按照 步骤 6：创建使用 Kerberos 身份验证的 Aurora MySQL 用户 中的说明，以您创建的数据库用户身份登录。

在命令提示符下，连接到其中一个与 Aurora MySQL 数据库集群关联的端点。当系统提示您输入密码时，请输入与该用户名关联的 Kerberos 密码。

当您使用 Kerberos 进行身份验证时，如果票证授予票证（TGT）尚不存在，则会生成此票证。`authentication_kerberos` 插件使用 TGT 获取服务票证，然后将其出示给 Aurora MySQL 数据库服务器。

可以使用 MySQL 客户端，在 Windows 或 Unix 上通过 Kerberos 身份验证连接到 Aurora MySQL。

Unix

您可以使用以下方法之一进行连接：

- 手动获取 TGT。在这种情况下，您不需要向 MySQL 客户端提供密码。
- 直接向 MySQL 客户端提供 Active Directory 登录的密码。

在 Unix 平台上，MySQL 客户端版本 8.0.26 及更高版本支持客户端插件。

通过手动获取 TGT 进行连接

1. 在命令行界面上，使用以下命令获取 TGT。

```
kinit user_name
```

2. 使用以下 `mysql` 命令登录数据库集群的数据库实例端点。

```
mysql -h DB_instance_endpoint -P 3306 -u user_name -p
```

> **Note**
>
> 如果在数据库实例上轮换 keytab，则身份验证可能会失败。在这种情况下，通过重新运行 `kinit` 获得新的 TGT。

直接连接

1. 在命令行界面上，使用以下 `mysql` 命令登录到数据库集群的数据库实例端点。

```
mysql -h DB_instance_endpoint -P 3306 -u user_name -p
```

2. 输入 Active Directory 用户的密码。

Windows

在 Windows 上，身份验证通常在登录时完成，因此，您无需手动获取 TGT 即可连接到 Aurora MySQL 数据库集群。数据库用户名的大小写必须与 Active Directory 中用户的字符大小写相匹配。例如，如果 Active Directory 中的用户显示为 `Admin`，则数据库用户名必须为 `Admin`。

在 Windows 上，MySQL 客户端版本 8.0.27 及更高版本支持客户端插件。

直接连接

- 在命令行界面上，使用以下 `mysql` 命令登录到数据库集群的数据库实例端点。

```
mysql -h DB_instance_endpoint -P 3306 -u user_name
```

Aurora 全局数据库的 Kerberos 身份验证

Aurora 全局数据库支持适用于 Aurora MySQL 的 Kerberos 身份验证。要使用主数据库集群的 Active Directory 对辅助数据库集群上的用户进行身份验证，请将 Active Directory 复制到辅助 AWS 区域。可以使用与主集群相同的域 ID 在辅助群集上开启 Kerberos 身份验证。只有企业版的 Active Directory 才支持 AWS Managed Microsoft AD 复制。有关更多信息，请参阅《AWS Directory Service 管理指南》中的多区域复制。

从 RDS for MySQL 迁移到 Aurora MySQL

从启用了 Kerberos 身份验证的 RDS for MySQL 迁移到 Aurora MySQL 后，修改使用 `auth_pam` 插件创建的用户以使用 `authentication_kerberos` 插件。例如：

```
ALTER USER user_name IDENTIFIED WITH 'authentication_kerberos' BY 'realm_name';
```

阻止票证缓存

如果 MySQL 客户端应用程序启动时不存在有效的 TGT，则应用程序可以获取并缓存 TGT。如果要阻止缓存 TGT，请在 `/etc/krb5.conf` 文件中设置配置参数。

> **Note**
>
> 此配置仅适用于运行 Unix 的客户端主机，而不适用于运行 Windows 的客户端主机。

防止 TGT 缓存

- 向 /etc/krb5.conf 中添加 [appdefaults] 部分，如下所示：

```
[appdefaults]
  mysql = {
    destroy_tickets = true
  }
```

针对 Kerberos 身份验证的日志记录

AUTHENTICATION_KERBEROS_CLIENT_LOG 环境变量设置 Kerberos 身份验证的日志记录级别。您可以使用日志进行客户端调试。

允许的值为 1–5。日志消息写入标准错误输出。下表描述了每个日志记录级别。

Logging level（日志记录级别）	描述
1 或未设置	不记录
2	错误消息
3	错误和警告消息
4	错误、警告和信息消息
5	错误、警告、信息和调试消息

在域中管理数据库集群

您可以使用 AWS CLI 或 RDS API 来管理您的数据库集群及其与托管式 Active Directory 的关系。例如，您可以关联一个 Active Directory 以进行 Kerberos 身份验证，并取消关联一个 Active Directory 以关闭 Kerberos 身份验证。也可以将由一个 Active Directory 在外部进行身份验证的数据库集群移到另一个 Active Directory。

例如，使用 Amazon RDS API，您可以执行下列操作：

- 要重新尝试为失败的成员资格开启 Kerberos 身份验证，请使用 ModifyDBInstance API 操作并指定当前成员的目录 ID。

- 要为成员资格更新 IAM 角色名称，请使用 ModifyDBInstance API 操作并指定当前成员资格的目录 ID 和新的 IAM 角色。
- 要在数据库集群上关闭 Kerberos 身份验证，请使用 ModifyDBInstance API 操作并指定 none 作为域参数。
- 要将数据库集群从一个域移至另一个域，请使用 ModifyDBInstance API 操作并指定新域的域标识符作为域参数。
- 要列出每个数据库集群的成员资格，请使用 DescribeDBInstances API 操作。

了解域成员资格

在创建或修改数据库集群后，此集群将成为域的成员。您可以通过运行 describe-db-clusters CLI 命令来查看数据库集群的域成员身份的状态。数据库集群的状态可以是以下状态之一：

- kerberos-enabled – 数据库集群已开启 Kerberos 身份验证。
- enabling-kerberos – AWS 正在此数据库集群上开启 Kerberos 身份验证。
- pending-enable-kerberos – 开启 Kerberos 身份验证的过程在此数据库集群上处于待处理状态。
- pending-maintenance-enable-kerberos – AWS 将尝试在下一个计划的维护时段在数据库集群上开启 Kerberos 身份验证。
- pending-disable-kerberos – 关闭 Kerberos 身份验证的过程在此数据库集群上处于待处理状态。
- pending-maintenance-disable-kerberos – AWS 将尝试在下一个计划的维护时段在数据库集群上关闭 Kerberos 身份验证。
- enable-kerberos-failed – 出现一个配置问题，导致 AWS 无法在数据库集群上开启 Kerberos 身份验证。在重新发出数据库集群修改命令之前检查并修复配置。
- disabling-kerberos – AWS 正在此数据库集群上关闭 Kerberos 身份验证。

开启 Kerberos 身份验证的请求可能因网络连接问题或 IAM 角色不正确而失败。例如，假设您创建数据库集群或修改现有数据库集群，但开启 Kerberos 身份验证的尝试失败。在这种情况下，请重新发出修改命令或修改新创建的数据库集群以加入域。

将数据迁移到 Amazon Aurora MySQL 数据库集群

对于将数据从现有数据库迁移到 Amazon Aurora MySQL 数据库集群，您有多种选择。您的迁移选项还取决于您从中迁移数据的数据库和您迁移数据的规模。

有两种不同类型的迁移：物理和逻辑。物理迁移意味着使用数据库文件的物理副本来迁移数据库。逻辑迁移意味着通过应用逻辑数据库更改 (如插入、更新和删除) 来完成迁移。

物理迁移有以下优势：

- 物理迁移比逻辑迁移要快，特别是对于大型数据库。
- 在进行物理迁移的备份时，数据库性能不会受到影响。
- 物理迁移可以迁移源数据库中的所有内容，包括复杂的数据库组件。

物理迁移具有以下限制：

- 必须将 innodb_page_size 参数设置为其默认值 (16KB)。
- innodb_data_file_path 参数必须仅配置一个使用默认数据文件名 "ibdata1:12M:autoextend" 的数据文件。使用此方法无法迁移具有两个数据文件或具有不同名称的数据文件的数据库。

  下面是不允许使用的文件名示例："innodb_data_file_path=ibdata1:50M; ibdata2:50M:autoextend" 和 "innodb_data_file_path=ibdata01:50M:autoextend"。
- 必须将 innodb_log_files_in_group 参数设置为其默认值 (2)。

逻辑迁移有以下优势：

- 您可以迁移数据库的子集，如特定表或表的若干部分。
- 无论物理存储结构如何，都可以迁移数据。

逻辑迁移具有以下限制：

- 逻辑迁移通常比物理迁移慢。
- 复杂的数据库组件可能会减慢逻辑迁移过程。在某些情况下，复杂的数据库组件甚至可以阻止逻辑迁移。

下表列出了您的选项以及每个选项的迁移类型。

迁移来源	迁移类型	解决方案
RDS for MySQL 数据库实例	物理	首先创建 MySQL 数据库实例的 Aurora MySQL 只读副本，然后可以从 RDS for MySQL 数据库实例进行迁移。如果 MySQL 数据库实例和 Aurora MySQL 只读副本之间的副本滞后为 0，您可以指示客户端应用程序从 Aurora 只读副本中读取数据，然后停止复制以使 Aurora MySQL 只读副本成为单独的 Aurora MySQL 数据库集群以进行读写。有关详细信息，请参阅使用 Aurora 只读副本将数据从 RDS for MySQL 数据库实例迁移到 Amazon Aurora MySQL 数据库集群。
RDS for MySQL 数据库快照	物理	您可以将数据直接从 RDS for MySQL 数据库快照迁移到 Amazon Aurora MySQL 数据库集群。有关详细信息，请参阅将 RDS for MySQL 快照迁移到 Aurora。
在 Amazon RDS 之外运行的 MySQL 数据库	逻辑	您可以使用 `mysqldump` 实用程序创建数据的转储，然后将该数据导入现有的 Amazon Aurora MySQL 数据库集群。有关详细信息，请参阅使用 mysqldump 从 MySQL 逻辑迁移到 Amazon Aurora MySQL。 要在从外部 MySQL 数据库迁移期间导出数据库用户的元数据，您还可以使用 MySQL Shell 命令而非 `mysqldump` 。有关更多信息，请参阅 Instance Dump Utility, Schema Dump Utility, and Table Dump Utility。

迁移来源	迁移类型	解决方案
		Note 从 MySQL 8.0.34 开始，mysqlpump 实用程序已弃用。
在 Amazon RDS 之外运行的 MySQL 数据库	物理	您可以将备份文件从数据库复制到 Amazon Simple Storage Service (Amazon S3) 存储桶，然后从这些文件还原 Amazon Aurora MySQL 数据库集群。该选项可能比使用 `mysqldump` 迁移数据要快得多。有关详细信息，请参阅使用 Percona XtraBackup 和 Amazon S3 从 MySQL 进行物理迁移。
在 Amazon RDS 之外运行的 MySQL 数据库	逻辑	您可将数据库中的数据另存为文本文件并将这些文件复制到 Amazon S3 存储桶。然后，您可以使用 `LOAD DATA FROM S3` MySQL 命令将数据加载到现有 Aurora MySQL 数据库集群中。有关更多信息，请参阅 将数据从 Amazon S3 存储桶中的文本文件加载到 Amazon Aurora MySQL 数据库集群。
不与 MySQL 兼容的数据库	逻辑	您可以使用 AWS Database Migration Service（AWS DMS）从不与 MySQL 兼容的数据库中迁移数据。有关 AWS DMS 的更多信息，请参阅什么是 AWS Database Migration Service？

Note

如果您正在迁移 Amazon RDS 外部的 MySQL 数据库，则仅当您的数据库支持 InnoDB 或 MyISAM 表空间时，表中描述的迁移选项才受支持。

如果要迁移到 Aurora MySQL 的 MySQL 数据库使用 memcached，请在迁移之前删除 memcached。

您无法从一些较早的 MySQL 8.0 版本（包括 8.0.11、8.0.13 和 8.0.15）迁移到 Aurora MySQL 版本 3.05 及更高版本。我们建议您在迁移之前先升级到 MySQL 版本 8.0.28。

将数据从外部 MySQL 数据库迁移到 Amazon Aurora MySQL 数据库集群

如果数据库支持 InnoDB 或 MyISAM 表空间，则可以使用以下选项将数据迁移到 Amazon Aurora MySQL 数据库集群：

- 您可以使用 `mysqldump` 实用程序创建数据的转储，然后将该数据导入现有的 Amazon Aurora MySQL 数据库集群。有关更多信息，请参阅 使用 mysqldump 从 MySQL 逻辑迁移到 Amazon Aurora MySQL。
- 您可以将完整备份文件和增量备份文件从数据库复制到 Amazon S3 桶，然后从这些文件还原到 Amazon Aurora MySQL 数据库集群。该选项可能比使用 `mysqldump` 迁移数据要快得多。有关更多信息，请参阅 使用 Percona XtraBackup 和 Amazon S3 从 MySQL 进行物理迁移。

主题

- 使用 Percona XtraBackup 和 Amazon S3 从 MySQL 进行物理迁移
- 使用 mysqldump 从 MySQL 逻辑迁移到 Amazon Aurora MySQL

使用 Percona XtraBackup 和 Amazon S3 从 MySQL 进行物理迁移

您可以将源 MySQL 版本 5.7 或 8.0 数据库中的完整和增量备份文件复制到 Amazon S3 桶中。然后，您可以从这些文件中还原到具有相同的主要数据库引擎版本的 Amazon Aurora MySQL 数据库集群。

该选项可能比使用 `mysqldump` 迁移数据要快得多，因为使用 `mysqldump` 可以重放所有命令，以便在新的 Aurora MySQL 数据库集群中从源数据库重新创建架构和数据。通过复制源 MySQL 数据文件，Aurora MySQL 可以立即将这些文件作为 Aurora MySQL 数据库集群的数据。

还可以在迁移过程中使用二进制日志复制来最大程度地减少停机时间。如果您使用二进制日志复制，则在将数据迁移到 Aurora MySQL 数据库集群时，外部 MySQL 数据库仍对事务保持开放。在创建 Aurora MySQL 数据库集群后，您可以使用二进制日志复制以将 Aurora MySQL 数据库集群与在备份后发生的事务同步。如果 Aurora MySQL 数据库集群与 MySQL 数据库一致，您可以完全切换到 Aurora MySQL 数据库集群以执行新事务，从而完成迁移。有关更多信息，请参阅 通过复制同步 Amazon Aurora MySQL 数据库集群和 MySQL 数据库。

目录

- 限制和注意事项
- 开始前的准备工作
  - 安装 Percona XtraBackup

  - 所需的权限
  - 创建 IAM 服务角色
- 备份要还原为 Amazon Aurora MySQL 数据库集群的文件
  - 使用 Percona XtraBackup 创建完整备份
  - 通过 Percona XtraBackup 使用增量备份
  - 备份注意事项
- 从 Amazon S3 存储桶还原 Amazon Aurora MySQL 数据库集群
- 通过复制同步 Amazon Aurora MySQL 数据库集群和 MySQL 数据库
  - 配置外部 MySQL 数据库和 Aurora MySQL 数据库集群以进行加密复制
  - 同步 Amazon Aurora MySQL 数据库集群和外部 MySQL 数据库
- 缩短物理迁移到 Amazon Aurora MySQL 的时间
  - 不支持的表类型
  - 具有不支持的权限的用户账户
  - Aurora MySQL 版本 3 中的动态权限
  - 以 'rdsadmin'@'localhost' 作为定义程序的存储对象

限制和注意事项

以下限制和注意事项适用于从 Amazon S3 桶还原到 Amazon Aurora MySQL 数据库集群：

- 您只能将数据迁移到新的数据库集群，而不能迁移到现有的数据库集群。
- 您必须使用 Percona XtraBackup 将数据备份到 S3。有关更多信息，请参阅 安装 Percona XtraBackup。
- Amazon S3 桶和 Aurora MySQL 数据库集群必须位于同一 AWS 区域中。
- 您无法从以下内容还原：
  - 导出到 Amazon S3 的数据库集群快照。您也无法将数据从数据库集群快照导出迁移到 S3 桶。
  - 加密的源数据库，但您可以加密正在迁移的数据。您也可以在迁移过程中不加密数据。
  - MySQL 5.5 或 5.6 数据库
- 不支持将 Percona Server for MySQL 作为源数据库，因为它可能在 `mysql` 架构中包含 `compression_dictionary*` 表。
- 您无法还原到 Aurora Serverless 数据库集群。

- 主要版本或次要版本都不支持反向迁移。例如，您不能从 MySQL 版本 8.0 迁移到 Aurora MySQL 2 版本（与 MySQL 5.7 兼容），也不能从 MySQL 版本 8.0.32 迁移到与 MySQL 社区版本 8.0.26 兼容的 Aurora MySQL 版本 3.03。
- 您无法从一些较早的 MySQL 8.0 版本（包括 8.0.11、8.0.13 和 8.0.15）迁移到 Aurora MySQL 版本 3.05 及更高版本。我们建议您在迁移之前先升级到 MySQL 版本 8.0.28。
- 在 db.t2.micro 数据库实例类上不支持从 Amazon S3 导入。不过，您可以还原到不同的数据库实例类，并稍后更改该数据库实例类。有关数据库实例类的更多信息，请参阅 Amazon Aurora 数据库实例类。
- Amazon S3 将上传到 S3 桶的文件大小限制为 5TB。如果备份文件超过 5 TB，则必须将备份文件拆分为较小的文件。
- Amazon RDS 将上传到 S3 桶的文件数限制为一百万个。如果数据库的备份数据（包括所有完整和增量备份）超过 100 万个文件，请使用 Gzip（.gz）、tar（.tar.gz）或 Percona xbstream（.xbstream）文件将完整和增量备份文件存储在 S3 桶中。Percona XtraBackup 8.0 仅支持使用 Percona xbstream 进行压缩。
- 要为每个数据库集群提供管理服务，需要在创建数据库集群时创建 `rdsadmin` 用户。由于这是 RDS 中的保留用户，因此以下限制适用：
  - 不会导入具有 `'rdsadmin'@'localhost'` 定义程序的函数、过程、视图、事件和触发器。有关更多信息，请参阅以 'rdsadmin'@'localhost' 作为定义程序的存储对象和Amazon Aurora MySQL 中的主用户权限。
  - 创建 Aurora MySQL 数据库集群时，将创建一个具有支持的最大权限的主用户。从备份还原时，分配给正在导入的用户的任何不受支持的权限都将在导入过程中自动删除。

    要识别可能受此影响的用户，请参阅具有不支持的权限的用户账户。有关 Aurora MySQL 中支持的权限的更多信息，请参阅基于角色的权限模型。
- 对于 Aurora MySQL 版本 3，不导入动态权限。迁移后可以导入 Aurora 支持的动态权限。有关更多信息，请参阅 Aurora MySQL 版本 3 中的动态权限。
- 不会迁移 `mysql` 架构中用户创建的表。
- `innodb_data_file_path` 参数必须仅配置一个使用默认数据文件名 `ibdata1:12M:autoextend` 的数据文件。使用此方法无法迁移具有两个数据文件或具有不同名称的数据文件的数据库。

  下面是不允许使用的文件名示例：`innodb_data_file_path=ibdata1:50M、ibdata2:50M:autoextend` 和 `innodb_data_file_path=ibdata01:50M:autoextend`。
- 您无法从在默认 MySQL 数据目录外部定义表的源数据库中迁移。

- 使用此方法的未压缩备份支持的最大大小目前限制为 64TiB。对于压缩备份，考虑到解压缩空间的需求，此限制会降低。在此类情况下，支持的最大备份大小为（`64 TiB – compressed backup size`）。
- Aurora MySQL 不支持导入 MySQL 以及其他外部组件和插件。
- Aurora MySQL 不会还原数据库中的所有内容。我们建议您从源 MySQL 数据库保存数据库架构以及以下项目的值，然后将这些内容添加到已还原的 Aurora MySQL 数据库集群（在创建该集群后）：
  - 用户账户
  - 函数
  - 存储过程
  - 时区信息。从 Aurora MySQL 数据库集群的本地操作系统中加载时区信息。有关更多信息，请参阅 Amazon Aurora 数据库集群的本地时区。

开始前的准备工作

在将数据复制到 Amazon S3 存储桶并从这些文件还原到数据库集群之前，您必须执行以下操作：

- 在您的本地服务器上安装 Percona XtraBackup。
- 准许 Aurora MySQL 代表您访问您的 Amazon S3 存储桶。

安装 Percona XtraBackup

Amazon Aurora 可以从使用 Percona XtraBackup 创建的文件还原数据库集群。您可以从软件下载 - Percona 安装 Percona XtraBackup。

对于 MySQL 5.7 迁移，使用 Percona XtraBackup 2.4。

对于 MySQL 8.0 迁移，使用 Percona XtraBackup 8.0。请确保 Percona XtraBackup 版本与您的源数据库的引擎版本兼容。

所需的权限

要将 MySQL 数据迁移到 Amazon Aurora MySQL 数据库集群，需要几种权限：

- 请求该 Aurora 从 Amazon S3 存储桶创建新集群的用户必须有权限列出您的AWS账户的存储桶。通过使用 AWS Identity and Access Management（IAM）策略，可以向用户授予该权限。
- Aurora 需要代表您访问 Amazon S3 存储桶（在该存储桶中，您存储了用于创建 Amazon Aurora MySQL 数据库集群的文件）的权限。您可以使用 IAM 服务角色为 Aurora 授予所需的权限。

- 发出请求的用户还必须有权限列出您的AWS账户的 IAM 角色。
- 如果用户发出请求的目的是创建 IAM 服务角色或请求 Aurora 创建 IAM 服务角色 (使用控制台)，则用户必须有权限为您的 AWS 账户创建 IAM 角色。
- 如果您打算在迁移过程中加密数据，请更新将要执行迁移操作的用户的 IAM 策略，以授予 RDS 对用于加密备份的 AWS KMS keys 的访问权限。有关说明，请参阅 创建 IAM 策略以访问 AWS KMS 资源。

例如，以下 IAM 策略为用户授予所需的最小权限，以使用控制台列出 IAM 角色、创建 IAM 角色、列出您的账户的 Amazon S3 存储桶以及列出 KMS 密钥。

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:ListRoles",
                "iam:CreateRole",
                "iam:CreatePolicy",
                "iam:AttachRolePolicy",
                "s3:ListBucket",
                "kms:ListKeys"
            ],
            "Resource": "*"
        }
    ]
}
```

此外，要使用户能够将 IAM 角色与 Amazon S3 存储桶关联，IAM 用户必须具有该 IAM 角色的 `iam:PassRole` 权限。该权限允许管理员限制用户可以将哪些 IAM 角色与 Amazon S3 存储桶关联。

例如，以下 IAM 策略允许用户将名为 `S3Access` 的角色与 Amazon S3 存储桶关联。

JSON

```
{
```

```
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"AllowS3AccessRole",
            "Effect":"Allow",
            "Action":"iam:PassRole",
            "Resource":"arn:aws:iam::123456789012:role/S3Access"
        }
    ]
}
```

有关 IAM 用户权限的更多信息，请参阅使用策略管理访问。

创建 IAM 服务角色

您可以在 AWS Management Console中选择创建新角色选项（本主题稍后介绍）以创建角色。如果您选择该选项并为新角色指定名称，Aurora 将使用您提供的名称为 Aurora 创建访问 Amazon S3 存储桶所需的 IAM 服务角色。

作为替代方法，您可以使用以下程序手动创建角色。

为 Aurora 创建 IAM 角色以访问 Amazon S3

1. 完成 创建 IAM 策略以访问 Amazon S3 资源 中的步骤。
2. 完成 创建 IAM 角色以允许 Amazon Aurora 访问AWS服务 中的步骤。
3. 完成 将 IAM 角色与 Amazon Aurora MySQL 数据库集群关联 中的步骤。

备份要还原为 Amazon Aurora MySQL 数据库集群的文件

您可以使用 Percona XtraBackup 创建 MySQL 数据库文件的完整备份，并且可将备份文件上传到 Amazon S3 存储桶。或者，如果您已使用 Percona XtraBackup 备份您的 MySQL 数据库文件，则可将现有的完整备份和增量备份目录和文件上传到 Amazon S3 存储桶。

主题

- 使用 Percona XtraBackup 创建完整备份
- 通过 Percona XtraBackup 使用增量备份
- 备份注意事项

使用 Percona XtraBackup 创建完整备份

要创建 MySQL 数据库文件的完整备份，以从 Amazon S3 中还原这些文件来创建 Aurora MySQL 数据库集群，请使用 Percona XtraBackup 实用程序（`xtrabackup`）备份数据库。

例如，以下命令创建 MySQL 数据库的备份并将备份文件存储在 `/on-premises/s3-restore/backup` 文件夹中。

```
xtrabackup --backup --user=<myuser> --password=<password> --target-dir=</on-premises/s3-restore/backup>
```

如果您要将备份压缩为单个文件 (可在需要时进行拆分)，则可以使用 `--stream` 选项来采用下列格式之一保存备份：

- Gzip (.gz)
- tar (.tar)
- Percona xbstream (.xbstream)

以下命令为您的 MySQL 数据库创建一个拆分成多个 Gzip 文件的备份。

```
xtrabackup --backup --user=<myuser> --password=<password> --stream=tar \
   --target-dir=</on-premises/s3-restore/backup> | gzip - | split -d --bytes=500MB \
   - </on-premises/s3-restore/backup/backup>.tar.gz
```

以下命令为您的 MySQL 数据库创建一个拆分成多个 tar 文件的备份。

```
xtrabackup --backup --user=<myuser> --password=<password> --stream=tar \
   --target-dir=</on-premises/s3-restore/backup> | split -d --bytes=500MB \
   - </on-premises/s3-restore/backup/backup>.tar
```

以下命令为您的 MySQL 数据库创建一个拆分成多个 xbstream 文件的备份。

```
xtrabackup --backup --user=<myuser> --password=<password> --stream=xbstream \
   --target-dir=</on-premises/s3-restore/backup> | split -d --bytes=500MB \
   - </on-premises/s3-restore/backup/backup>.xbstream
```

> **Note**
>
> 如果您看到以下错误，则可能是由于命令中混用了文件格式所致：

```
ERROR:/bin/tar: This does not look like a tar archive
```

在使用 Percona XtraBackup 实用程序备份 MySQL 数据库后，您可以将备份目录和文件复制到 Amazon S3 存储桶。

有关创建文件并将其上传到 Amazon S3 存储桶的信息，请参阅 Amazon S3 入门指南 中的 Amazon Simple Storage Service 入门。

通过 Percona XtraBackup 使用增量备份

Amazon Aurora MySQL 支持使用 Percona XtraBackup 创建的完整备份和增量备份。如果您已使用 Percona XtraBackup 对 MySQL 数据库文件进行完整备份和增量备份，则无需创建完整备份和将备份文件上传到 Amazon S3。相反，您可以通过将网站备份和增量备份的现有备份目录和文件复制到 Amazon S3 存储桶来节约大量时间。有关更多信息，请参阅 Percona 网站上的创建增量备份。

在将现有的完整备份和增量备份文件复制到 Amazon S3 存储桶时，您必须以递归方式复制基目录的内容。这些内容包括完整备份以及所有增量备份目录和文件。此副本必须在 Amazon S3 存储桶中保留目录结构。Aurora 将循环访问所有文件和目录。Aurora 使用每个增量备份中包含的 `xtrabackup-checkpoints` 文件来标识基本目录，并按日志序列号（LSN）范围对增量备份进行排序。

有关创建文件并将其上传到 Amazon S3 存储桶的信息，请参阅 Amazon S3 入门指南 中的 Amazon Simple Storage Service 入门。

备份注意事项

Aurora 不支持使用 Percona XtraBackup 创建的部分备份。在备份数据库的源文件时，不能使用以下选项创建部分备份：`--tables`、`--tables-exclude`、`--tables-file`、`--databases`、`--databases-exclude` 或 `--databases-file`。

有关使用 Percona XtraBackup 备份数据库的更多信息，请参阅 Percona 网站上的 Percona XtraBackup - 文档和使用二进制日志。

Aurora 支持使用 Percona XtraBackup 创建的增量备份。有关更多信息，请参阅 Percona 网站上的创建增量备份。

Aurora 根据文件名使用您的备份文件。确保根据文件格式为备份文件指定相应的文件扩展名，例如对于使用 Percona xbstream 格式存储的文件，指定 `.xbstream`。

Aurora 按照字母顺序以及自然数字顺序使用您的备份文件。始终在您发出 `split` 命令时使用 `xtrabackup` 选项，以确保备份文件按适当的顺序写入和命名。

Amazon S3 将上传到 Amazon S3 存储桶的文件大小限制为 5 TB。如果数据库的备份数据超过 5 TB，请使用 `split` 命令将备份文件拆分为多个文件，每个小于 5 TB。

Aurora 将上传到 Amazon S3 存储桶的源文件数限制为 100 万个。在某些情况下，数据库的备份数据 (包括所有完整和增量备份) 可以提供大量文件。在这些情况下，使用 tarball (.tar.gz) 文件，将完整和增量备份文件存储在 Amazon S3 存储桶中。

将文件上传到 Amazon S3 存储桶时，您可以使用服务器端加密来加密数据。之后，您可以通过这些加密文件还原 Amazon Aurora MySQL 数据库集群。Amazon Aurora MySQL 可以从使用以下类型的服务器端加密功能加密的文件还原数据库集群：

- 使用具有 Amazon S3 托管密钥的服务器端加密 (SSE-S3) – 通过增强的多因素加密使用唯一密钥加密每个对象。
- 具有 AWS KMS 托管密钥的服务器端加密（SSE-KMS）与 SSE-S3 类似，但您可以选择自己创建和管理加密密钥，并且还具有其他一些区别。

有关在将文件上传到 Amazon S3 存储桶时使用服务器端加密的信息，请参阅 Amazon S3 开发人员指南 中的使用服务器端加密保护数据。

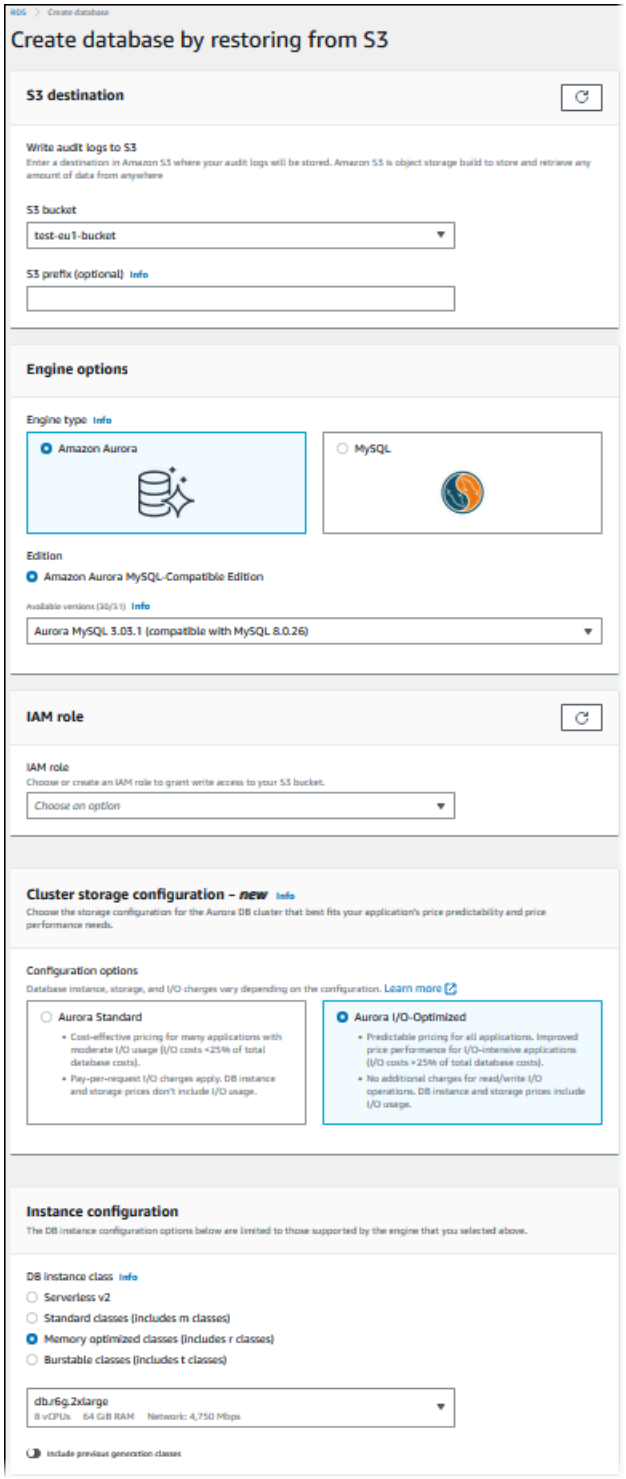
从 Amazon S3 存储桶还原 Amazon Aurora MySQL 数据库集群

您可以使用 Amazon RDS 控制台从 Amazon S3 存储桶中还原备份文件以创建新的 Amazon Aurora MySQL 数据库集群。

从 Amazon S3 存储桶上的文件还原 Amazon Aurora MySQL 数据库集群

1. 登录AWS Management Console并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在 Amazon RDS 控制台右上角，选择要在其中创建数据库集群的AWS区域。选择与包含数据库备份的 Amazon S3 存储桶相同的AWS区域。
3. 在导航窗格中，选择 Databases (数据库)，然后选择 Restore from S3 (从 S3 还原)。
4. 选择从 S3 还原。

   此时将显示通过从 S3 还原创建数据库页面。



5. 在 S3 目标下：

   a. 选择包含备份文件的 S3 存储桶。

   b. （可选）对于 S3 文件夹路径前缀，输入存储在 Amazon S3 存储桶中的文件的文件路径前缀。

如果未指定前缀，则 RDS 使用 S3 存储桶的根文件夹中的所有文件和文件夹创建数据库实例。如果指定了前缀，则 RDS 使用 S3 存储桶中文件路径以指定前缀开头的文件和文件夹创建数据库实例。

例如，假设将备份文件存储在 S3 上名为 backups 的子文件夹中，并且具有多组备份文件，每个文件位于自己的目录 (gzip_backup1、gzip_backup2，依此类推) 中。在这种情况下，请指定 backups/gzip_backup1 前缀以从 gzip_backup1 文件夹中的文件还原。

6. 在引擎选项下：

   a. 对于 Engine type（引擎类型），选择 Amazon Aurora。

   b. 对于 Version（版本），为还原的数据库实例选择 Aurora MySQL 引擎版本。

7. 对于 IAM 角色，您可以选择现有 IAM 角色。

8. （可选）您还可以通过选择 Create a new role（创建新角色）为自己创建新的 IAM 角色。如果是这样：

   a. 请输入 IAM role name（IAM 角色名称）。

   b. 选择是否允许访问 KMS 密钥：

      - 如果您不加密备份文件，请选择否。
      - 如果在将备份文件上传到 Amazon S3 时使用 AES-256（SSE-S3）加密这些文件，请选择 No (否)。在此情况下，数据将会自动解密。
      - 如果在将备份文件上载到 Amazon S3 时使用 AWS KMS (SSE-KMS) 服务器端加密功能加密这些文件，请选择 Yes (是)。接下来，为 AWS KMS key 选择正确的 KMS 密钥。

        AWS Management Console 将创建一个 IAM 策略以允许 Aurora 解密数据。

   有关更多信息，请参阅 Amazon S3 开发人员指南 中的使用服务器端加密保护数据。

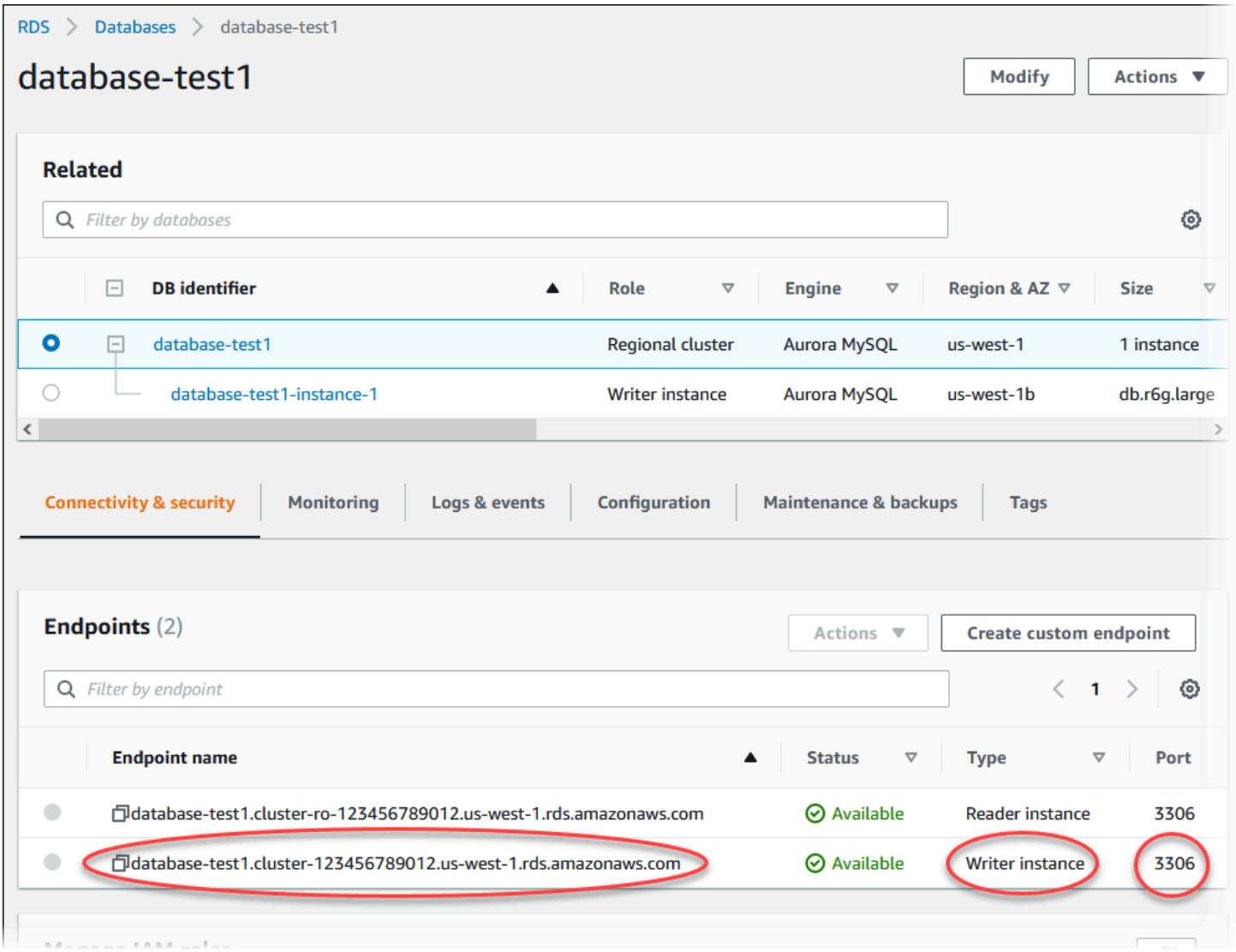
9. 选择数据库集群的设置，例如数据库集群存储配置、数据库实例类、数据库集群标识符和登录凭证。有关每项设置的信息，请参阅 Aurora 数据库集群的设置。

10. 根据需要自定义 Aurora MySQL 数据库集群的其他设置。

11. 选择 Create database（创建数据库）以启动您的 Aurora 数据库实例。

在 Amazon RDS 控制台中，新数据库实例显示在数据库实例列表中。数据库实例具有 creating (创建) 状态，直到该数据库实例完成创建并可供使用。当状态更改为 available 时，您可连接到数据库集群的主实例。根据所分配的数据库实例类和存储的不同，新实例可能需要数分钟时间才能变得可用。

要查看新创建的集群，请在 Amazon RDS 控制台中选择 Databases(数据库) 视图，然后选择数据库集群。有关更多信息，请参阅“查看 Amazon Aurora 数据库集群”。



记下数据库集群的端口和写入器终端节点。在执行写入或读取操作的任何应用程序的 JDBC 和 ODBC 连接字符串中，使用数据库集群的写入器终端节点和端口。

通过复制同步 Amazon Aurora MySQL 数据库集群和 MySQL 数据库

为了在迁移过程中实现最少或零停机时间，您可以将在 MySQL 数据库中提交的事务复制到 Aurora MySQL 数据库集群。通过复制，数据库集群可以与迁移期间发生的 MySQL 数据库上的事务保持一致。当数据库集群完全一致时，您可以停止复制并完成向 Aurora MySQL 的迁移。

主题

- 配置外部 MySQL 数据库和 Aurora MySQL 数据库集群以进行加密复制

- 同步 Amazon Aurora MySQL 数据库集群和外部 MySQL 数据库

配置外部 MySQL 数据库和 Aurora MySQL 数据库集群以进行加密复制

要安全地复制数据，您可以使用加密复制。

> **Note**
>
> 如果您不需要使用加密复制，可以跳过以下步骤，然后继续参阅 同步 Amazon Aurora MySQL 数据库集群和外部 MySQL 数据库 中的说明。

以下是使用加密复制的先决条件：

- 必须在外部 MySQL 主数据库上启用安全套接字层 (SSL)。
- 必须为 Aurora MySQL 数据库集群准备客户端密钥和客户端证书。

在加密复制期间，Aurora MySQL 数据库集群充当 MySQL 数据库服务器的客户端。Aurora MySQL 客户端的证书和密钥必须是 .pem 格式的文件。

配置外部 MySQL 数据库和 Aurora MySQL 数据库集群以进行加密复制

1. 确保您为加密复制做好以下准备：

   - 如果您没有在外部 MySQL 主数据库上启用 SSL，并且没有准备客户端密钥和客户端证书，请在 MySQL 数据库服务器上启用 SSL 并生成所需的客户端密钥和客户端证书。
   - 如果在外部主数据库上启用了 SSL，请提供 Aurora MySQL 数据库集群的客户端密钥和证书。如果未提供，请为 Aurora MySQL 数据库集群生成新密钥和证书。要对客户端证书进行签名，您必须拥有您用来在外部 MySQL 主数据库上配置 SSL 的证书颁发机构密钥。

   有关更多信息，请参阅 MySQL 文档中的使用 openssl 创建 SSL 证书和密钥。

   您需要证书颁发机构证书、客户端密钥和客户端证书。
2. 通过 SSL 以主用户身份连接到 Aurora MySQL 数据库集群。

   有关通过 SSL 连接到 Aurora MySQL 数据库集群的信息，请参阅与 Aurora MySQL 数据库集群的 TLS 连接。

3. 运行 mysql.rds_import_binlog_ssl_material 存储过程，以将 SSL 信息导入到 Aurora MySQL 数据库集群中。

   对于 `ssl_material_value` 参数，将 Aurora MySQL 数据库集群的 .pem 格式文件中的信息插入到正确的 JSON 负载中。

   以下示例将 SSL 信息导入到 Aurora MySQL 数据库集群中。在 .pem 格式文件中，主体代码通常比示例中显示的主体代码长。

```
call mysql.rds_import_binlog_ssl_material(
'{"ssl_ca":"-----BEGIN CERTIFICATE-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS7O6V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSFJuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz221CBt5IMucxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END CERTIFICATE-----\n","ssl_cert":"-----BEGIN CERTIFICATE-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS7O6V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSFJuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz221CBt5IMucxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END CERTIFICATE-----\n","ssl_key":"-----BEGIN RSA PRIVATE KEY-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS7O6V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSFJuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz221CBt5IMucxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END RSA PRIVATE KEY-----\n"}');
```

   有关更多信息，请参阅mysql.rds_import_binlog_ssl_material和与 Aurora MySQL 数据库集群的 TLS 连接。

   > **Note**
   >
   > 运行该过程后，密钥会存储在文件中。稍后如果要删除文件，您可以运行 mysql.rds_remove_binlog_ssl_material 存储过程。

同步 Amazon Aurora MySQL 数据库集群和外部 MySQL 数据库

您可以使用复制，同步 Amazon Aurora MySQL 数据库集群和 MySQL 数据库。

通过复制同步 Aurora MySQL 数据库集群和 MySQL 数据库

1. 确保外部 MySQL 数据库的 /etc/my.cnf 文件具有相关条目。

   如果不需要加密复制，请确保先为外部 MySQL 数据库启用二进制日志 (binlogs) 并禁用 SSL。以下是 /etc/my.cnf 文件中未加密数据的相关条目。

```
log-bin=mysql-bin
server-id=2133421
innodb_flush_log_at_trx_commit=1
sync_binlog=1
```

   如果需要加密复制，请确保先为外部 MySQL 数据库启用 SSL 和二进制日志。/etc/my.cnf 文件的条目包括 MySQL 数据库服务器的 .pem 文件位置。

```
log-bin=mysql-bin
server-id=2133421
innodb_flush_log_at_trx_commit=1
sync_binlog=1

# Setup SSL.
ssl-ca=/home/sslcerts/ca.pem
ssl-cert=/home/sslcerts/server-cert.pem
ssl-key=/home/sslcerts/server-key.pem
```

   您可以通过以下命令验证是否已启用 SSL。

```
mysql> show variables like 'have_ssl';
```

   您的输出应类似于以下内容。

```
+~-~-~-
```

```
+~-~-~-
```

```
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO
 '<user_name>'@'<domain_name>';
```

如果您需要使用加密复制，则需为复制用户要求 SSL 连接。例如，您可以使用以下语句来要求用户账户的 SSL 连接 *<user_name>*。

```
GRANT USAGE ON *.* TO '<user_name>'@'<domain_name>' REQUIRE SSL;
```

> **Note**
>
> 如果不包括 REQUIRE SSL，则复制连接可能会无提示地返回到未加密连接。

4. 在 Amazon RDS 控制台中，将托管外部 MySQL 数据库的服务器的 IP 地址添加到 Aurora MySQL 数据库集群的 VPC 安全组中。有关修改 VPC 安全组的更多信息，请参阅 Amazon Virtual Private Cloud 用户指南 中的您的 VPC 的安全组。

   您可能还需要配置本地网络以允许来自 Aurora MySQL 数据库集群的 IP 地址的连接，以便它能与外部 MySQL 数据库进行通信。要查找 Aurora MySQL 数据库集群的 IP 地址，请使用 `host` 命令。

   ```
   host <db_cluster_endpoint>
   ```

   主机名是 Aurora MySQL 数据库集群终端节点中的 DNS 名称。

5. 通过运行 mysql.rds_reset_external_master（Aurora MySQL 版本 2）或 mysql.rds_reset_external_source（Aurora MySQL 版本 3）存储过程，启用二进制日志复制。此存储过程使用以下语法。

   ```
   CALL mysql.rds_set_external_master (
     host_name
     , host_port
     , replication_user_name
     , replication_user_password
     , mysql_binary_log_file_name
     , mysql_binary_log_file_location
     , ssl_encryption
   ```

```
);

CALL mysql.rds_set_external_source (
  host_name
  , host_port
  , replication_user_name
  , replication_user_password
  , mysql_binary_log_file_name
  , mysql_binary_log_file_location
  , ssl_encryption
);
```

有关参数的信息，请参阅mysql.rds_reset_external_master（Aurora MySQL 版本 2）和mysql.rds_reset_external_source（Aurora MySQL 版本 3）。

对于 mysql_binary_log_file_name 和 mysql_binary_log_file_location，使用您之前记下的从二进制日志文件名还原事件的位置。

如果未加密 Aurora MySQL 数据库集群中的数据，则必须将 ssl_encryption 参数设置为 0。如果数据经过加密，则必须将 ssl_encryption 参数设置为 1。

以下示例为含有加密数据的 Aurora MySQL 数据库集群运行该过程。

```
CALL mysql.rds_set_external_master(
  'Externaldb.some.com',
  3306,
  'repl_user'@'mydomain.com',
  'password',
  'mysql-bin.000010',
  120,
  1);

CALL mysql.rds_set_external_source(
  'Externaldb.some.com',
  3306,
  'repl_user'@'mydomain.com',
  'password',
  'mysql-bin.000010',
  120,
  1);
```

此存储过程设置了参数，Aurora MySQL 数据库集群使用此类参数连接到外部 MySQL 数据库并读取其二进制日志。如果数据经过加密，则它还会将 SSL 证书颁发机构证书、客户端证书和客户端密钥下载到本地磁盘。

6. 通过运行 mysql.rds_start_replication 存储过程，启动二进制日志复制。

```
CALL mysql.rds_start_replication;
```

7. 监控 Aurora MySQL 数据库集群落后于 MySQL 副本主数据库的程度。为了做到这一点，请连接到 Aurora MySQL 数据库集群并运行以下命令。

```
Aurora MySQL version 2:
SHOW SLAVE STATUS;

Aurora MySQL version 3:
SHOW REPLICA STATUS;
```

在命令输出中，`Seconds Behind Master` 字段显示 Aurora MySQL 数据库集群落后于 MySQL 主实例的程度。如果该值为 `0`（零），则表示 Aurora MySQL 数据库集群与主实例保持一致，您可以转到下一步以停止复制。

8. 连接到 MySQL 副本主数据库并停止复制。为实现此目的，请运行 mysql.rds_stop_replication 存储过程。

```
CALL mysql.rds_stop_replication;
```

缩短物理迁移到 Amazon Aurora MySQL 的时间

您可以进行以下数据库修改，以加快将数据库迁移到 Amazon Aurora MySQL 的过程。

> **Important**
>
> 请确保在生产数据库的副本而不是在生产数据库上执行这些更新。然后，您可以备份副本并将其还原到 Aurora MySQL 数据库集群，以避免生产数据库上产生任何服务中断。

不支持的表类型

Aurora MySQL 仅支持用于数据库表的 InnoDB 引擎。如果您的数据库中有 MyISAM 表，则必须先转换这些表，然后才能迁移到 Aurora MySQL 中。迁移过程中，转换过程需要额外的空间以便将 MyISAM 转换为 InnoDB。

若要降低空间用尽的可能性或加快迁移过程，请先将所有 MyISAM 表转换为 InnoDB 表，然后再迁移这些表。生成的 InnoDB 表的大小与 Aurora MySQL 要求该表具有的大小相同。要将 MyISAM 表转换为 InnoDB 表，请运行以下命令：

```
ALTER TABLE schema.table_name engine=innodb, algorithm=copy;
```

Aurora MySQL 不支持压缩的表或页（即，使用 ROW_FORMAT=COMPRESSED 或 COMPRESSION = {"zlib"|"lz4"} 创建的表）。

为了避免用完空间或为了加速迁移过程，请通过将 ROW_FORMAT 设置为 DEFAULT、COMPACT、DYNAMIC 或 REDUNDANT 来扩展您的压缩表。对于压缩的页，请设置 COMPRESSION="none"。

有关更多信息，请参阅 MySQL 文档中的 InnoDB 行格式和 InnoDB 表和页压缩。

您可以在现有 MySQL 数据库实例上使用以下 SQL 脚本来列出数据库中属于 MyISAM 表或压缩表的表。

```
-- This script examines a MySQL database for conditions that block
-- migrating the database into Aurora MySQL.
-- It must be run from an account that has read permission for the
-- INFORMATION_SCHEMA database.

-- Verify that this is a supported version of MySQL.

select msg as `==> Checking current version of MySQL.`
from
  (
  select
    'This script should be run on MySQL version 5.6 or higher. ' +
    'Earlier versions are not supported.' as msg,
    cast(substring_index(version(), '.', 1) as unsigned) * 100 +
      cast(substring_index(substring_index(version(), '.', 2), '.', -1)
      as unsigned)
    as major_minor
  ) as T
```

```
where major_minor <> 506;


-- List MyISAM and compressed tables. Include the table size.

select concat(TABLE_SCHEMA, '.', TABLE_NAME) as `==> MyISAM or Compressed Tables`,
round(((data_length + index_length) / 1024 / 1024), 2) "Approx size (MB)"
from INFORMATION_SCHEMA.TABLES
where
  ENGINE <> 'InnoDB'
  and
  (
    -- User tables
    TABLE_SCHEMA not in ('mysql', 'performance_schema',
                         'information_schema')
    or
    -- Non-standard system tables
    (
      TABLE_SCHEMA = 'mysql' and TABLE_NAME not in
        (
          'columns_priv', 'db', 'event', 'func', 'general_log',
          'help_category', 'help_keyword', 'help_relation',
          'help_topic', 'host', 'ndb_binlog_index', 'plugin',
          'proc', 'procs_priv', 'proxies_priv', 'servers', 'slow_log',
          'tables_priv', 'time_zone', 'time_zone_leap_second',
          'time_zone_name', 'time_zone_transition',
          'time_zone_transition_type', 'user'
        )
    )
  )
  or
  (
    -- Compressed tables
       ROW_FORMAT = 'Compressed'
  );
```

具有不支持的权限的用户账户

如果用户账户的权限不受 Aurora MySQL 支持，则在导入这些账户时将不包含这些不受支持的权限。有关支持的权限列表，请参阅基于角色的权限模型。

可以在源数据库上运行以下 SQL 查询，以列出具有不受支持的权限的用户账户。

```
SELECT
```

```
    user,
    host
FROM
    mysql.user
WHERE
    Shutdown_priv = 'y'
    OR File_priv = 'y'
    OR Super_priv = 'y'
    OR Create_tablespace_priv = 'y';
```

Aurora MySQL 版本 3 中的动态权限

不会导入动态权限。Aurora MySQL 版本 3 支持以下动态权限。

```
'APPLICATION_PASSWORD_ADMIN',
'CONNECTION_ADMIN',
'REPLICATION_APPLIER',
'ROLE_ADMIN',
'SESSION_VARIABLES_ADMIN',
'SET_USER_ID',
'XA_RECOVER_ADMIN'
```

以下示例脚本向 Aurora MySQL 数据库集群中的用户账户授予支持的动态权限。

```
-- This script finds the user accounts that have Aurora MySQL supported dynamic
 privileges
-- and grants them to corresponding user accounts in the Aurora MySQL DB cluster.

/home/ec2-user/opt/mysql/8.0.26/bin/mysql -uusername -pxxxxx -P8026 -h127.0.0.1 -BNe
 "SELECT
  CONCAT('GRANT ', GRANTS, ' ON *.* TO ', GRANTEE ,';') AS grant_statement
  FROM (select GRANTEE, group_concat(privilege_type) AS GRANTS FROM
 information_schema.user_privileges
      WHERE privilege_type IN (
        'APPLICATION_PASSWORD_ADMIN',
        'CONNECTION_ADMIN',
        'REPLICATION_APPLIER',
        'ROLE_ADMIN',
        'SESSION_VARIABLES_ADMIN',
        'SET_USER_ID',
        'XA_RECOVER_ADMIN')
      AND GRANTEE NOT IN (\"'mysql.session'@'localhost'\",
\"'mysql.infoschema'@'localhost'\",\"'mysql.sys'@'localhost'\") GROUP BY GRANTEE)
```

```
      AS PRIVGRANTS; " | /home/ec2-user/opt/mysql/8.0.26/bin/mysql -u master_username -
p master_password -h DB_cluster_endpoint
```

以 'rdsadmin'@'localhost' 作为定义程序的存储对象

不会导入以 'rdsadmin'@'localhost' 作为定义程序的函数、过程、视图、事件和触发器。

您可以在源 MySQL 数据库上使用以下 SQL 脚本来列出具有不受支持的定义程序的存储对象。

```
-- This SQL query lists routines with `rdsadmin`@`localhost` as the definer.

SELECT
    ROUTINE_SCHEMA,
    ROUTINE_NAME
FROM
    information_schema.routines
WHERE
    definer = 'rdsadmin@localhost';

-- This SQL query lists triggers with `rdsadmin`@`localhost` as the definer.

SELECT
    TRIGGER_SCHEMA,
    TRIGGER_NAME,
    DEFINER
FROM
    information_schema.triggers
WHERE
    DEFINER = 'rdsadmin@localhost';

-- This SQL query lists events with `rdsadmin`@`localhost` as the definer.

SELECT
    EVENT_SCHEMA,
    EVENT_NAME
FROM
    information_schema.events
WHERE
    DEFINER = 'rdsadmin@localhost';

-- This SQL query lists views with `rdsadmin`@`localhost` as the definer.
SELECT
    TABLE_SCHEMA,
    TABLE_NAME
```

```
FROM
    information_schema.views
WHERE
    DEFINER = 'rdsadmin@localhost';
```

使用 mysqldump 从 MySQL 逻辑迁移到 Amazon Aurora MySQL

因为 Amazon Aurora MySQL 是与 MySQL 兼容的数据库，所以您可以使用 `mysqldump` 实用程序从 MySQL 或 MariaDB 数据库中将数据复制到现有 Aurora MySQL 数据库集群。

有关如何为大型 MySQL 数据库执行该操作的讨论，请参阅将数据导入到 MySQL 或 MariaDB 数据库实例中，同时减少停机时间。对于具有较少量数据的 MySQL 数据库，请参阅将数据从 MySQL 或 MariaDB 数据库导入到 MySQL 或 MariaDB 数据库实例。

将数据从 RDS for MySQL DB 数据库实例迁移到 Amazon Aurora MySQL 数据库集群

您可以将数据直接从 RDS for MySQL 数据库实例迁移（复制）到 Amazon Aurora MySQL 数据库集群。

主题

- 将 RDS for MySQL 快照迁移到 Aurora
- 使用 Aurora 只读副本将数据从 RDS for MySQL 数据库实例迁移到 Amazon Aurora MySQL 数据库集群

> **Note**
>
> 由于 Amazon Aurora MySQL 与 MySQL 兼容，因此，您可以在 MySQL 数据库和 Amazon Aurora MySQL 数据库集群之间设置复制以迁移 MySQL 数据库中的数据。有关更多信息，请参阅 使用 Amazon Aurora 进行复制。

将 RDS for MySQL 快照迁移到 Aurora

您可以迁移 RDS for MySQL 数据库实例的数据库快照来创建 Aurora MySQL 数据库集群。将使用原始 RDS for MySQL 数据库实例中的数据填充新的 Aurora MySQL 数据库集群。必须已从运行与 Aurora MySQL 兼容的 MySQL 版本的 Amazon RDS 数据库实例中创建数据库快照。

手动数据库快照和自动数据库快照都可以迁移。创建数据库集群后，您可创建可选 Aurora 副本。

> **Note**
>
> 您还可以通过创建源 RDS for MySQL 数据库实例的 Aurora 只读副本，将 RDS for MySQL 数据库实例迁移到 Aurora MySQL 数据库集群。有关更多信息，请参阅 使用 Aurora 只读副本将数据从 RDS for MySQL 数据库实例迁移到 Amazon Aurora MySQL 数据库集群。
> 您无法从一些较早的 MySQL 8.0 版本（包括 8.0.11、8.0.13 和 8.0.15）迁移到 Aurora MySQL 版本 3.05 及更高版本。我们建议您在迁移之前先升级到 MySQL 版本 8.0.28。

您必须采取的常见步骤如下：

1. 确定要为 Aurora MySQL 数据库集群配置的空间量。有关更多信息，请参阅“需要多少空间？”
2. 使用控制台在 Amazon RDS MySQL 实例所在的AWS区域中创建快照。有关创建数据库快照的信息，请参阅创建数据库快照。
3. 如果数据库快照不在数据库集群所在的AWS区域中，请使用 Amazon RDS 控制台将数据库快照复制到该AWS区域。有关复制数据库快照的信息，请参阅复制数据库快照。
4. 使用控制台迁移数据库快照，并创建包含的数据库与原始 MySQL 数据库实例相同的 Aurora MySQL 数据库集群。

> **Warning**
>
> Amazon RDS 只允许每个AWS账户一次对应于每个AWS区域中的一个快照副本。

需要多少空间？

将 MySQL 数据库实例的快照迁移到 Aurora MySQL 数据库集群时，Aurora 将先使用 Amazon Elastic Block Store (Amazon EBS) 卷设置快照中数据的格式，然后再迁移数据。在某些情况下，为迁移设置数据格式操作需要额外的空间。

MyISAM 之外的未压缩的表的大小可高达 16 TB。如果具有 MyISAM 表，则 Aurora 必须使用卷中的额外空间转换这些表以使其与 Aurora MySQL 兼容。如果有压缩表，则 Aurora 必须使用卷中的额外空间将这些表解压缩，然后再将它们存储在 Aurora 集群卷中。由于此额外的空间要求，您应确保从 MySQL 数据库实例迁移的 MyISAM 表和压缩表的大小不会超出 8 TB。

减少将数据迁移到 Amazon Aurora MySQL 中所需的空间量

您可能需要先修改数据库架构，然后再将其迁移到 Amazon Aurora 中。这种修改在以下情况下会很有用：

- 您需要加快迁移过程。
- 您不确定需要配置的空间量。
- 您已尝试迁移数据，但迁移因配置的空间不足而失败。

您可进行以下更改来改善将数据库迁移到 Amazon Aurora 的过程。

> **Important**
>
> 请务必对从生产数据库快照还原的新数据库实例而非生产实例执行这些更新。然后，您可以将新数据库实例快照中的数据迁移到 Aurora 数据库集群，以避免生产数据库发生任何服务中断。

表类型	限制或指南
MyISAM 表	Aurora MySQL 仅支持 InnoDB 表。如果您的数据库中有 MyISAM 表，则必须先转换这些表，然后才能将其迁移到 Aurora MySQL 中。迁移过程中，转换过程需要额外的空间以便将 MyISAM 转换为 InnoDB。 若要降低空间用尽的可能性或加快迁移过程，请先将所有 MyISAM 表转换为 InnoDB 表，然后再迁移这些表。生成的 InnoDB 表的大小与 Aurora MySQL 要求该表具有的大小相同。要将 MyISAM 表转换为 InnoDB 表，请运行以下命令： `alter table <schema>.<table_name> engine=innodb, algorithm=copy;`
压缩表	Aurora MySQL 不支持压缩的表（即，使用 `ROW_FORMAT=COMPRESSED` 创建的表）。 为了避免用完空间或为了加速迁移过程，请通过将 `ROW_FORMAT` 设置为 `DEFAULT`、`COMPACT`、`DYNAMIC` 或 `REDUNDANT` 来扩展您的压缩表。有关更多信息，请参阅 MySQL 文档中的 InnoDB 行格式。

您可以在现有 MySQL 数据库实例上使用以下 SQL 脚本来列出数据库中属于 MyISAM 表或压缩表的表。

```
-- This script examines a MySQL database for conditions that block
-- migrating the database into Amazon Aurora.
-- It needs to be run from an account that has read permission for the
-- INFORMATION_SCHEMA database.
```

```
-- Verify that this is a supported version of MySQL.

select msg as `==> Checking current version of MySQL.`
from
  (
  select
    'This script should be run on MySQL version 5.6 or higher. ' +
    'Earlier versions are not supported.' as msg,
    cast(substring_index(version(), '.', 1) as unsigned) * 100 +
      cast(substring_index(substring_index(version(), '.', 2), '.', -1)
      as unsigned)
    as major_minor
  ) as T
where major_minor <> 506;


-- List MyISAM and compressed tables. Include the table size.

select concat(TABLE_SCHEMA, '.', TABLE_NAME) as `==> MyISAM or Compressed Tables`,
round(((data_length + index_length) / 1024 / 1024), 2) "Approx size (MB)"
from INFORMATION_SCHEMA.TABLES
where
  ENGINE <> 'InnoDB'
  and
  (
    -- User tables
    TABLE_SCHEMA not in ('mysql', 'performance_schema',
                         'information_schema')
    or
    -- Non-standard system tables
    (
      TABLE_SCHEMA = 'mysql' and TABLE_NAME not in
        (
          'columns_priv', 'db', 'event', 'func', 'general_log',
          'help_category', 'help_keyword', 'help_relation',
          'help_topic', 'host', 'ndb_binlog_index', 'plugin',
          'proc', 'procs_priv', 'proxies_priv', 'servers', 'slow_log',
          'tables_priv', 'time_zone', 'time_zone_leap_second',
          'time_zone_name', 'time_zone_transition',
          'time_zone_transition_type', 'user'
        )
    )
  )
  or
```

```
  (
    -- Compressed tables
       ROW_FORMAT = 'Compressed'
  );
```

脚本会生成类似于以下示例的输出。该示例显示了两个必须从 MyISAM 转换为 InnoDB 的表。输出还包括每个表的相应大小（以 MB 为单位）。

```
+---------------------------------+------------------+
| ==> MyISAM or Compressed Tables | Approx size (MB) |
+---------------------------------+------------------+
| test.name_table                 |          2102.25 |
| test.my_table                   |            65.25 |
+---------------------------------+------------------+
2 rows in set (0.01 sec)
```

将 RDS for MySQL 数据库快照迁移到 Aurora MySQL 数据库集群

您可以使用 AWS Management Console或 AWS CLI 迁移 RDS for MySQL 数据库实例的数据库快照来创建 Aurora MySQL 数据库集群。将使用原始 RDS for MySQL 数据库实例中的数据填充新的 Aurora MySQL 数据库集群。有关创建数据库快照的信息，请参阅创建数据库快照。

如果数据库快照不在要从中找到数据的 AWS 区域中，请将数据库快照复制到该 AWS 区域。有关复制数据库快照的信息，请参阅复制数据库快照。

控制台

使用 AWS Management Console迁移数据库快照时，控制台将执行必要的操作来仅创建数据库集群。

您还可以选择使用 AWS KMS key 静态加密新的 Aurora MySQL DB 数据库集群。

使用 AWS Management Console 迁移 MySQL 数据库快照

1. 登录AWS Management Console并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 从 MySQL 数据库实例或快照中开始迁移：

   要从数据库实例中开始迁移，请执行以下操作：

   1. 在导航窗格中，选择 Databases (数据库)，然后选择 MySQL 数据库实例。
   2. 对于 Actions (操作)，选择 Migrate latest snapshot (迁移最新快照)。

要从快照中开始迁移，请执行以下操作：

1. 选择 Snapshots。
2. 在 Snapshots 页面上，选择要迁移到 Aurora MySQL 数据库集群的快照。
3. 选择快照操作，然后选择迁移快照。

将显示迁移数据库页。

3. 在 Migrate Database 页面上设置以下值：

   - 迁移到数据库引擎：选择 `aurora`。
   - 数据库引擎版本：选择 Aurora MySQL 数据库集群的数据库引擎版本。
   - 数据库实例类：选择具有数据库所需的存储和容量的数据库实例类，例如 `db.r3.large`。Aurora 集群卷随着数据库中数据量的增加自动增大。Aurora 集群卷可增大到最大大小 128 tebibytes (TiB)。因此，您只需选择满足当前存储要求的数据库实例类。有关更多信息，请参阅“Amazon Aurora 存储概述”。
   - DB Instance Identifier（数据库实例标识符）：为数据库集群键入一个名称，该名称在您选择的 AWS 区域中对于您的账户是唯一的。此标识符将在数据库集群中实例的终端节点地址中使用。您可选择对该名称进行一些巧妙处理，例如将所选的 AWS 区域和数据库引擎包括在名称中（如 **`aurora-cluster1`**）。

     数据库实例标识符具有以下限制：

     - 它必须包含 1 到 63 个字母数字字符或连字符。
     - 它的第一个字符必须是字母。
     - 它不能以连字符结束或包含两个连续连字符。
     - 对于每个 AWS 区域的每个 AWS 账户的所有数据库实例必须是唯一的。
   - Virtual Private Cloud (VPC)：如果已具有 VPC，您可以选择 VPC 标识符（如 `vpc-a464d1c1`）以将该 VPC 用于 Aurora MySQL 数据库集群。有关创建 VPC 的信息，请参阅教程：创建 VPC 以用于数据库集群（仅限 IPv4）。

     否则，可以通过选择 Create a new VPC (新建 VPC)，让 Aurora 为您创建 VPC。
   - DB subnet group（数据库子网组）：如果已具有子网组，您可以选择子网组标识符（如 `gs-subnet-group1`）以将该子网组用于 Aurora MySQL 数据库集群。

     否则，可以通过选择 Create a new subnet group (新建子网组)，让 Aurora 为您创建子网组。

- 公有可访问性：选择否以指定仅 VPC 中的资源可以访问数据库集群中的实例。选择 Yes 可指定数据库集群中的实例可以由公用网络上的资源访问。默认值为 Yes。

  > **Note**
  >
  > 您的生产数据库集群可能不需要位于公有子网中，因为仅应用程序服务器将需要访问数据库集群。如果数据库集群不需要位于公有子网中，请将 Publicly Accessible 设置为 No。

- 可用区：选择可用区以托管 Aurora MySQL 数据库集群的主实例。要让 Aurora 选择可用区，请选择 No Preference (无首选项)。
- 数据库端口：键入在连接到 Aurora MySQL 数据库集群中的实例时使用的默认端口。默认为 `3306`。

  > **Note**
  >
  > 您可能位于企业防火墙后面，该防火墙不允许访问默认端口 (例如，MySQL 默认端口 3306)。在此情况下，请提供企业防火墙允许的端口值。请记住此端口值，以便在稍后连接到 Aurora MySQL 数据库集群时使用。

- 加密：为要静态加密的新 Aurora MySQL 数据库集群选择启用加密。如果选择启用加密，您必须选择一个 KMS 密钥以作为 AWS KMS key 值。

  如果数据库快照未加密，请指定一个加密密钥来对数据库集群进行静态加密。

  如果数据库快照已加密，请指定一个加密密钥，以便使用指定的加密密钥对您的数据库集群进行静态加密。您可以指定数据库快照使用的加密密钥或其他密钥。您无法从加密的数据库快照创建未加密的数据库集群。

- 自动次要版本升级：该设置不适用于 Aurora MySQL 数据库集群。

  有关 Aurora MySQL 引擎更新的更多信息，请参阅Amazon Aurora MySQL 的数据库引擎更新。

4. 选择 Migrate 以迁移您的数据库快照。
5. 选择 Instances，然后选择箭头图标以显示数据库集群详细信息并监控迁移的进度。在详细信息页面上，您可以找到用于连接到数据库集群的主实例的集群终端节点。有关连接到 Aurora MySQL 数据库集群的更多信息，请参阅连接到 Amazon Aurora 数据库集群。

AWS CLI

您可以使用包含以下参数的 `restore-db-cluster-from-snapshot` 命令从 RDS for MySQL 数据库实例的数据库快照创建 Aurora 数据库集群：

- `--db-cluster-identifier` – 要创建的数据库集群的名称。
- `--engine aurora-mysql` – 适用于 MySQL 5.7 兼容或 8.0 兼容的数据库集群。
- `--kms-key-id` – 用于选择性加密数据库集群的 AWS KMS key，具体取决于是否加密了数据库快照。
  - 如果数据库快照未加密，请指定一个加密密钥来对数据库集群进行静态加密。否则，不会对您的数据库集群进行加密。
  - 如果数据库快照已加密，请指定一个加密密钥，以便使用指定的加密密钥对您的数据库集群进行静态加密。否则，您的数据库集群将使用数据库快照的加密密钥进行静态加密。

  > **Note**
  >
  > 您无法从加密的数据库快照创建未加密的数据库集群。

- `--snapshot-identifier`：待迁移数据库快照的 Amazon Resource Name (ARN)。有关 Amazon RDS ARN 的更多信息，请参阅 Amazon Relational Database Service (Amazon RDS)。

在使用 `RestoreDBClusterFromSnapshot` 命令迁移数据库快照时，该命令将创建数据库集群和主实例。

在该示例中，您从将 ARN 设置为 *`mydbsnapshotARN`* 的数据库快照创建一个名为 *`mydbcluster`* 的 MySQL 5.7 兼容数据库集群。

对于 Linux、macOS 或 Unix：

```
aws rds restore-db-cluster-from-snapshot \
    --db-cluster-identifier mydbcluster \
    --snapshot-identifier mydbsnapshotARN \
    --engine aurora-mysql
```

对于 Windows：

```
aws rds restore-db-cluster-from-snapshot ^
    --db-cluster-identifier mydbcluster ^
```

```
    --snapshot-identifier mydbsnapshotARN ^
    --engine aurora-mysql
```

在该示例中，您从将 ARN 设置为 *mydbsnapshotARN* 的数据库快照创建一个名为 *mydbcluster* 的 MySQL 5.7 兼容数据库集群。

对于 Linux、macOS 或 Unix：

```
aws rds restore-db-cluster-from-snapshot \
    --db-cluster-identifier mydbcluster \
    --snapshot-identifier mydbsnapshotARN \
    --engine aurora-mysql
```

对于 Windows：

```
aws rds restore-db-cluster-from-snapshot ^
    --db-cluster-identifier mydbcluster ^
    --snapshot-identifier mydbsnapshotARN ^
    --engine aurora-mysql
```

使用 Aurora 只读副本将数据从 RDS for MySQL 数据库实例迁移到 Amazon Aurora MySQL 数据库集群

Aurora 使用 MySQL 数据库引擎的二进制日志复制功能，为源 RDS for MySQL 数据库实例创建一个特殊类型的数据库集群，称为 Aurora 只读副本。对源 RDS for MySQL 数据库实例的更新将异步复制到 Aurora 只读副本。

我们建议您创建源 RDS for MySQL 数据库实例的 Aurora 只读副本，以使用该功能从 RDS for MySQL 数据库实例迁移到 Aurora MySQL 数据库集群。如果 RDS for MySQL 数据库实例和 Aurora 只读副本之间的副本滞后为 0，您可以将客户端应用程序引导到 Aurora 只读副本，然后停止复制以使 Aurora 只读副本成为独立的 Aurora MySQL 数据库集群。准备迁移需要花费一段时间，每 TiB 数据大约需要几小时。

有关 Aurora 可用的区域的列表，请参阅《AWS 一般参考》中的 Amazon Aurora。

在创建 RDS for MySQL 数据库实例的 Aurora 只读副本时，Amazon RDS 为源 RDS for MySQL 数据库实例创建数据库快照（Amazon RDS 私有，不产生费用）。Amazon RDS 之后会将数据从数据库快照迁移到 Aurora 只读副本。在将数据从数据库快照迁移到新的 Aurora MySQL 数据库集群后，Amazon RDS 开始在 RDS for MySQL 数据库实例和 Aurora MySQL 数据库集群之间进行复制。如果 RDS for MySQL 数据库实例中包含的表使用 InnoDB 之外的存储引擎，或者使用压缩行格式，您可以在创建 Aurora 只读副本之前更改这些表以使用 InnoDB 存储引擎和动态行格式，从而加快创建 Aurora 只读副本的过程。有关将 MySQL 数据库快照复制到 Aurora MySQL 数据库集群的过程的更多信息，请参阅将数据从 RDS for MySQL DB 数据库实例迁移到 Amazon Aurora MySQL 数据库集群。

一个 RDS for MySQL 数据库实例只能有一个 Aurora 只读副本。

> **Note**
>
> 由于 Aurora MySQL 和作为复制主实例的 RDS for MySQL 数据库实例的 MySQL 数据库引擎版本之间的特征不同，可能导致复制问题。如果遇到错误，您可以在 Amazon RDS 社群论坛或联系 AWS 支持 以获得帮助。
>
> 如果您的 RDS for MySQL 数据库实例已经是跨区域只读副本的来源，则无法创建 Aurora 只读副本。
>
> 您无法从一些较早的 RDS for MySQL 8.0 版本（包括 8.0.11、8.0.13 和 8.0.15）迁移到 Aurora MySQL 版本 3.05 及更高版本。我们建议您在迁移之前先升级到 RDS for MySQL 版本 8.0.28。

有关 MySQL 只读副本的更多信息，请参阅使用 MariaDB、MySQL 和 PostgreSQL 数据库实例的只读副本。

创建 Aurora 只读副本

您可以使用控制台、AWS CLI 或 RDS API 为 RDS for MySQL 数据库实例创建 Aurora 只读副本。

控制台

从源 RDS for MySQL 数据库实例创建 Aurora 只读副本

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Databases (数据库)。
3. 选择要作为 Aurora 只读副本源的 MySQL 数据库实例。
4. 对于操作，请选择创建 Aurora 只读副本。
5. 按照下表所述，选择 Aurora 只读副本要使用的数据库集群规格。

选项	描述
数据库实例类	选择定义数据库集群中主实例的处理和内存要求的数据库实例类。有关数据库实例类选项的更多信息，请参阅Amazon Aurora 数据库实例类。
多可用区部署	选择 Create Replica in Different Zone（在不同区域创建副本）可在目标 AWS 区域的另一个可用区中创建新数据库集群的备用副本以支持故障转移。有关多可用区的详细信息，请参阅区域及可用区。
数据库实例标识符	键入 Aurora 只读副本数据库集群中主实例的名称。此标识符在新数据库集群主实例的终端节点地址中使用。 数据库实例标识符具有以下限制： • 它必须包含 1 到 63 个字母数字字符或连字符。 • 它的第一个字符必须是字母。 • 它不能以连字符结束或包含两个连续连字符。

选项	描述
	• 它对于每个 AWS 区域每个 AWS 账户的所有数据库实例必须是唯一的。 由于 Aurora 只读副本数据库集群是从源数据库实例快照创建的，因此，Aurora 只读副本的主用户名和主密码与源数据库实例的主用户名和主密码相同。
Virtual Private Cloud (VPC)	选择要托管数据库集群的 VPC。选择 Create new VPC (新建 VPC) 以让 Aurora 为您创建 VPC。有关更多信息，请参阅 数据库集群先决条件。
DB subnet group（数据库子网组）	选择要用于数据库集群的数据库子网组。选择 Create new DB subnet group (创建新的数据库子网组) 以让 Aurora 为您创建数据库子网组。有关更多信息，请参阅“数据库集群先决条件”。
公开可用性	选择 Yes 可向数据库集群提供公有 IP 地址；否则，请选择 No。数据库集群可以混合使用公有和私有数据库实例。有关隐藏实例以防止公开访问的更多信息，请参阅对互联网隐藏 VPC 中的数据库集群。
可用区	确定您是否希望指定特定的可用区。有关可用区的更多信息，请参阅区域及可用区。
VPC security group (firewall) [VPC 安全组（防火墙）]	选择 Create new VPC security group (新建 VPC 安全组) 以让 Aurora 为您创建 VPC 安全组。选择选择现有的 VPC 安全组，然后指定一个或多个 VPC 安全组以保护对数据库集群的网络访问。有关更多信息，请参阅“数据库集群先决条件”。
数据库端口	指定应用程序和实用程序用来访问数据库的端口。Aurora MySQL 数据库集群默认为使用默认 MySQL 端口 3306。有些公司的防火墙不允许连接到此端口。如果您的公司防火墙阻止使用默认端口，请为新数据库集群选择其他端口。

选项	描述
数据库参数组	为 Aurora MySQL 集群选择数据库参数组。Aurora 具有一个可使用的默认数据库参数组，您也可以创建自己的数据库参数组。有关数据库参数组的更多信息，请参阅 Amazon Aurora 的参数组。
数据库集群参数组	为 Aurora MySQL 集群选择数据库集群参数组。Aurora 具有一个可使用的默认数据库集群参数组，您也可以创建自己的数据库集群参数组。有关数据库集群参数组的更多信息，请参阅 Amazon Aurora 的参数组。
加密	如果您不希望对新的 Aurora 数据库集群进行加密，请选择禁用加密。为要静态加密的新 Aurora 数据库集群选择启用加密。如果选择启用加密，您必须选择一个 KMS 密钥以作为 AWS KMS key 值。 如果 MySQL 数据库实例未加密，请指定一个加密密钥来对数据库集群进行静态加密。 如果 MySQL 数据库实例已加密，请指定一个加密密钥，以便使用指定的加密密钥对您的数据库集群进行静态加密。您可以指定 MySQL 数据库快照使用的加密密钥或其他密钥。您无法从已加密的 MySQL 数据库实例创建未加密的数据库集群。
优先级	选择数据库集群的故障转移优先级。如果您未选择值，则默认值为 tier-1。此优先级决定从主实例故障恢复时提升 Aurora 副本的顺序。有关更多信息，请参阅“Aurora 数据库集群的容错能力”。
备份保留期	选择 Aurora 保留数据库的备份副本的时间长度 (1 到 35 天)。可使用备份副本对数据库执行时间点还原 (PITR)，以还原到第二个时间点。
增强监控	选择启用增强监控可启用您的数据库集群在其上运行的操作系统的实时指标收集。有关更多信息，请参阅“使用增强监控来监控操作系统指标”。

选项	描述
监控角色	仅当增强监控设置为启用增强监控时可用。选择您创建的 IAM 角色以允许 Aurora 与 Amazon CloudWatch Logs 通信，或选择 Default (默认) 以让 Aurora 为您创建一个名为 `rds-monitoring-role` 的角色。有关更多信息，请参阅“使用增强监控来监控操作系统指标”。
粒度	仅当增强监控设置为启用增强监控时可用。设置为数据库集群收集指标的时间间隔（以秒为单位）。
自动次要版本升级	该设置不适用于 Aurora MySQL 数据库集群。 有关 Aurora MySQL 引擎更新的更多信息，请参阅Amazon Aurora MySQL 的数据库引擎更新。
维护窗口	选择选择时段并指定可以进行系统维护的每周时间范围。或者，为 Aurora 选择 No preference (无首选项) 以随机分配一个时段。

6. 选择 Create read replica (创建只读副本)。

AWS CLI

要从源 RDS for MySQL 数据库实例中创建 Aurora 只读副本，请使用 `create-db-cluster` 和 `create-db-instance` AWS CLI 命令创建新的 Aurora MySQL 数据库集群。当您调用 `create-db-cluster` 命令时，请加入 `--replication-source-identifier` 参数，识别源 MySQL 数据库实例的 Amazon Resource Name (ARN)。有关 Amazon RDS ARN 的更多信息，请参阅 Amazon Relational Database Service (Amazon RDS)。

不要指定主用户名、主密码或数据库名称，因为 Aurora 只读副本使用和源 MySQL 数据库实例相同的主用户名、主密码和数据库名称。

对于 Linux、macOS 或 Unix：

```
aws rds create-db-cluster --db-cluster-identifier sample-replica-cluster --engine
 aurora \
    --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2 \
```

```
    --replication-source-identifier arn:aws:rds:us-west-2:123456789012:db:primary-
mysql-instance
```

对于 Windows：

```
aws rds create-db-cluster --db-cluster-identifier sample-replica-cluster --engine
 aurora ^
    --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2 ^
    --replication-source-identifier arn:aws:rds:us-west-2:123456789012:db:primary-
mysql-instance
```

如果您使用控制台创建 Aurora 只读副本，Aurora 将自动为您的数据库集群 Aurora 只读副本创建主实例。如果您使用 AWS CLI 创建 Aurora 只读副本，则必须明确为数据库集群创建主实例。主实例是在数据库集群中创建的第一个实例。

您可以使用带以下参数的 `create-db-instance` AWS CLI 命令为数据库集群创建主实例。

- `--db-cluster-identifier`

  数据库集群的名称。

- `--db-instance-class`

  要用于主实例的数据库实例类的名称。

- `--db-instance-identifier`

  主实例的名称。

- `--engine aurora`

在该示例中，您将使用 *myinstanceclass* 中指定的数据库实例类为名为 *myreadreplicacluster* 的数据库集群创建一个名为 *myreadreplicainstance* 的主实例。

Example

对于 Linux、macOS 或 Unix：

```
aws rds create-db-instance \
    --db-cluster-identifier myreadreplicacluster \
    --db-instance-class myinstanceclass \
    --db-instance-identifier myreadreplicainstance \
```

```
    --engine aurora
```

对于 Windows：

```
aws rds create-db-instance ^
    --db-cluster-identifier myreadreplicacluster ^
    --db-instance-class myinstanceclass ^
    --db-instance-identifier myreadreplicainstance ^
    --engine aurora
```

RDS API

要从源 RDS for MySQL 数据库实例中创建 Aurora 只读副本，请使用 `CreateDBCluster` 和 `CreateDBInstance` Amazon RDS API 命令创建新的 Aurora 数据库集群和主实例。不要指定主用户名、主密码或数据库名称，因为 Aurora 只读副本使用和源 RDS for MySQL 数据库实例相同的主用户名、主密码和数据库名称。

您可以使用带以下参数的 `CreateDBCluster` Amazon RDS API 命令从源 RDS for MySQL 数据库实例为 Aurora 只读副本创建新的 Aurora 数据库集群：

- `DBClusterIdentifier`

  要创建的数据库集群的名称。

- `DBSubnetGroupName`

  要与该数据库集群关联的数据库子网组的名称。

- `Engine=aurora`
- `KmsKeyId`

  用于选择性加密数据库集群的 AWS KMS key，具体取决于是否加密了 MySQL 数据库实例。

  - 如果 MySQL 数据库实例未加密，请指定一个加密密钥来对数据库集群进行静态加密。否则，将使用您的账户的默认加密密钥对数据库集群进行静态加密。
  - 如果 MySQL 数据库实例已加密，请指定一个加密密钥，以便使用指定的加密密钥对您的数据库集群进行静态加密。否则，将使用 MySQL 数据库实例的加密密钥对您的数据库集群进行静态加密。

  > **Note**
  >
  > 您无法从已加密的 MySQL 数据库实例创建未加密的数据库集群。

- `ReplicationSourceIdentifier`

  源 MySQL 数据库实例的 Amazon Resource Name (ARN)。有关 Amazon RDS ARN 的更多信息，请参阅 Amazon Relational Database Service (Amazon RDS)。

- `VpcSecurityGroupIds`

  要与该数据库集群关联的 EC2 VPC 安全组列表。

在该示例中，您将从 ARN 设置为 *mysqlprimaryARN*、与名为 *mysubnetgroup* 的数据库子网组和名为 *mysecuritygroup* 的 VPC 安全组关联的源 MySQL 数据库实例创建一个名为 *myreadreplicacluster* 的数据库集群。

Example

```
https://rds.us-east-1.amazonaws.com/
    ?Action=CreateDBCluster
    &DBClusterIdentifier=myreadreplicacluster
    &DBSubnetGroupName=mysubnetgroup
    &Engine=aurora
    &ReplicationSourceIdentifier=mysqlprimaryARN
    &SignatureMethod=HmacSHA256
    &SignatureVersion=4
    &Version=2014-10-31
    &VpcSecurityGroupIds=mysecuritygroup
    &X-Amz-Algorithm=AWS4-HMAC-SHA256
    &X-Amz-Credential=AKIADQKE4SARGYLE/20150927/us-east-1/rds/aws4_request
    &X-Amz-Date=20150927T164851Z
    &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
    &X-Amz-Signature=6a8f4bd6a98f649c75ea04a6b3929ecc75ac09739588391cd7250f5280e716db
```

如果您使用控制台创建 Aurora 只读副本，Aurora 将自动为您的数据库集群 Aurora 只读副本创建主实例。如果您使用 AWS CLI 创建 Aurora 只读副本，则必须明确为数据库集群创建主实例。主实例是在数据库集群中创建的第一个实例。

您可以使用带以下参数的 `CreateDBInstance` Amazon RDS API 命令为数据库集群创建主实例：

- `DBClusterIdentifier`

  数据库集群的名称。

- `DBInstanceClass`

要用于主实例的数据库实例类的名称。

- `DBInstanceIdentifier`

  主实例的名称。

- `Engine=aurora`

在该示例中，您将使用 *myinstanceclass* 中指定的数据库实例类为名为 *myreadreplicacluster* 的数据库集群创建一个名为 *myreadreplicainstance* 的主实例。

Example

```
https://rds.us-east-1.amazonaws.com/
    ?Action=CreateDBInstance
    &DBClusterIdentifier=myreadreplicacluster
    &DBInstanceClass=myinstanceclass
    &DBInstanceIdentifier=myreadreplicainstance
    &Engine=aurora
    &SignatureMethod=HmacSHA256
    &SignatureVersion=4
    &Version=2014-09-01
    &X-Amz-Algorithm=AWS4-HMAC-SHA256
    &X-Amz-Credential=AKIADQKE4SARGYLE/20140424/us-east-1/rds/aws4_request
    &X-Amz-Date=20140424T194844Z
    &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
    &X-Amz-Signature=bee4aabc750bf7dad0cd9e22b952bd6089d91e2a16592c2293e532eeaab8bc77
```

查看 Aurora 只读副本

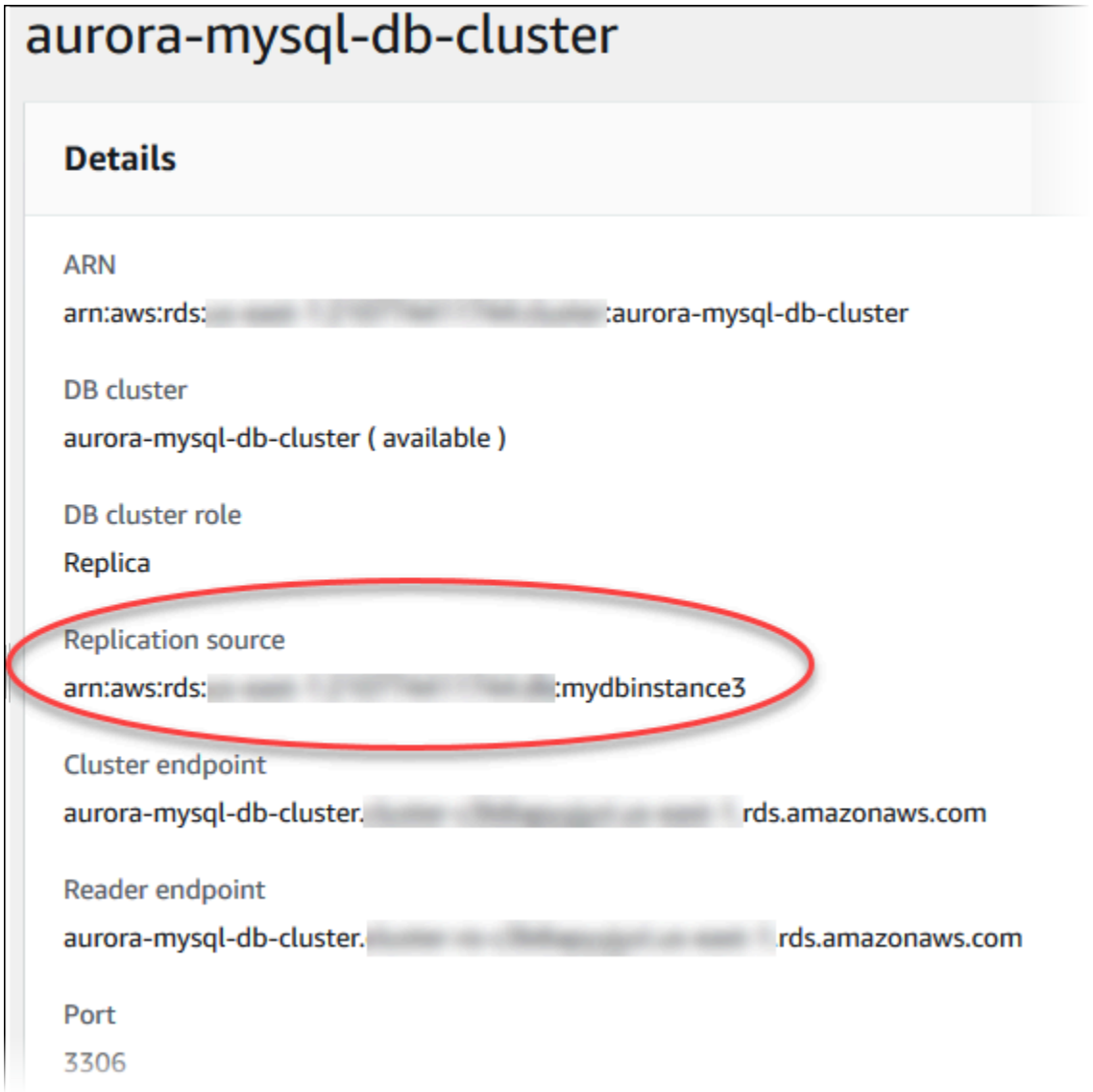
您可以使用 AWS Management Console 或 AWS CLI 查看 Aurora 数据库集群的 MySQL 与 Aurora MySQL 的复制关系。

控制台

要查看 Aurora 只读副本的主 MySQL 数据库实例

1. 登录AWS Management Console并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Databases (数据库)。

3. 选择 Aurora 只读副本的数据库集群以显示其详细信息。主 MySQL 数据库实例信息位于 Replication source（复制源）字段中。



AWS CLI

要使用 AWS CLI 针对您的 Aurora MySQL 数据库集群查看 MySQL 与 Aurora MySQL 的复制关系，请使用 `describe-db-clusters` 和 `describe-db-instances` 命令。

要确定哪个 MySQL 数据库实例是主实例，请使用 `describe-db-clusters` 并在 `--db-cluster-identifier` 选项中指定 Aurora 只读副本的集群标识符。请参阅作为复制主实例的数据库实例的 ARN 输出中的 `ReplicationSourceIdentifier` 元素。

要确定哪个数据库集群是 Aurora 只读副本，请使用 describe-db-instances 并在 --db-instance-identifier 选项中指定 MySQL 数据库实例的实例标识符。请参阅 Aurora 只读副本的数据库集群标识符输出中的 ReadReplicaDBClusterIdentifiers 元素。

Example

对于 Linux、macOS 或 Unix：

```
aws rds describe-db-clusters \
    --db-cluster-identifier myreadreplicacluster
```

```
aws rds describe-db-instances \
    --db-instance-identifier mysqlprimary
```

对于 Windows：

```
aws rds describe-db-clusters ^
    --db-cluster-identifier myreadreplicacluster
```

```
aws rds describe-db-instances ^
    --db-instance-identifier mysqlprimary
```

提升 Aurora 只读副本

迁移完成后，您可以使用 AWS Management Console或 AWS CLI 将 Aurora 只读副本提升为独立的数据库集群。

然后，您可以将客户端应用程序引导到 Aurora 只读副本的端点。有关 Aurora 终端节点的更多信息，请参阅Amazon Aurora 端点连接。提升会很快完成，您可以在提升过程中读取和写入 Aurora 只读副本。但是在此期间您无法删除 MySQL 主数据库实例，或取消数据库实例和 Aurora 只读副本之间的关联。

在提升 Aurora 只读副本之前，终止写入到源 MySQL 数据库实例的所有事务，然后等待 Aurora 只读副本的副本滞后达到 0。您可以通过对 Aurora 只读副本调用 SHOW SLAVE STATUS（Aurora MySQL 版本 2）或 SHOW REPLICA STATUS（Aurora MySQL 版本 3）命令，以查看 Aurora 只读副本的副本滞后。检查 Seconds behind master 值。

写入主实例的事务终止，且副本滞后为 0 时，可以开始写入 Aurora 只读副本。如果在此之前写入到 Aurora 只读副本，并修改了在 MySQL 主实例上也进行修改的表，则可能会中断复制到 Aurora 的过程。如果发生此情况，您必须删除并重新创建 Aurora 只读副本。

控制台

将 Aurora 只读副本提升为 Aurora 数据库集群

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Databases (数据库)。
3. 选择 Aurora 只读副本的数据库集群。
4. 对于操作，请选择提升。
5. 选择 Promote Read Replica（提升只读副本）。

升级后，请使用以下步骤确认提升已完成。

要确认 Aurora 只读副本已提升

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Events。
3. 在 Events（事件）页面上，验证您提升的集群是否存在 `Promoted Read Replica cluster to a stand-alone database cluster` 事件。

提升完成后，MySQL 主数据库实例和 Aurora 只读副本之间的关联会取消，如果您愿意，可以安全删除数据库实例。

AWS CLI

要将 Aurora 只读副本提升为独立数据库集群，请使用 `promote-read-replica-db-cluster`AWS CLI 命令。

Example

对于 Linux、macOS 或 Unix：

```
aws rds promote-read-replica-db-cluster \
    --db-cluster-identifier myreadreplicacluster
```

对于 Windows：

```
aws rds promote-read-replica-db-cluster ^
    --db-cluster-identifier myreadreplicacluster
```

管理 Amazon Aurora MySQL

以下各部分介绍管理 Amazon Aurora MySQL 数据库集群。

主题

- 管理 Amazon Aurora MySQL 性能和扩展
- 回溯 Aurora 数据库集群
- 使用错误注入查询测试 Amazon Aurora MySQL
- 使用快速 DDL 在 Amazon Aurora 中修改表
- 显示 Aurora MySQL 数据库集群的卷状态

管理 Amazon Aurora MySQL 性能和扩展

扩展 Aurora MySQL 数据库实例

您可通过两种方式扩展 Aurora MySQL 数据库实例，即实例扩展和读取扩展。有关读取扩展的更多信息，请参阅读取扩展。

您可以通过修改数据库集群中每个数据库实例的数据库实例类，来扩展 Aurora MySQL 数据库集群。Aurora MySQL 支持多种针对 Aurora 优化的数据库实例类。不要将 db.t2 或 db.t3 实例类用于大小大于 40 TB 的较大 Aurora 集群。有关 Aurora MySQL 支持的数据库实例类的规格，请参阅Amazon Aurora 数据库实例类。

> **Note**
>
> 建议仅将 T 数据库实例类用于开发和测试服务器，或其他非生产服务器。有关 T 实例类的更多详细信息，请参阅使用 T 实例类进行开发和测试。

至 Aurora MySQL 数据库实例的最大连接数

允许连接到 Aurora MySQL 数据库实例的最大数量由数据库实例的实例级参数组中的 `max_connections` 参数确定。

下表列出了可用于 Aurora MySQL 的每个数据库实例类的结果 `max_connections` 默认值。您可通过增加内存将实例缩放为数据库实例类，或通过将实例的数据库参数组中的 `max_connections` 参数设置为更大的值（最大为 16,000）来增加至您的 Aurora MySQL 数据库实例的最大连接数。

> **Tip**
>
> 如果您的应用程序经常打开和关闭连接，或使大量长期连接保持打开，建议您使用 Amazon RDS 代理。RDS 代理是一种完全托管的高可用性数据库代理，它使用连接池安全有效地共享数据库连接。要了解有关 RDS 代理的更多信息，请参阅 适用于 Aurora 的 Amazon RDS 代理。

有关 Aurora Serverless v2 实例如何处理此参数的详细信息，请参阅 Aurora Serverless v2 的最大连接数。

实例类	max_connections 默认值		
db.t2.small	45		
db.t2.medium	90		
db.t3.small	45		
db.t3.medium	90		
db.t3.large	135		
db.t4g.medium	90		
db.t4g.large	135		
db.r3.large	1000		
db.r3.xlarge	2000		
db.r3.2xlarge	3000		
db.r3.4xlarge	4000		
db.r3.8xlarge	5000		
db.r4.large	1000		

实例类	max_connections 默认值		
db.r4.xlarge	2000		
db.r4.2xlarge	3000		
db.r4.4xlarge	4000		
db.r4.8xlarge	5000		
db.r4.16xlarge	6000		
db.r5.large	1000		
db.r5.xlarge	2000		
db.r5.2xlarge	3000		
db.r5.4xlarge	4000		
db.r5.8xlarge	5000		
db.r5.12xlarge	6000		
db.r5.16xlarge	6000		
db.r5.24xlarge	7000		
db.r6g.large	1000		
db.r6g.xlarge	2000		
db.r6g.2xlarge	3000		
db.r6g.4xlarge	4000		
db.r6g.8xlarge	5000		
db.r6g.12xlarge	6000		

实例类	max_connections 默认值		
db.r6g.16xlarge	6000		
db.r6i.large	1000		
db.r6i.xlarge	2000		
db.r6i.2xlarge	3000		
db.r6i.4xlarge	4000		
db.r6i.8xlarge	5000		
db.r6i.12xlarge	6000		
db.r6i.16xlarge	6000		
db.r6i.24xlarge	7000		
db.r6i.32xlarge	7000		
db.r7g.large	1000		
db.r7g.xlarge	2000		
db.r7g.2xlarge	3000		
db.r7g.4xlarge	4000		
db.r7g.8xlarge	5000		
db.r7g.12xlarge	6000		
db.r7g.16xlarge	6000		
db.r7i.large	1000		
db.r7i.xlarge	2000		

实例类	max_connections 默认值		
db.r7i.2xlarge	3000		
db.r7i.4xlarge	4000		
db.r7i.8xlarge	5000		
db.r7i.12xlarge	6000		
db.r7i.16xlarge	6000		
db.r7i.24xlarge	7000		
db.r7i.48xlarge	8000		
db.r8g.large	1000		
db.r8g.xlarge	2000		
db.r8g.2xlarge	3000		
db.r8g.4xlarge	4000		
db.r8g.8xlarge	5000		
db.r8g.12xlarge	6000		
db.r8g.16xlarge	6000		
db.r8g.24xlarge	7000		
db.r8g.48xlarge	8000		
db.x2g.large	2000		
db.x2g.xlarge	3000		
db.x2g.2xlarge	4000		

实例类	max_connections 默认值		
db.x2g.4xlarge	5000		
db.x2g.8xlarge	6000		
db.x2g.12xlarge	7000		
db.x2g.16xlarge	7000		

> **Tip**
>
> 对于所选 Aurora MySQL 实例类，`max_connections` 参数计算使用对数底数 2（不同于自然对数）和 `DBInstanceClassMemory` 值（以字节为单位）。该参数仅接受整数值，并从计算中截断小数部分。该公式实施了连接限制，如下所示：
>
> - 对于较大的 R3、R4 和 R5 实例，连接增量为 1000
> - 对于 T2 和 T3 实例内存变体，连接增量为 45
>
> 示例：对于 db.r6g.large，当公式计算 1069.2 时，系统实施 1000 以保持一致的增量模式。

如果创建新的参数组以自定义您自己的连接限制默认值，您将看到默认连接限制是使用基于 `DBInstanceClassMemory` 值的公式得出的。正如上表中所示，由于逐渐增大的 R3、R4 和 R5 实例之间的内存加倍，该公式生成的连接限制将增加 1000；而对于 T2 和 T3 实例的不同内存大小，该限制将增加 45。

请参阅指定数据库参数，详细了解 `DBInstanceClassMemory` 的计算方式。

Aurora MySQL 和 RDS for MySQL 数据库实例的内存开销不同。因此，对于使用相同实例类的 Aurora MySQL 和 RDS for MySQL 数据库实例，`max_connections` 值可能不同。表中的值仅适用于 Aurora MySQL 数据库实例。

> **Note**
>
> T2 和 T3 实例的连接限制低得多是因为，对于 Aurora，这些实例类仅用于开发和测试方案，而不用于生产工作负载。

在对其他主要内存使用者（例如，缓冲池和查询缓存）使用默认值的系统中，将调整默认连接限制。如果为集群更改这些其他设置，请考虑调整连接限制以将数据库实例上的可用内存增加或减少情况考虑在内。

Aurora MySQL 的临时存储限制

Aurora MySQL 将表和索引储存在 Aurora 存储子系统中。Aurora MySQL 对非持久性临时文件和非 InnoDB 临时表使用单独的临时或本地存储。本地存储还包括用于在查询处理过程中对大型数据集进行排序或者用于索引构建操作等用途的文件。它不包括 InnoDB 临时表。

有关 Aurora MySQL 版本 3 中临时表的更多信息，请参阅 Aurora MySQL 版本 3 中的新临时表行为。有关版本 2 中临时表的更多信息，请参阅 Aurora MySQL 版本 2 中的临时表空间行为。

在启动和停止数据库实例以及更换主机期间，这些卷上的数据和临时文件会丢失。

这些本地存储卷由 Amazon Elastic Block Store（EBS）提供支持，并可以通过使用更大的数据库实例类来进行扩展。有关存储的更多信息，请参阅 Amazon Aurora 存储。

本地存储还用于使用 `LOAD DATA FROM S3` 或 `LOAD XML FROM S3` 从 Amazon S3 导入数据，以及使用 SELECT INTO OUTFILE S3 将数据导出到 S3。有关从 S3 导入和导出到 S3 的更多信息，请参阅以下内容：

- 将数据从 Amazon S3 存储桶中的文本文件加载到 Amazon Aurora MySQL 数据库集群
- 将数据从 Amazon Aurora MySQL 数据库集群保存到 Amazon S3 存储桶中的文本文件

Aurora MySQL 使用单独的永久存储空间来存储大多数 Aurora MySQL 数据库实例类的错误日志、常规日志、慢速查询日志和审计日志（不包括可突增性能实例类类型，例如 db.t2、db.t3 和 db.t4g）。在启动和停止数据库实例以及更换主机期间，会保留该卷上的数据。

此永久性存储卷还由 Amazon EBS 提供支持，并且根据数据库实例类别具有固定大小。不能通过使用更大的数据库实例类来对它进行扩展。

下表显示了每个 Aurora MySQL 数据库实例类可用的最大临时和永久性存储空间。有关 Aurora 的数据库实例类支持的更多信息，请参阅 Amazon Aurora 数据库实例类。

数据库实例类	最大可用临时/本地存储空间（GiB）	日志文件可用的额外最大存储空间（GiB）
db.x2g.16xlarge	1 280	500
db.x2g.12xlarge	960	500
db.x2g.8xlarge	640	500
db.x2g.4xlarge	320	500
db.x2g.2xlarge	160	60
db.x2g.xlarge	80	60
db.x2g.large	40	60
db.r8g.48xlarge	3840	500
db.r8g.24xlarge	1920	500
db.r8g.16xlarge	1 280	500
db.r8g.12xlarge	960	500
db.r8g.8xlarge	640	500
db.r8g.4xlarge	320	500
db.r8g.2xlarge	160	60
db.r8g.xlarge	80	60
db.r8g.large	32	60
db.r7i.48xlarge	3840	500
db.r7i.24xlarge	1920	500
db.r7i.16xlarge	1 280	500
db.r7i.12xlarge	960	500

数据库实例类	最大可用临时/本地存储空间（GiB）	日志文件可用的额外最大存储空间（GiB）
db.r7i.8xlarge	640	500
db.r7i.4xlarge	320	500
db.r7i.2xlarge	160	60
db.r7i.xlarge	80	60
db.r7i.large	32	60
db.r7g.16xlarge	1 280	500
db.r7g.12xlarge	960	500
db.r7g.8xlarge	640	500
db.r7g.4xlarge	320	500
db.r7g.2xlarge	160	60
db.r7g.xlarge	80	60
db.r7g.large	32	60
db.r6i.32xlarge	2560	500
db.r6i.24xlarge	1920	500
db.r6i.16xlarge	1 280	500
db.r6i.12xlarge	960	500
db.r6i.8xlarge	640	500
db.r6i.4xlarge	320	500
db.r6i.2xlarge	160	60
db.r6i.xlarge	80	60

数据库实例类	最大可用临时/本地存储空间（GiB）	日志文件可用的额外最大存储空间（GiB）
db.r6i.large	32	60
db.r6g.16xlarge	1 280	500
db.r6g.12xlarge	960	500
db.r6g.8xlarge	640	500
db.r6g.4xlarge	320	500
db.r6g.2xlarge	160	60
db.r6g.xlarge	80	60
db.r6g.large	32	60
db.r5.24xlarge	1920	500
db.r5.16xlarge	1 280	500
db.r5.12xlarge	960	500
db.r5.8xlarge	640	500
db.r5.4xlarge	320	500
db.r5.2xlarge	160	60
db.r5.xlarge	80	60
db.r5.large	32	60
db.r4.16xlarge	1 280	500
db.r4.8xlarge	640	500
db.r4.4xlarge	320	500
db.r4.2xlarge	160	60

数据库实例类	最大可用临时/本地存储空间（GiB）	日志文件可用的额外最大存储空间（GiB）
db.r4.xlarge	80	60
db.r4.large	32	60
db.t4g.large	32	–
db.t4g.medium	32	–
db.t3.large	32	–
db.t3.medium	32	–
db.t3.small	32	–
db.t2.medium	32	–
db.t2.small	32	–

Important

这些值表示每个数据库实例理论上的最大可用存储空间。可供您使用的实际本地存储空间可能会更小。Aurora 会在管理过程中占用一些本地存储，数据库实例甚至在加载任何数据之前也会占用一些本地存储。您可以使用 `FreeLocalStorage`CloudWatch 指标监控指定数据库实例可用的临时存储，如 Amazon Aurora 的 Amazon CloudWatch 指标 中所述。目前您可以检查空闲存储空间大小。您还可以随时间推移绘制空闲存储空间大小。监控一段时间内的空闲存储情况可以帮助您确定空闲存储空间大小是在增加还是减少，或者找到最小值、最大值或平均值。

（这不适用于 Aurora Serverless v2。）

回溯 Aurora 数据库集群

使用 Amazon Aurora MySQL 兼容版，您可以将数据库集群回溯到特定时间，而无需从备份还原数据。

目录

- 回溯概述
  - 回溯时段
  - 回溯时间
  - 回溯限制
- 区域和版本可用性
- 启用回溯的集群的升级注意事项
- 配置 Aurora MySQL 数据库集群回溯
- 对 Aurora MySQL 数据库集群执行回溯
- 监控 Aurora MySQL 数据库集群回溯
- 使用控制台订阅回溯事件
- 检索现有回溯
- 对 Aurora MySQL 数据库集群禁用回溯

回溯概述

回溯可以将数据库集群“倒回”到您指定的时间。回溯不是备份数据库集群以使您能够还原到某个时间点的替代方法。不过，相比传统备份和还原，回溯具有以下优势：

- 您可以轻松撤消错误。如果您错误地执行了破坏性操作，例如没有 WHERE 子句的 DELETE 操作，您可以通过尽可能减少服务中断的方式，将数据库集群回溯到执行破坏性操作之前的时间。
- 您可以快速回溯数据库集群。将数据库集群还原到某个时间点需要启动新数据库集群，然后从备份数据或数据库集群快照来还原它，这可能需要数个小时的时间。回溯数据库集群不需要新数据库集群，可在数分钟内倒回数据库集群。
- 您可以浏览以前的数据更改。您可以在时间点中向前和向后反复回溯数据库集群，帮助确定什么时候发生了特定数据更改。例如，您可以向后回溯数据库集群三个小时，然后向前回溯一个小时。在这种情况下，回溯时间为原始时间的两个小时之前。

> **Note**
>
> 有关将数据库集群回溯到某个时间点的信息，请参阅备份和还原 Aurora 数据库集群的概述。

回溯时段

使用回溯时，有一个目标回溯时段和一个实际回溯时段：

- 目标回溯时段是您希望数据库集群能够回溯的时间长度。在启用回溯时，您需要指定目标回溯时段。例如，如果您希望能够将数据库集群回溯 1 天，则指定 24 小时的目标回溯时段。
- 实际回溯时段是您可以实际回溯数据库集群的时间长度，这可能会小于目标回溯时段。实际回溯时段基于您的工作负载以及可用于存储数据库更改相关信息 (称为更改记录) 的存储。

当您在启用回溯的情况下对 Aurora 数据库集群进行更新时，会生成更改记录。Aurora 会保留目标回溯时段内的更改记录，您需要按小时支付这些记录的储存费用。目标回溯时段和数据库集群上的工作负载共同确定了您存储的更改记录数。工作负载是您在指定时间段内对数据库集群进行更改的数量。如果您的工作负载很重，会在回溯时段内存储相比工作负载较轻时更多的更改记录。

您可以将目标回溯时段看作您希望能够回溯数据库集群的最大时间长度。在很多情况下，您可以回溯自己指定的最大时间长度。不过，在一些情况下，数据库集群无法存储足够的更改记录以回溯最大时间长度，您的实际回溯时段小于目标回溯时段。通常，当数据库集群上有非常重的工作负载时，实际回溯时段小于目标回溯时段。在实际回溯时段小于目标回溯时段时，我们会向您发送通知。

为数据库集群启用了回溯时，如果您删除了存储在数据库集群中的表，Aurora 会在回溯更改记录中保留该表。通过这样做，您可以返回到删除表之前的时间。如果您的回溯时段中没有足够的空间来存储表，表最终可能会从回溯更改记录中删除。

回溯时间

Aurora 始终回溯到与数据库集群相一致的时间。这样做可以消除回溯完成时出现未提交事务的可能性。在您为回溯指定时间时，Aurora 自动选择尽可能接近的一致时间。此方法意味着已完成的回溯可能并非与您指定的时间完全一致，不过您可以使用 describe-db-cluster-backtracks AWS CLI 命令确定回溯的确切时间。有关更多信息，请参阅“检索现有回溯”。

回溯限制

以下限制适用于回溯：

- 回溯只能用于在启用回溯功能的情况下创建的数据库集群。您无法修改数据库集群来启用回溯功能。您可以在创建新数据库集群或还原数据库集群的快照时启用回溯功能。
- 回溯时段的限制为 72 小时。
- 回溯会影响整个数据库集群。例如，您无法选择性回溯单个表或单个数据更新。

- 您无法从启用回溯的集群创建跨区域只读副本，但您仍然可以在集群上启用二进制日志（binlog）复制。如果您尝试回溯启用了二进制日志记录的数据库集群，除非您选择强制执行回溯，否则通常会出错。任何强制执行回溯的尝试都将中断下游只读副本并干扰其它操作，例如蓝绿部署。
- 您无法将数据库克隆回溯到该数据库克隆创建之前的时间点。不过，您可以使用原始数据库回溯到创建克隆之前的时间。有关数据库克隆的更多信息，请参阅克隆 Amazon Aurora 数据库集群卷。
- 回溯会导致短暂的数据库实例中断。您必须先停止或暂停应用程序，然后启动回溯操作，这可以确保没有新的读取或写入请求。在回溯操作期间，Aurora 暂停数据库、克隆所有打开的连接并丢弃任何未提交的读取和写入。然后等待回溯操作完成。
- 您无法在不支持回溯的 AWS 区域中还原启用回溯的集群的跨区域快照。
- 如果您对启用了回溯的集群执行从 Aurora MySQL 版本 2 到版本 3 的就地升级，则无法回溯到升级发生之前的某个时间点。

区域和版本可用性

回溯不适用于 Aurora PostgreSQL。

以下是 Aurora MySQL 的回溯功能支持的引擎和区域可用性。

区域	Aurora MySQL 版本 3	Aurora MySQL 版本 2
美国东部（弗吉尼亚州北部）	所有版本	所有版本
美国东部（俄亥俄州）	所有版本	所有版本
美国西部（加利福尼亚北部）	所有版本	所有版本
美国西部（俄勒冈州）	所有版本	所有版本
Africa (Cape Town)	–	–
Asia Pacific (Hong Kong)	–	–

区域	Aurora MySQL 版本 3	Aurora MySQL 版本 2
亚太地区（雅加达）	–	–
亚太地区（马来西亚）	–	–
亚太地区（墨尔本）	–	–
亚太地区（孟买）	所有版本	所有版本
亚太地区（大阪）	所有版本	版本 2.07.3 及更高版本
亚太地区（首尔）	所有版本	所有版本
亚太地区（新加坡）	所有版本	所有版本
亚太地区（悉尼）	所有版本	所有版本
Asia Pacific (Tokyo)	所有版本	所有版本
加拿大（中部）	所有版本	所有版本
加拿大西部（卡尔加里）	–	–
中国（北京）	–	–
中国（宁夏）	–	–
欧洲地区（法兰克福）	所有版本	所有版本

区域	Aurora MySQL 版本 3	Aurora MySQL 版本 2
欧洲地区（爱尔兰）	所有版本	所有版本
欧洲地区（伦敦）	所有版本	所有版本
欧洲地区（米兰）	–	–
欧洲地区（巴黎）	所有版本	所有版本
欧洲（西班牙）	–	–
欧洲地区（斯德哥尔摩）	–	–
欧洲（苏黎世）	–	–
以色列（特拉维夫）	–	–
中东（巴林）	–	–
中东（阿联酋）	–	–
南美洲（圣保罗）	–	–
AWS GovCloud（美国东部）	–	–
AWS GovCloud（美国西部）	–	–

启用回溯的集群的升级注意事项

您可以将支持回溯的数据库集群从 Aurora MySQL 版本 2 升级到版本 3，因为回溯支持 Aurora MySQL 版本 3 的所有次要版本。

配置 Aurora MySQL 数据库集群回溯

要使用回溯功能，您必须启用回溯并指定目标回溯时段。否则将禁用回溯。

对于目标回溯时段，请指定您希望能够使用回溯功能将数据库倒回的时间长度。Aurora 尝试保留足够的更改记录以支持该时段。

控制台

在创建新的数据库集群时，您可以使用控制台配置回溯。您还可以修改数据库集群，以更改启用回溯的集群的回溯窗口。如果您通过将回溯窗口设置为 0 来完全关闭集群的回溯，则无法为该集群再次启用回溯。

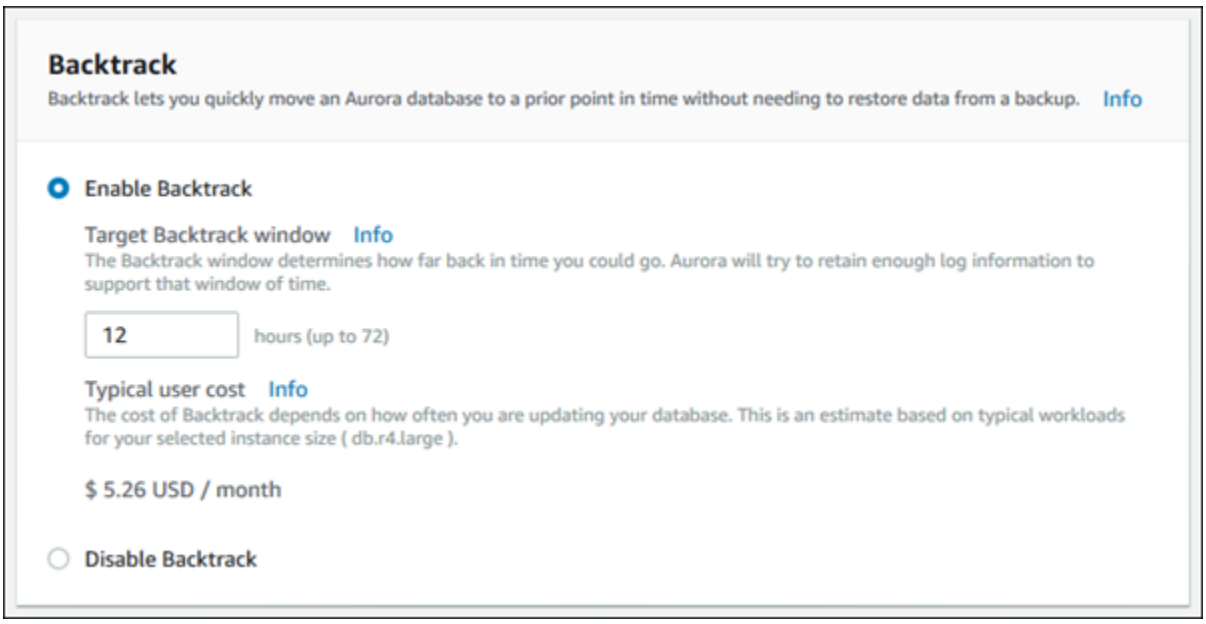
主题

- 在创建数据库集群时使用控制台配置回溯
- 在修改数据库集群时使用控制台配置回溯

在创建数据库集群时使用控制台配置回溯

在创建新的 Aurora MySQL 数据库集群时，您可以选择启用回溯，并在回溯部分中指定大于零的目标回溯时段值。

要创建数据库集群，请按创建 Amazon Aurora 数据库集群中的说明操作。下图显示了回溯部分。



创建新数据库集群时，Aurora 没有数据库集群的工作负载数据。因此，无法为新数据库集群具体估算成本。控制台会改为基于典型工作负载，针对指定的目标回溯时段提供典型用户成本。典型成本用于为回溯功能的成本提供一般参考。

> **Important**
>
> 您的实际成本可能与典型成本不一样，因为实际成本基于您的数据库集群的工作负载。

在修改数据库集群时使用控制台配置回溯

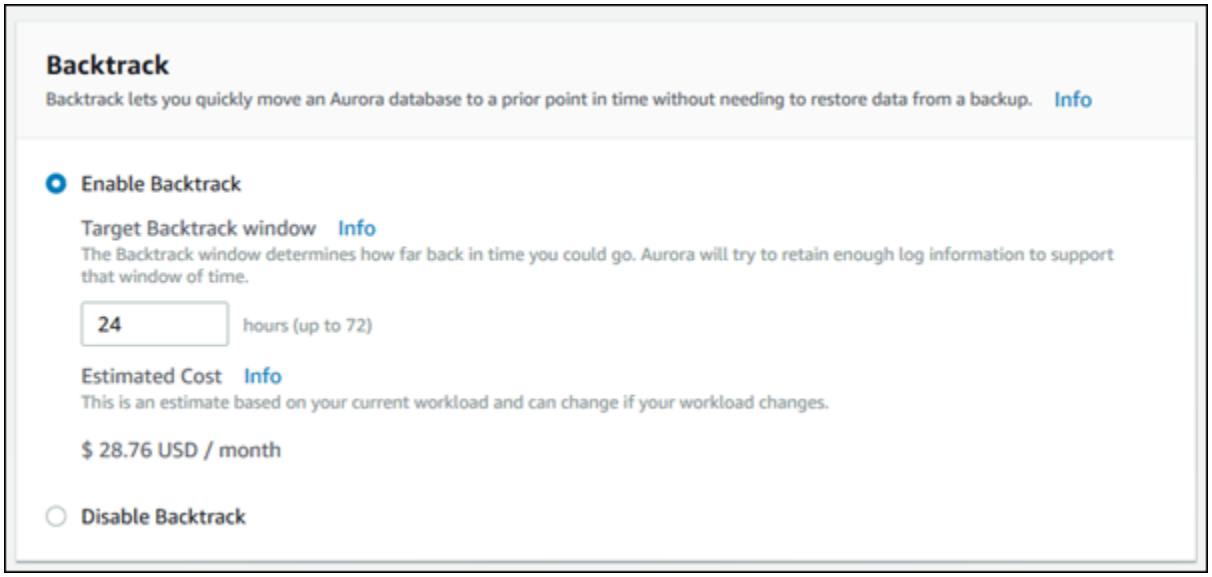
您可以使用控制台修改数据库集群的回溯。

> **Note**
>
> 目前，您只能为启用了回溯功能的数据库集群修改回溯。对于在禁用回溯功能的情况下创建的数据库集群或者如果为数据库集群禁用了回溯功能，则不会显示回溯部分。

使用控制台修改数据库集群的回溯

1. 登录AWS Management Console并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 选择数据库。

3. 选择要修改的集群，然后选择修改。
4. 对于目标回溯时段，修改您希望可以回溯的时间长度。限制为 72 小时。



控制台根据数据库集群过去的工作负载，显示您所指定的时间长度的预计成本：

- 如果在数据库集群上禁用了回溯，则估计的成本基于 Amazon CloudWatch 中的数据库集群的 `VolumeWriteIOPS` 指标。
- 如果以前在数据库集群上启用了回溯，则估计的成本基于 Amazon CloudWatch 中的数据库集群的 `BacktrackChangeRecordsCreationRate` 指标。

5. 选择 Continue (继续)。
6. 对于修改计划，请选择下列选项之一：
   - 在下一个计划的维护时段内应用 – 等到下一个维护时段以应用目标回溯时段修改。
   - 立即应用 – 尽快应用目标回溯时段修改。
7. 选择修改集群。

AWS CLI

在使用 create-db-cluster AWS CLI 命令创建新的 Aurora MySQL 数据库集群时，如果指定的 `--backtrack-window` 值大于零，则会配置回溯。`--backtrack-window` 值指定目标回溯时段。有关更多信息，请参阅“创建 Amazon Aurora 数据库集群”。

您还可以使用以下 `--backtrack-window` CLI 命令指定 AWS 值：

- modify-db-cluster

- restore-db-cluster-from-s3
- restore-db-cluster-from-snapshot
- restore-db-cluster-to-point-in-time

以下过程介绍了如何使用 AWS CLI 修改数据库集群的目标回溯时段。

使用 AWS CLI 修改数据库集群的目标回溯时段

- 调用 modify-db-cluster AWS CLI 命令并提供以下值：
  - `--db-cluster-identifier` – 数据库集群的名称。
  - `--backtrack-window` – 您希望能够将数据库集群回溯的最大秒数。

  以下示例将目标回溯时段 `sample-cluster` 设置为一天 (86,400 秒)。

  对于 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster \
    --db-cluster-identifier sample-cluster \
    --backtrack-window 86400
```

  对于 Windows：

```
aws rds modify-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --backtrack-window 86400
```

> **Note**
>
> 目前，您只能为在启用回溯功能的情况下创建的数据库集群启用回溯。

RDS API

在使用 CreateDBCluster Amazon RDS API 操作创建新的 Aurora MySQL 数据库集群时，如果指定的 `BacktrackWindow` 值大于零，则会配置回溯。`BacktrackWindow` 值针对在 `DBClusterIdentifier` 值中指定的数据库集群，指定目标回溯时段。有关更多信息，请参阅“创建 Amazon Aurora 数据库集群”。

您还可以使用以下 API 操作指定 `BacktrackWindow` 值：

- ModifyDBCluster
- RestoreDBClusterFromS3
- RestoreDBClusterFromSnapshot
- RestoreDBClusterToPointInTime

> **Note**
>
> 目前，您只能为在启用回溯功能的情况下创建的数据库集群启用回溯。

对 Aurora MySQL 数据库集群执行回溯

您可以将数据库集群回溯到指定的回溯时间戳。如果回溯时间戳不早于最早的可回溯时间，并且也不在未来，则数据库集群会回溯到该时间戳。

否则，通常将出错。此外，如果您尝试回溯启用了二进制日志记录的数据库集群，除非您选择了强制执行回溯，否则通常会出错。执行强制回溯可能会干扰使用二进制日志记录的其他操作。

> **Important**
>
> 回溯不会为所执行的更改生成二进制日志条目。如果您为数据库集群启用了二进制日志记录，则回溯可能会与您的二进制日志实施不兼容。

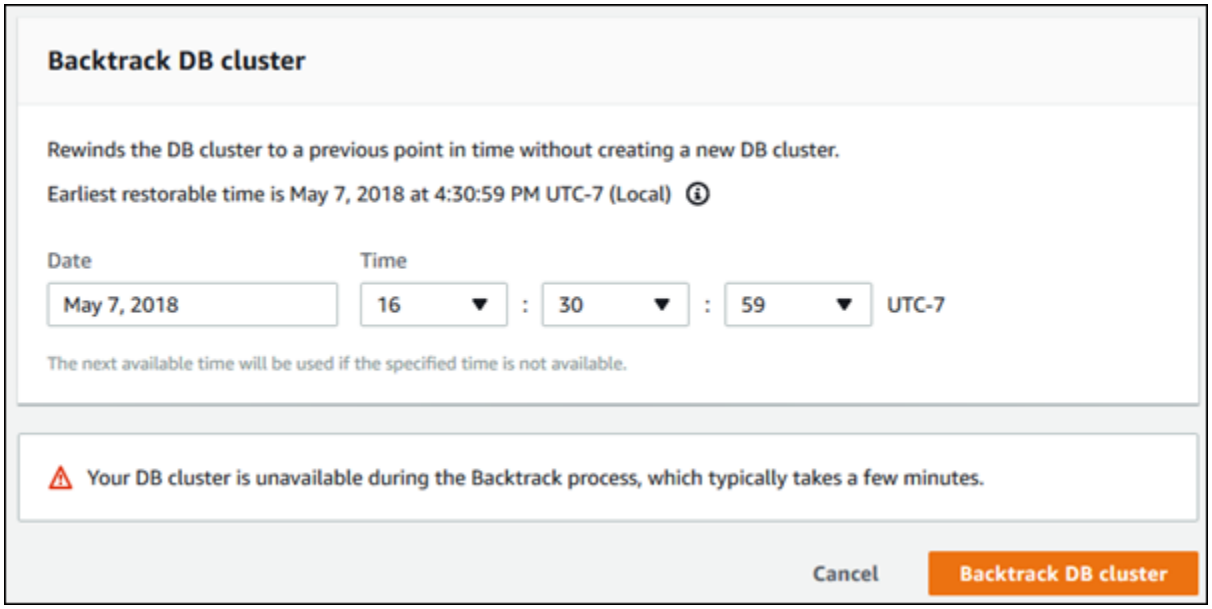
> **Note**
>
> 对于数据库克隆，在创建克隆后，您无法将数据库集群回溯到创建克隆之前的日期和时间。有关数据库克隆的更多信息，请参阅克隆 Amazon Aurora 数据库集群卷。

控制台

以下过程介绍了如何使用控制台为数据库集群的执行回溯操作。

使用控制台执行回溯操作

1. 登录AWS Management Console并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择实例。
3. 选择要回溯的数据库集群的主实例。
4. 对于 Actions (操作)，选择 Backtrack DB cluster (回溯数据库集群)。
5. 在 Backtrack DB cluster (回溯数据库集群) 页上，输入要将数据库集群回溯到的回溯时间戳。



6. 选择回溯数据库集群。

AWS CLI

以下过程介绍了如何使用 AWS CLI 回溯数据库集群。

使用 AWS CLI 回溯数据库集群

- 调用 backtrack-db-cluster AWS CLI 命令并提供以下值：
  - `--db-cluster-identifier` – 数据库集群的名称。
  - `--backtrack-to` – 将数据库集群回溯到的回溯时间戳，这是使用 ISO 8601 格式指定的。

以下示例将数据库集群 `sample-cluster` 回溯到 2018 年 3 月 19 日上午 10 点。

对于 Linux、macOS 或 Unix：

```
aws rds backtrack-db-cluster \
    --db-cluster-identifier sample-cluster \
    --backtrack-to 2018-03-19T10:00:00+00:00
```

对于 Windows：

```
aws rds backtrack-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --backtrack-to 2018-03-19T10:00:00+00:00
```

RDS API

要使用 Amazon RDS API 回溯数据库集群，请使用 BacktrackDBCluster 操作。该操作将在 `DBClusterIdentifier` 值中指定的数据库集群回溯到指定的时间。

监控 Aurora MySQL 数据库集群回溯

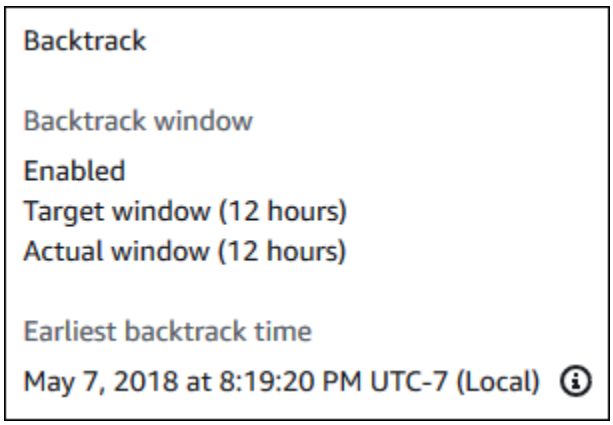
您可以查看回溯信息和监控数据库集群的回溯指标。

控制台

使用控制台查看回溯信息和监控回溯指标

1. 登录AWS Management Console并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 选择数据库。
3. 选择数据库集群名称以显示其相关信息。

   回溯信息位于回溯部分中。



启用了回溯时，以下信息可用：

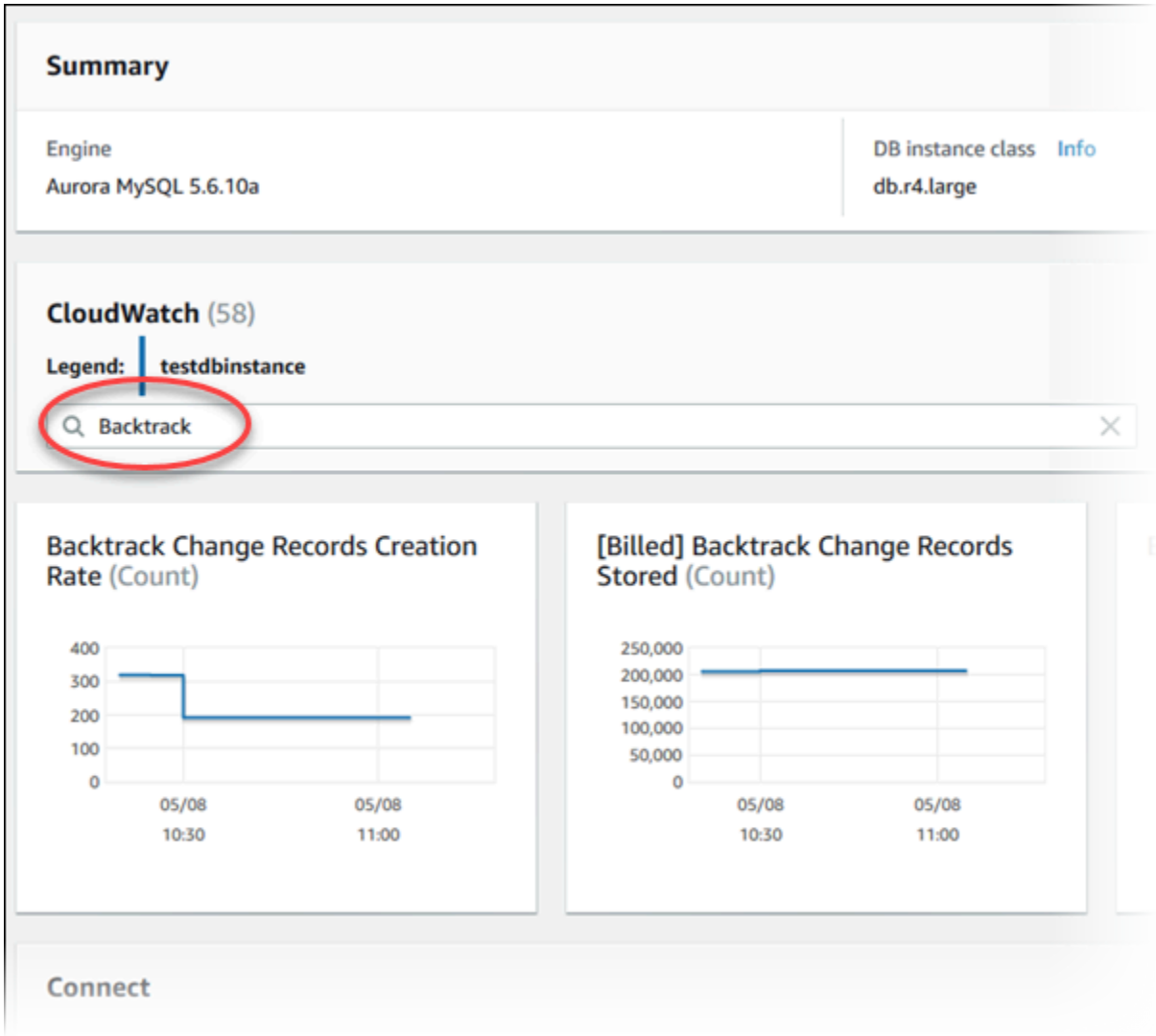
- 目标时段 – 当前为目标回溯时段指定的时间长度。目标回溯时段是在有足够的存储时您可以回溯的最长时间。
- 实际时段 – 您可以实际回溯的时间长度，这可能会小于目标回溯时段。实际回溯时段基于您的工作负载以及可用于保留回溯更改记录的存储。
- 最早的回溯时间 – 可以将数据库集群回溯到的最早时间。您无法将数据库集群回溯到显示的时间之前的时间。

4. 执行以下操作可查看数据库集群的回溯指标：

   a. 在导航窗格中，选择实例。

   b. 选择数据库集群的主实例以显示其详细信息。

   c. 在 CloudWatch 部分中，在 CloudWatch 框中键入**`Backtrack`**以仅显示回溯指标。



显示以下指标：

- 回溯更改记录创建速率（计数）– 该指标显示在 5 分钟的时间内为数据库集群创建的回溯更改记录数。您可以使用该指标估算目标回溯时段的时间成本。
- [已计费] 存储的回溯更改记录（计数）– 该指标显示数据库集群使用的实际回溯更改记录数。
- 实际回溯时段（分钟）– 该指标显示在目标回溯时段和实际回溯时段之间是否存在差异。例如，如果您的目标回溯时段为 2 小时 (120 分钟)，并且该指标显示实际回溯时段为 100 分钟，则实际回溯时段小于目标回溯时段。
- 回溯时段提醒（计数）– 该指标显示在给定时间段内实际回溯时段小于目标回溯时段的频率。

> **Note**
>
> 以下指标可能滞后于当前时间：
>
> - 回溯更改记录创建速率 (计数)
> - [已计费] 存储的回溯更改记录 (计数)

AWS CLI

以下过程介绍了如何使用 AWS CLI 查看数据库集群的回溯信息。

使用 AWS CLI 查看数据库集群的回溯信息

- 调用 describe-db-clusters AWS CLI 命令并提供以下值：
  - `--db-cluster-identifier` – 数据库集群的名称。

  以下示例列出 `sample-cluster` 的回溯信息。

  对于 Linux、macOS 或 Unix：

```
aws rds describe-db-clusters \
    --db-cluster-identifier sample-cluster
```

  对于 Windows：

```
aws rds describe-db-clusters ^
    --db-cluster-identifier sample-cluster
```

RDS API

要使用 Amazon RDS API 查看数据库集群的回溯信息，请使用 DescribeDBClusters 操作。该操作返回在 `DBClusterIdentifier` 值中指定的数据库集群的回溯信息。

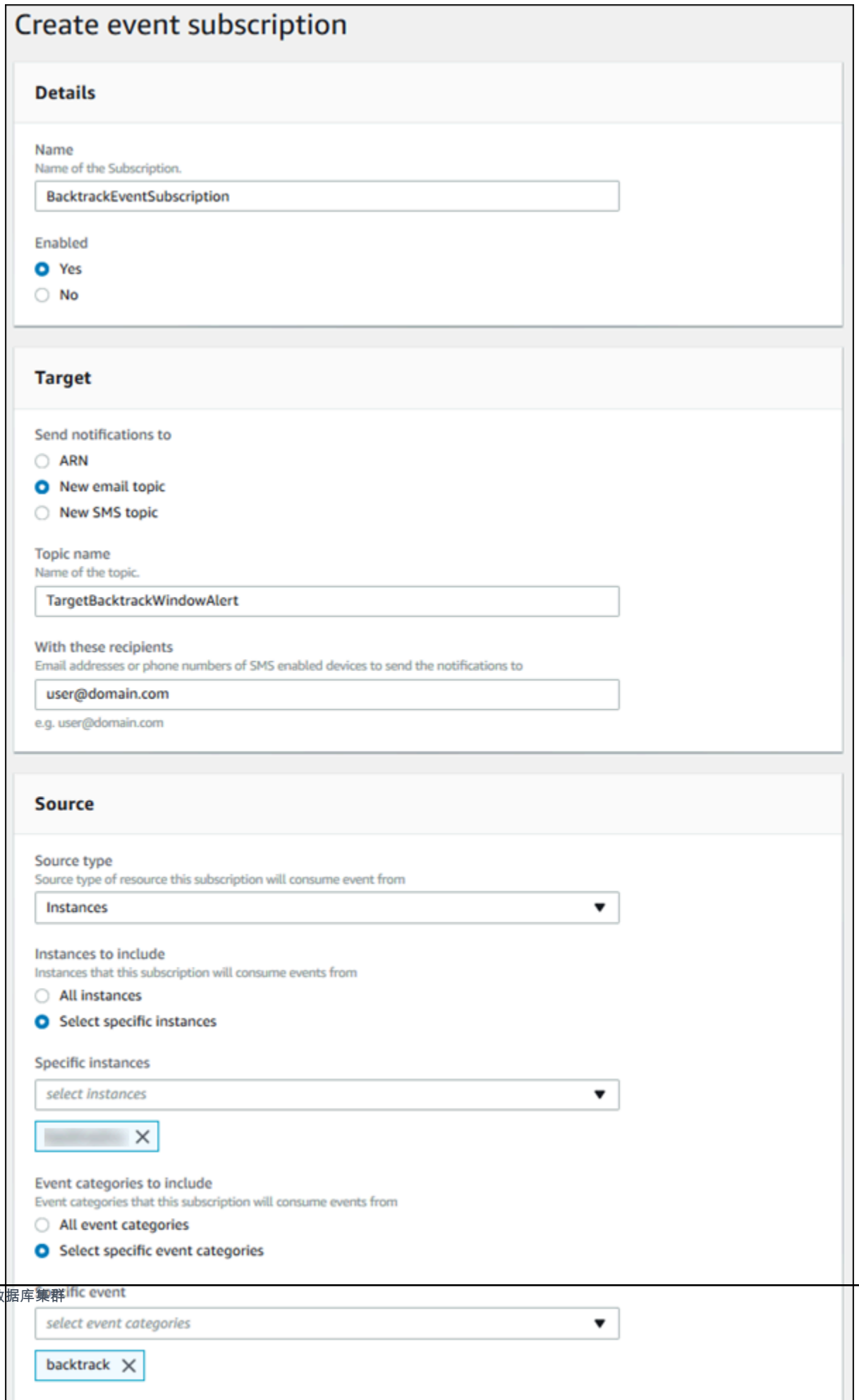
使用控制台订阅回溯事件

以下过程介绍了如何使用控制台运行订阅回溯事件。在您的实际回溯时段小于目标回溯时段时，该事件向您发送电子邮件或文本通知。

使用控制台查看回溯信息

1. 登录AWS Management Console并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 选择事件订阅。
3. 选择创建事件订阅。
4. 在名称框中，键入事件订阅的名称，然后确保为已启用选择了是。
5. 在目标部分中，选择新电子邮件主题。
6. 对于主题名称，键入主题的名称，对于使用以下收件人，键入接收通知的电子邮件地址或电话号码。
7. 在源部分中，为源类型选择实例。
8. 对于要包含的实例，选择选择特定实例，然后选择您的数据库实例。
9. 对于要包含的事件类别，选择选择特定事件类别，然后选择回溯。

   您的页面应类似于以下页面。


Create event subscription
Details
Name
Name of the Subscription.
BacktrackEventSubscription
Enabled
Yes
No
Target
Send notifications to
ARN
New email topic
New SMS topic
Topic name
Name of the topic.
TargetBacktrackWindowAlert
With these recipients
Email addresses or phone numbers of SMS enabled devices to send the notifications to
user@domain.com
e.g. user@domain.com
Source
Source type
Source type of resource this subscription will consume event from
Instances
Instances to include
Instances that this subscription will consume events from
All instances
Select specific instances
Specific instances
select instances
Event categories to include
Event categories that this subscription will consume events from
All event categories
Select specific event categories
select event categories
backtrack

10. 选择创建。

检索现有回溯

您可以检索有关数据库集群现有回溯的信息。此信息包含回溯的唯一标识符、可以来回回溯的日期和时间、请求回溯的日期和时间以及回溯的当前状态。

> **Note**
>
> 目前不能使用控制台检索现有回溯。

AWS CLI

以下过程介绍了如何使用 AWS CLI 检索数据库集群的现有回溯。

使用 AWS CLI 检索现有回溯

- 调用 describe-db-cluster-backtracks AWS CLI 命令并提供以下值：
  - `--db-cluster-identifier` – 数据库集群的名称。

  以下示例检索 `sample-cluster` 的现有回溯。

  对于 Linux、macOS 或 Unix：

```
aws rds describe-db-cluster-backtracks \
    --db-cluster-identifier sample-cluster
```

  对于 Windows：

```
aws rds describe-db-cluster-backtracks ^
    --db-cluster-identifier sample-cluster
```

RDS API

要使用 Amazon RDS API 检索有关数据库集群回溯的信息，请使用 DescribeDBClusterBacktracks 操作。该操作返回在 `DBClusterIdentifier` 值中指定的数据库集群的回溯相关信息。

对 Aurora MySQL 数据库集群禁用回溯

您可以禁用数据库集群的回溯功能。

控制台

您可以使用控制台禁用数据库集群的回溯。完全关闭集群的回溯后，则无法为该集群再次启用回溯。

使用控制台为数据库集群禁用回溯功能

1. 登录AWS Management Console并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 选择数据库。
3. 选择要修改的集群，然后选择修改。
4. 在回溯部分中，选择禁用回溯。
5. 选择 Continue (继续)。
6. 对于修改计划，请选择下列选项之一：
   - 在下一个计划的维护时段内应用 – 等到下一个维护时段以应用修改。
   - 立即应用 – 尽快应用修改。
7. 选择 Modify Cluster。

AWS CLI

您可以使用 AWS CLI 将目标回溯时段设置为 `0`（零），以便为数据库集群禁用回溯功能。完全关闭集群的回溯后，则无法为该集群再次启用回溯。

使用 AWS CLI 修改数据库集群的目标回溯时段

- 调用 modify-db-cluster AWS CLI 命令并提供以下值：
  - `--db-cluster-identifier` – 数据库集群的名称。
  - `--backtrack-window` – 指定 `0` 以关闭回溯。

以下示例通过将 sample-cluster 设置为 --backtrack-window，禁用 0 的回溯功能。

对于 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster \
    --db-cluster-identifier sample-cluster \
    --backtrack-window 0
```

对于 Windows：

```
aws rds modify-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --backtrack-window 0
```

RDS API

要使用 Amazon RDS API 为数据库集群禁用回溯功能，请使用 ModifyDBCluster 操作。将 BacktrackWindow 值设置为 0（零），然后在 DBClusterIdentifier 值中指定数据库集群。完全关闭集群的回溯后，则无法为该集群再次启用回溯。

使用错误注入查询测试 Amazon Aurora MySQL

您可以使用错误注入查询来测试 Aurora MySQL 数据库集群的容错能力。错误注入查询作为 SQL 命令发布到 Amazon Aurora 实例。它们允许您计划以下事件之一的模拟发生：

- 写入器或读取器数据库实例崩溃
- Aurora 副本故障
- 磁盘故障
- 磁盘拥塞

当故障注入查询指定崩溃时，它会强制 Aurora MySQL 数据库实例崩溃。其他错误注入查询将导致模拟故障事件，但不会导致事件发生。提交错误注入查询时，还可指定故障事件模拟发生的时间长度。

可通过连接到 Aurora 副本的终端节点来将错误注入查询提交到 Aurora 副本实例之一。有关更多信息，请参阅 Amazon Aurora 端点连接。

运行故障注入查询需要所有主用户权限。有关更多信息，请参阅 主用户账户权限。

测试实例崩溃

可使用 ALTER SYSTEM CRASH 错误注入查询强制使 Amazon Aurora 实例发生崩溃。

对于该错误注入查询，将不会进行故障转移。如果要测试故障转移，您可以在 RDS 控制台中为数据库集群选择 Failover（故障转移）实例操作，或者使用 failover-db-cluster AWS CLI 命令或 FailoverDBCluster RDS API 操作。

语法

```
ALTER SYSTEM CRASH [ INSTANCE | DISPATCHER | NODE ];
```

Options

该错误注入查询采用下列崩溃类型之一：

- **INSTANCE** – 模拟 Amazon Aurora 实例的 MySQL 兼容数据库崩溃。
- **DISPATCHER** – 模拟 Aurora 数据库集群的写入器实例上的调度程序崩溃。调度程序 将更新写入到 Amazon Aurora 数据库集群的集群卷中。
- **NODE** – 模拟 Amazon Aurora 实例的 MySQL 兼容数据库和调度程序崩溃。对于该错误注入模拟，还将删除缓存。

默认崩溃类型为 INSTANCE。

测试 Aurora 副本故障

可使用错误注入查询 ALTER SYSTEM SIMULATE READ REPLICA FAILURE 来模拟 Aurora 副本的故障。

Aurora 副本故障将在指定的时间间隔内，阻止对数据库集群中的一个 Aurora 副本或所有 Aurora 副本的所有写入器请求。在该时间间隔过后，受影响的 Aurora 副本将自动与写入器实例同步。

语法

```
ALTER SYSTEM SIMULATE percentage_of_failure PERCENT READ REPLICA FAILURE
    [ TO ALL | TO "replica name" ]
    FOR INTERVAL quantity { YEAR | QUARTER | MONTH | WEEK | DAY | HOUR | MINUTE |
 SECOND };
```

Options

该错误注入查询采用以下参数：

- **percentage_of_failure** – 在故障事件期间阻止的请求百分比。该值可为 0 到 100 之间的双数。如果指定 0，则不阻止任何请求。如果指定 100，则阻止所有请求。
- 故障类型 – 要模拟的故障的类型。指定 TO ALL 以模拟数据库集群中所有 Aurora 副本的故障。指定 TO 及单个 Aurora 副本的名称，以模拟单个 Aurora 副本故障的情况。默认故障类型为 TO ALL。
- **quantity** – 要模拟 Aurora 副本故障的时间长度。该间隔为后跟一个时间单位的时间长度。模拟将在指定单位的时间长度内发生。例如，20 MINUTE 将促使模拟持续运行 20 分钟。

> **Note**
>
> 在指定 Aurora 副本故障事件的时间间隔时，请小心谨慎。如果指定的时间间隔太长，并且您的写入器实例在故障事件期间写入大量数据，则您的 Aurora 数据库集群可能假定您的 Aurora 副本已发生崩溃并将替换它。

测试磁盘故障

可使用 ALTER SYSTEM SIMULATE DISK FAILURE 错误注入查询模拟 Aurora 数据库集群的磁盘故障。

磁盘故障模拟期间，Aurora 数据库集群会随机将磁盘区段标记为故障。在模拟期内，对这些区段的请求将被阻止。

语法

```
ALTER SYSTEM SIMULATE percentage_of_failure PERCENT DISK FAILURE
    [ IN DISK index | NODE index ]
    FOR INTERVAL quantity { YEAR | QUARTER | MONTH | WEEK | DAY | HOUR | MINUTE |
 SECOND };
```

Options

该错误注入查询采用以下参数：

- **percentage_of_failure** – 在故障事件期间标记为故障的磁盘百分比。该值可为 0 到 100 之间的双数。如果指定 0，则不会将任何磁盘标记为故障。如果指定 100，则整个磁盘将标记为故障。

- **DISK index** – 要模拟故障事件的特定逻辑数据块。如果您超出了可用逻辑数据块的范围，将收到一条错误，告知您可指定的最大索引值。有关更多信息，请参阅“显示 Aurora MySQL 数据库集群的卷状态”。
- **NODE index** – 要模拟故障事件的特定存储节点。如果您超出了可用存储节点的范围，将收到一条错误，告知您可指定的最大索引值。有关更多信息，请参阅“显示 Aurora MySQL 数据库集群的卷状态”。
- **quantity** – 要模拟磁盘故障的时间长度。该间隔为后跟一个时间单位的时间长度。模拟将在指定单位的时间长度内发生。例如，20 MINUTE 将促使模拟持续运行 20 分钟。

测试磁盘拥塞

可使用 ALTER SYSTEM SIMULATE DISK CONGESTION 错误注入查询模拟 Aurora 数据库集群的磁盘故障。

磁盘拥塞模拟期间，Aurora 数据库集群会随机将磁盘区段标记为拥塞。在模拟持续时间内，对这些区段的请求将在指定的最小和最大延迟时间之间延迟。

语法

```
ALTER SYSTEM SIMULATE percentage_of_failure PERCENT DISK CONGESTION
    BETWEEN minimum AND maximum MILLISECONDS
    [ IN DISK index | NODE index ]
    FOR INTERVAL quantity { YEAR | QUARTER | MONTH | WEEK | DAY | HOUR | MINUTE |
 SECOND };
```

Options

该错误注入查询采用以下参数：

- **percentage_of_failure** – 在故障事件期间标记为拥塞的磁盘百分比。该值可为 0 到 100 之间的双数。如果指定 0，则不会将任何磁盘标记为拥塞。如果指定 100，则整个磁盘将标记为拥塞。
- **DISK index** 或 **NODE index** – 要模拟故障事件的特定磁盘或节点。如果您超出了磁盘或节点的索引范围，则将收到一条错误，告知您可指定的最大索引值。
- **minimum** 和 **maximum** – 拥塞延迟的最小和最大时间长度（以毫秒为单位）。标记为拥塞的磁盘区段将在模拟持续时间内的最小和最大时间范围内的任意时间长度内 (以毫秒为单位) 延迟。
- **quantity** – 磁盘拥塞的模拟时间长度。该间隔为后跟一个时间单位的时间长度。模拟将在指定时间单位的时间长度内发生。例如，20 MINUTE 将促使模拟持续运行 20 分钟。

使用快速 DDL 在 Amazon Aurora 中修改表

Amazon Aurora 包括就地（几乎是瞬时的）运行 ALTER TABLE 操作的优化。完成该操作无需复制表，对其他 DML 语句也没有重大影响。由于该操作不会在表复制中使用临时存储，因此，甚至可以对小型实例类中的大型表使用 DDL 语句。

Aurora MySQL 版本 3 与社群 MySQL 即时 DDL 功能兼容。Aurora MySQL 版本 2 使用称为快速 DDL 的不同实现。

主题

- 即时 DDL（Aurora MySQL 版本 3）
- 快速 DDL（Aurora MySQL 版本 2）

即时 DDL（Aurora MySQL 版本 3）

Aurora MySQL 版本 3 为提高某些 DDL 操作效率而执行的优化称为即时 DDL。

Aurora MySQL 版本 3 与社群 MySQL 8.0 的即时 DDL 兼容。您可以将子句 ALGORITHM=INSTANT 与 ALTER TABLE 语句结合使用来执行即时 DDL 操作。有关即时 DDL 的语法和用法详细信息，请参阅 MySQL 文档中的 ALTER TABLE 和在线 DDL 操作。

以下示例演示了即时 DDL 功能。ALTER TABLE 语句可以添加列以及更改默认列值。这些示例包括常规列和虚拟列，以及常规表和分区表。在每个步骤中，都可以通过发布 SHOW CREATE TABLE 和 DESCRIBE 语句查看结果。

```
mysql> CREATE TABLE t1 (a INT, b INT, KEY(b)) PARTITION BY KEY(b) PARTITIONS 6;
Query OK, 0 rows affected (0.02 sec)

mysql> ALTER TABLE t1 RENAME TO t2, ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.01 sec)

mysql> ALTER TABLE t2 ALTER COLUMN b SET DEFAULT 100, ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.00 sec)

mysql> ALTER TABLE t2 ALTER COLUMN b DROP DEFAULT, ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.01 sec)

mysql> ALTER TABLE t2 ADD COLUMN c ENUM('a', 'b', 'c'), ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> ALTER TABLE t2 MODIFY COLUMN c ENUM('a', 'b', 'c', 'd', 'e'), ALGORITHM =
 INSTANT;
Query OK, 0 rows affected (0.01 sec)

mysql> ALTER TABLE t2 ADD COLUMN (d INT GENERATED ALWAYS AS (a + 1) VIRTUAL), ALGORITHM
 = INSTANT;
Query OK, 0 rows affected (0.02 sec)

mysql> ALTER TABLE t2 ALTER COLUMN a SET DEFAULT 20,
    ->   ALTER COLUMN b SET DEFAULT 200, ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE TABLE t3 (a INT, b INT) PARTITION BY LIST(a)(
    ->   PARTITION mypart1 VALUES IN (1,3,5),
    ->   PARTITION MyPart2 VALUES IN (2,4,6)
    -> );
Query OK, 0 rows affected (0.03 sec)

mysql> ALTER TABLE t3 ALTER COLUMN a SET DEFAULT 20, ALTER COLUMN b SET DEFAULT 200,
 ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE TABLE t4 (a INT, b INT) PARTITION BY RANGE(a)
    ->   (PARTITION p0 VALUES LESS THAN(100), PARTITION p1 VALUES LESS THAN(1000),
    ->   PARTITION p2 VALUES LESS THAN MAXVALUE);
Query OK, 0 rows affected (0.05 sec)

mysql> ALTER TABLE t4 ALTER COLUMN a SET DEFAULT 20,
    ->   ALTER COLUMN b SET DEFAULT 200, ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.01 sec)

/* Sub-partitioning example */
mysql> CREATE TABLE ts (id INT, purchased DATE, a INT, b INT)
    ->   PARTITION BY RANGE( YEAR(purchased) )
    ->     SUBPARTITION BY HASH( TO_DAYS(purchased) )
    ->     SUBPARTITIONS 2 (
    ->       PARTITION p0 VALUES LESS THAN (1990),
    ->       PARTITION p1 VALUES LESS THAN (2000),
    ->       PARTITION p2 VALUES LESS THAN MAXVALUE
    ->    );
Query OK, 0 rows affected (0.10 sec)

mysql> ALTER TABLE ts ALTER COLUMN a SET DEFAULT 20,
    ->   ALTER COLUMN b SET DEFAULT 200, ALGORITHM = INSTANT;
```

```
Query OK, 0 rows affected (0.01 sec)
```

快速 DDL（Aurora MySQL 版本 2）

Aurora MySQL 中的快速 DDL 是一种优化方法，旨在通过减少停机时间和资源使用来提高某些架构更改（例如添加或删除列）的性能。与传统的 DDL 方法相比，此方法可以更高效地完成这些操作。

> **Important**
>
> 目前，您必须启用 Aurora 实验室模式才能使用快速 DDL。有关启用实验室模式的信息，请参阅 Amazon Aurora MySQL 实验室模式。
> 快速 DDL 优化最初在 Aurora MySQL 版本 2 的实验室模式中引入，用来提高某些 DDL 操作的效率。在 Aurora MySQL 版本 3 中，实验室模式停用，快速 DDL 已被 MySQL 8.0 Instant DDL 功能所取代。

在 MySQL 中，很多数据定义语言 (DDL) 操作会对性能产生明显影响。

例如，假设您使用 `ALTER TABLE` 操作在表中添加一列。根据指定的算法，该操作可涉及以下步骤：

- 创建表的完整副本
- 创建临时表，以处理并发数据操控语言 (DML) 操作
- 重建此表的所有索引
- 应用并发 DML 更改时应用表锁定
- 减慢并发 DML 吞吐量

在具有大型表或高事务量的环境中，这种性能影响可能尤其具有挑战性。快速 DDL 优化了架构更改，使得操作速度更快，耗费的资源更少，从而缓解这些挑战。

快速 DDL 限制

目前，快速 DDL 具有以下限制：

- 快速 DDL 仅支持将没有默认值且可为空的列添加到现有表的最后。
- 快速 DDL 无法作用于分区表。
- 快速 DDL 无法作用于使用 REDUNDANT 行格式的 InnoDB 表。
- 快速 DDL 无法作用于具有全文搜索索引的表。

- 如果 DDL 操作的最大可能的记录大小太大，则不会使用快速 DDL。如果大于页面大小的一半，则说明记录大小太大。记录的最大大小是将所有列的最大大小相加得出的。对于大小可变的列，按照 InnoDB 标准，不会在计算中包含外部字节。

快速 DDL 语法

```
ALTER TABLE tbl_name ADD COLUMN col_name column_definition
```

该语句具有以下选项：

- **tbl_name** — 要修改的表的名称。
- **col_name** — 要添加的列的名称。
- **col_definition** — 要添加的列的定义。

> **Note**
>
> 您必须指定不带默认值且可为空的列定义，否则，无法使用快速 DDL。

快速 DDL 示例

以下示例演示了快速 DDL 操作带来的加速。第一个 SQL 示例在不使用快速 DDL 的情况下在大型表上运行 ALTER TABLE 语句。这项操作需要大量时间。CLI 示例说明了如何为集群启用快速 DDL。然后，另一个 SQL 示例在相同的表上运行相同的 ALTER TABLE 语句。启用快速 DDL 后，操作会非常快。

本示例使用 TPC-H 基准测试中的 ORDERS 表，其中包含 1.5 亿行。此集群有意使用相对较小的实例类来演示无法使用快速 DDL 时 ALTER TABLE 语句可能需要多长时间。该示例创建包含相同数据的原始表的克隆。检查 aurora_lab_mode 设置可确认集群没有使用快速 DDL，因为实验室模式未启用。然后，ALTER TABLE ADD COLUMN 语句需要大量时间才能在表格末尾添加新列。

```
mysql> create table orders_regular_ddl like orders;
Query OK, 0 rows affected (0.06 sec)

mysql> insert into orders_regular_ddl select * from orders;
Query OK, 150000000 rows affected (1 hour 1 min 25.46 sec)

mysql> select @@aurora_lab_mode;
+-------------------+
```

```
| @@aurora_lab_mode |
+-------------------+
|                 0 |
+-------------------+

mysql> ALTER TABLE orders_regular_ddl ADD COLUMN o_refunded boolean;
Query OK, 0 rows affected (40 min 31.41 sec)

mysql> ALTER TABLE orders_regular_ddl ADD COLUMN o_coverletter varchar(512);
Query OK, 0 rows affected (40 min 44.45 sec)
```

此示例与前一个示例一样准备了大型表。但是，您不能简单地在交互式 SQL 会话中启用实验室模式。必须在自定义参数组中启用该设置。这样做需要断开 mysql 会话并运行一些 AWS CLI 命令或使用 AWS Management Console。

```
mysql> create table orders_fast_ddl like orders;
Query OK, 0 rows affected (0.02 sec)

mysql> insert into orders_fast_ddl select * from orders;
Query OK, 150000000 rows affected (58 min 3.25 sec)

mysql> set aurora_lab_mode=1;
ERROR 1238 (HY000): Variable 'aurora_lab_mode' is a read only variable
```

为集群启用实验室模式需要使用参数组进行一些操作。此 AWS CLI 示例使用集群参数组，以确保集群中的所有数据库实例对实验室模式设置使用相同的值。

```
$ aws rds create-db-cluster-parameter-group \
  --db-parameter-group-family aurora5.7 \
    --db-cluster-parameter-group-name lab-mode-enabled-57 --description 'TBD'
$ aws rds describe-db-cluster-parameters \
  --db-cluster-parameter-group-name lab-mode-enabled-57 \
    --query '*[*].[ParameterName,ParameterValue]' \
      --output text | grep aurora_lab_mode
aurora_lab_mode 0
$ aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name lab-mode-enabled-57 \
    --parameters ParameterName=aurora_lab_mode,ParameterValue=1,ApplyMethod=pending-
reboot
{
    "DBClusterParameterGroupName": "lab-mode-enabled-57"
}
```

```
# Assign the custom parameter group to the cluster that's going to use Fast DDL.
$ aws rds modify-db-cluster --db-cluster-identifier tpch100g \
  --db-cluster-parameter-group-name lab-mode-enabled-57
{
  "DBClusterIdentifier": "tpch100g",
  "DBClusterParameterGroup": "lab-mode-enabled-57",
  "Engine": "aurora-mysql",
  "EngineVersion": "5.7.mysql_aurora.2.10.2",
  "Status": "available"
}

# Reboot the primary instance for the cluster tpch100g:
$ aws rds reboot-db-instance --db-instance-identifier instance-2020-12-22-5208
{
  "DBInstanceIdentifier": "instance-2020-12-22-5208",
  "DBInstanceStatus": "rebooting"
}

$ aws rds describe-db-clusters --db-cluster-identifier tpch100g \
  --query '*[].[DBClusterParameterGroup]' --output text
lab-mode-enabled-57

$ aws rds describe-db-cluster-parameters \
  --db-cluster-parameter-group-name lab-mode-enabled-57 \
    --query '*[*].{ParameterName:ParameterName,ParameterValue:ParameterValue}' \
      --output text | grep aurora_lab_mode
aurora_lab_mode 1
```

以下示例显示了参数组更改生效后的剩余步骤。它将测试 aurora_lab_mode 设置以确保集群可以使用快速 DDL。然后，它将运行 ALTER TABLE 语句以将列添加到另一个大型表的末尾。这一次，语句结束的非常快。

```
mysql> select @@aurora_lab_mode;
+-------------------+
| @@aurora_lab_mode |
+-------------------+
|                 1 |
+-------------------+

mysql> ALTER TABLE orders_fast_ddl ADD COLUMN o_refunded boolean;
Query OK, 0 rows affected (1.51 sec)
```

```
mysql> ALTER TABLE orders_fast_ddl ADD COLUMN o_coverletter varchar(512);
Query OK, 0 rows affected (0.40 sec)
```

显示 Aurora MySQL 数据库集群的卷状态

在 Amazon Aurora 中，数据库集群卷中包含一组逻辑块。每个逻辑块代表 10 GB 分配的存储空间。这些块称为保护组。

每个保护组中的数据在六个物理存储设备中进行复制，这些存储设备称为存储节点。这些存储节点分配给数据库集群所在 AWS 区域中的三个可用区 (AZ)。而每个存储节点又包含数据库集群卷的一个或多个逻辑数据块。有关保护组和存储节点的更多信息，请参阅 AWS 数据库博客上的 Aurora 存储引擎简介。

您可以模拟整个存储节点的故障，或存储节点中单个逻辑数据块的故障。要进行模拟，您可以使用 `ALTER SYSTEM SIMULATE DISK FAILURE` 错误注入语句。对于该语句，应指定特定逻辑数据块或存储节点的索引值。不过，如果您指定的索引值大于数据库集群卷使用的逻辑数据块或存储节点数，该语句将返回错误。有关错误注入查询的更多信息，请参阅 使用错误注入查询测试 Amazon Aurora MySQL。

您可以使用 `SHOW VOLUME STATUS` 语句以避免该错误。该语句返回两个服务器状态变量（`Disks` 和 `Nodes`）。这些变量分别表示数据库集群卷的逻辑数据块和存储节点总数。

语法

```
SHOW VOLUME STATUS
```

示例

以下示例说明了 SHOW VOLUME STATUS 的典型结果。

```
mysql> SHOW VOLUME STATUS;
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| Disks         | 96    |
| Nodes         | 74    |
+---------------+-------+
```

优化 Aurora MySQL

等待事件和线程状态是 Aurora MySQL 的一个重要的优化工具。如果您能查明会话为什么在等待资源以及它们在做什么，您就能更好地减少瓶颈。您可以使用本节中的信息来查找可能的原因和纠正措施。

Amazon DevOps Guru for RDS 可以主动识别您的 Aurora MySQL 数据库是否遇到问题，否则这些问题可能会在以后导致更大的问题。Amazon DevOps Guru for RDS 会在主动见解中发布有关纠正措施的解释和建议。本节包含对常见问题的见解。

> **Important**
>
> 本节中的等待事件和线程状态特定于 Aurora MySQL。使用本节中的信息仅能优化 Amazon Aurora，而不能优化 Amazon RDS for MySQL。
> 本节中的一些等待事件在这些数据库引擎的开源版本中没有类似内容。其他等待事件与开源引擎中的事件名称相同，但行为不同。例如，Amazon Aurora 存储的工作原理与开源存储不同，因此与存储相关的等待事件表明不同的资源状况。

主题

- Aurora MySQL 优化的基本概念
- 使用等待事件优化 Aurora MySQL
- 使用线程状态优化 Aurora MySQL
- 使用 Amazon DevOps Guru 主动见解优化 Aurora MySQL

Aurora MySQL 优化的基本概念

在优化 Aurora MySQL 数据库之前，请务必了解等待事件和线程状态及其发生的原因。同时，在使用 InnoDB 存储引擎时查看 Aurora MySQL 的基本内存和磁盘架构。有关有用的架构图，请参阅 MySQL 参考手册。

主题

- Aurora MySQL 等待事件
- Aurora MySQL 线程状态
- Aurora MySQL 内存
- Aurora MySQL 进程

Aurora MySQL 等待事件

等待事件表示会话正在等待的资源。例如，等待事件 `io/socket/sql/client_connection` 表示线程正在处理新连接。会话等待的典型资源包括以下内容：

- 例如，当会话试图修改缓冲区时，对缓冲区的单线程访问
- 当前被另一个会话锁定的行
- 已读取一个数据文件
- 已写入一个日志文件

例如，为了满足查询，会话可能会执行完整的表扫描。如果数据尚未在内存中，会话将等待磁盘输入/输出完成。当缓冲区读取到内存时，会话可能需要等待，因为其他会话正在访问相同的缓冲区。数据库使用预定义的等待事件记录等待。这些事件按类别进行分组。

等待事件本身并不显示性能问题。例如，如果请求的数据不在内存中，则必须从磁盘读取数据。如果一个会话锁定行以进行更新，则另一个会话将等待解锁该行，以便它可以更新该行。提交需要等待对日志文件的写入完成。等待是数据库正常运行不可或缺的组成部分。

大量等待事件通常显示性能问题。在这种情况下，您可以使用等待事件数据来确定会话将时间花费在哪里。例如，如果通常在几分钟内运行的报告现在运行了数小时，则可以确定对总等待时间贡献最大的等待事件。如果您能确定顶级等待事件的原因，您有时就可以进行更改来提高性能。例如，如果您的会话正在等待已被另一个会话锁定的行，则可以结束锁定会话。

Aurora MySQL 线程状态

常规线程状态指的是与常规查询处理相关联的 `State` 值。例如，线程状态 `sending data` 表示线程正在读取和筛选查询的行以确定正确的结果集。

您可以使用线程状态以类似于使用等待事件的方式来优化 Aurora MySQL。例如，频繁发生 `sending data` 通常表示查询没有使用索引。有关线程状态的更多信息，请参阅 MySQL 参考手册中的常规线程状态。

使用性能详情时，以下条件之一为真：

- 性能架构已开启 – Aurora MySQL 显示等待事件而不是线程状态。
- 性能架构未开启 – Aurora MySQL 显示线程状态。

我们建议您配置性能架构以实现自动管理。性能架构提供了更多洞察和更好的工具来调查潜在的性能问题。有关更多信息，请参阅 Aurora MySQL 上性能详情的性能架构概述。

Aurora MySQL 内存

在 Aurora MySQL 中，最重要的内存区域是缓冲池和日志缓冲区。

主题

- 缓冲池

缓冲池

缓冲池是 Aurora MySQL 在其中缓存表和索引数据的共享内存区域。查询可以直接从内存访问常用的数据，而无需从磁盘读取。

缓冲池的结构为页面链接列表。一个页面可以容纳多个行。Aurora MySQL 使用最近最少使用的 (LRU) 算法将页面从池中排除。

有关更多信息，请参阅《MySQL 参考手册》中的缓冲池。

Aurora MySQL 进程

Aurora MySQL 使用的流程模型与 Aurora PostgreSQL 截然不同。

主题

- MySQL 服务器 (mysqld)
- 线程
- 线程池

MySQL 服务器 (mysqld)

MySQL 服务器是一个名为 mysqld 的单个操作系统进程。MySQL 服务器不会产生额外的进程。因此，Aurora MySQL 数据库使用 mysqld 来执行其大部分工作。

当 MySQL 服务器启动时，它会侦听来自 MySQL 客户端的网络连接。当客户端连接到数据库时，mysqld 会打开一个线程。

线程

连接管理器线程将每个客户端连接与专用线程关联。该线程管理身份验证、运行语句并将结果返回到客户端。必要时，连接管理器会创建新线程。

线程缓存是一组可用的线程。连接结束时，如果缓存未满，MySQL 会将线程返回到线程缓存。`thread_cache_size` 系统变量决定了线程缓存的大小。

线程池

线程池由许多线程组组成。每个组管理一组客户端连接。当客户端连接到数据库时，线程池将以循环方式将连接分配给线程组。线程池将连接和线程分开。连接和运行从这些连接接收到的语句的线程之间没有固定的关系。

使用等待事件优化 Aurora MySQL

下表汇总了最常用于表示性能问题的 Aurora MySQL 等待事件。以下等待事件是 Aurora MySQL 等待事件 中的列表子集。

等待事件	描述
cpu	当线程在 CPU 中处于活动状态或正在等待 CPU 时，会发生此事件。
io/aurora_redo_log_flush	在会话向 Aurora 存储中写入持久数据时，将发生此事件。
io/aurora_respond_to_client	当线程等待将结果集返回给客户端时，会发生此事件。
io/redo_log_flush	在会话向 Aurora 存储中写入持久数据时，将发生此事件。
io/socket/sql/client_connection	当线程正处理新连接时，将发生此事件。
io/table/sql/handler	当工作被委派给存储引擎时，会发生此事件。
synch/cond/innodb/row_lock_wait	当一个会话锁定了一行以进行更新，而另一个会话尝试更新同一行时，会发生此事件。
synch/cond/innodb/row_lock_wait_cond	当一个会话锁定了一行以进行更新，而另一个会话尝试更新同一行时，会发生此事件。
synch/cond/sql/MDL_context::COND_wait_status	当有正等待表元数据锁定的线程时，会发生此事件。

等待事件	描述
synch/mutex/innodb/aurora_lock_thread_slot_futex	当一个会话锁定了一行以进行更新，而另一个会话尝试更新同一行时，会发生此事件。
synch/mutex/innodb/buf_pool_mutex	当线程在 InnoDB 缓冲池上获取锁定以访问内存中的页面时，将发生此事件。
synch/mutex/innodb/fil_system_mutex	当会话等待访问表空间内存缓存时，会发生此事件。
synch/mutex/innodb/trx_sys_mutex	当存在大量事务处理的大量数据库活动时，会发生此事件。
synch/sxlock/innodb/hash_table_locks	当必须从文件读取缓冲池中找不到的页面时，会发生此事件。

cpu

当线程在 CPU 中处于活动状态或正在等待 CPU 时，会发生 cpu 等待事件。

主题

- 支持的引擎版本
- 上下文
- 等待次数增加的可能原因
- 操作

支持的引擎版本

以下引擎版本支持此等待事件信息：

- Aurora MySQL 版本 2 和 3

上下文

对于每个 vCPU，连接可以在此 CPU 上运行。在某些情况下，准备运行的活动连接数高于 vCPU 的数量。这种不平衡导致连接等待 CPU 资源。如果活动连接数始终高于 vCPU 的数量，则您的实例会遇到 CPU 争用情况。争用会导致 cpu 等待事件发生。

> **Note**
>
> CPU 的性能详情指标为 DBLoadCPU。DBLoadCPU 的值可能与 CloudWatch 指标 CPUUtilization 的值不同。后一个指标是从 Hypervisor 中收集的，用于数据库实例。

性能详情操作系统指标提供有关 CPU 利用率的详细信息。例如，您可以显示以下指标：

- os.cpuUtilization.nice.avg
- os.cpuUtilization.total.avg
- os.cpuUtilization.wait.avg
- os.cpuUtilization.idle.avg

性能详情将数据库引擎的 CPU 使用情况报告为 os.cpuUtilization.nice.avg。

等待次数增加的可能原因

当此事件的发生率超过正常（可能表示性能问题）时，典型原因包括以下几点：

- 分析查询
- 高度并发的事务
- 长时间运行的事务
- 连接数量突然增加，称为登录风暴
- 上下文切换增加

操作

如果 cpu 等待事件主导着数据库活动，它不一定表示性能问题。只在性能下降时应对此事件。

根据 CPU 利用率提高的原因，考虑以下策略：

- 增加主机的 CPU 容量。这种方法通常只能提供临时的缓解。
- 确定潜在优化的主要查询。
- 如果适用，将一些只读工作负载重新导向到读取器节点。

主题

- 确定导致问题的会话或查询
- 分析和优化高 CPU 工作负载

确定导致问题的会话或查询

要查找会话和查询，请查看性能详情的主要 SQL 表格，以了解 CPU 负载最高的 SQL 语句。有关更多信息，请参阅 使用 Performance Insights 控制面板分析指标。

通常，一两条 SQL 语句会占用大部分的 CPU 周期。把精力集中在这些语句上。假设您的数据库实例有 2 个数据库负载为 3.1 个平均活动会话 (AAS) 的 vCPU，且它们全部处于 CPU 状态。在这种情况下，您的实例受 CPU 限制。请考虑以下策略：

- 升级到具有更多 vCPU 的更大的实例类。
- 优化查询以降低 CPU 负载。

在此示例中，主要 SQL 查询的数据库负载为 1.5 个 AAS，全部处于 CPU 状态。另一条 SQL 语句在 CPU 状态下的负载为 0.1。在此示例中，如果停止了负载最低的 SQL 语句，不会显著降低数据库负载。但是，如果您优化两个高负载查询，使其效率提高一倍，就可以消除 CPU 瓶颈。如果将 1.5 个 AAS 的 CPU 负载减少 50%，则每条语句的 AAS 将降至 0.75。现在花在 CPU 上的总数据库负载为 1.6 个 AAS。该值低于 2.0 的最大 vCPU 行。

有关使用性能详情进行故障排除的有用概览，请参阅博客文章利用性能详情分析 Amazon Aurora MySQL 工作负载。另请参阅 AWS Support 文章如何对 Amazon RDS for MySQL 实例上的的高 CPU 使用率进行故障排除和解决？。

分析和优化高 CPU 工作负载

在确定提高 CPU 使用率的查询之后，您可以优化它们或结束连接。以下示例说明如何结束连接。

```
CALL mysql.rds_kill(processID);
```

有关更多信息，请参阅 mysql.rds_kill。

如果结束会话，该操作可能会触发长时间回滚。

遵循优化查询的指南

要优化查询，请考虑以下指南：

- 运行 EXPLAIN 语句。

  此命令显示运行查询所涉及的各个步骤。有关更多信息，请参阅 MySQL 文档中的使用 EXPLAIN 优化查询。
- 运行 SHOW PROFILE 语句。

  使用此语句可查看配置文件详细信息，这些详细信息可以指示在当前会话期间运行的语句的资源使用情况。有关更多信息，请参阅 MySQL 文档中的 SHOW PROFILE 语句。
- 运行 ANALYZE TABLE 语句。

  使用此语句刷新 CPU 占用量较高的查询访问的表的索引统计信息。通过分析语句，您可以帮助优化程序选择合适的执行计划。有关更多信息，请参阅 MySQL 文档中的 ANALYZE TABLE 语句。

遵循提高 CPU 使用率的指南

要提高数据库实例中的 CPU 使用率，请遵循以下指南：

- 确保所有查询都使用正确的索引。
- 了解您是否可以使用 Aurora 并行查询。您可以使用此方法通过向下推送 WHERE 子句的函数处理、行筛选和列投影，减少头节点上的 CPU 使用率。
- 了解每秒 SQL 执行次数是否达到预期阈值。
- 了解索引维护或新索引创建是否占用生产工作负载所需的 CPU 周期。在活动高峰时间之外安排维护活动。
- 了解是否可以使用分区来帮助减少查询数据集。有关更多信息，请参阅博客文章如何计划和优化与 MySQL 兼容的 Amazon Aurora 以实现工作负载整合。

检查连接风暴

如果 DBLoadCPU 指标不是很高，但 CPUUtilization 指标很高，则 CPU 使用率高的原因在于数据库引擎之外。一个典型的例子是连接风暴。

检查以下条件是否为真：

- 性能详情 `CPUUtilization` 指标和 Amazon CloudWatch `DatabaseConnections` 指标都有所提高。
- CPU 中的线程数大于 vCPU 的数量。

如果上述条件为真，请考虑减少数据库连接的数量。例如，您可以使用 RDS 代理之类的连接池。要了解有效的连接管理和扩缩的最佳实践，请参阅白皮书 Amazon Aurora MySQL DBA 连接管理手册。

io/aurora_redo_log_flush

在会话向 Amazon Aurora 存储中写入持久数据时，将发生 `io/aurora_redo_log_flush` 事件。

主题

- 支持的引擎版本
- 上下文
- 等待次数增加的可能原因
- 操作

支持的引擎版本

以下引擎版本支持此等待事件信息：

- Aurora MySQL 版本 2

上下文

`io/aurora_redo_log_flush` 事件用于 Aurora MySQL 中的写入输入/输出 (I/O) 操作。

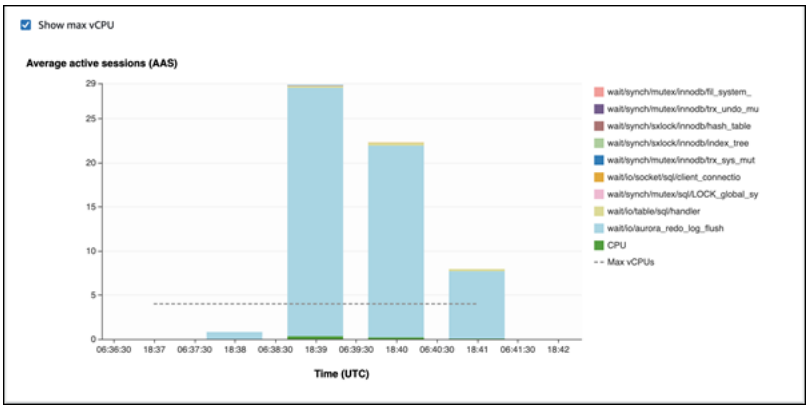
> **Note**
>
> 在 Aurora MySQL 版本 3 中，此等待事件名为 io/redo_log_flush。

等待次数增加的可能原因

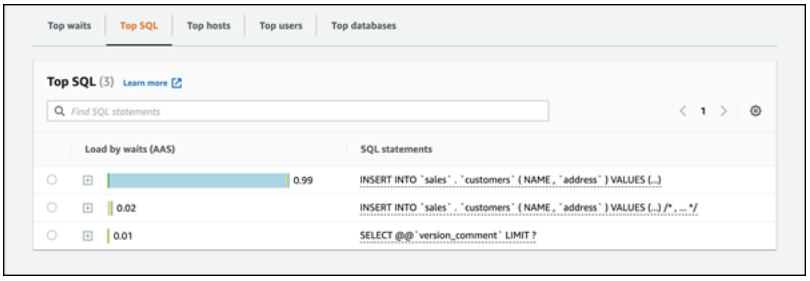
对于数据持久性，提交需要对稳定存储进行持久写入操作。如果数据库执行的提交太多，写入输入/输出操作会出现等待事件，即 `io/aurora_redo_log_flush` 等待事件。

在以下示例中，使用 db.r5.xlarge 数据库实例类将 50000 条记录插入到 Aurora MySQL 数据库集群中：

- 在第一个示例中，每个会话逐行插入 10000 条记录。预设情况下，如果数据操纵语言 (DML) 命令不在事务中，Aurora MySQL 将使用隐式提交。自动提交已开启。这意味着每行插入都有一个提交。性能详情显示，连接的大部分时间都花在等待 `io/aurora_redo_log_flush` 等待事件上。

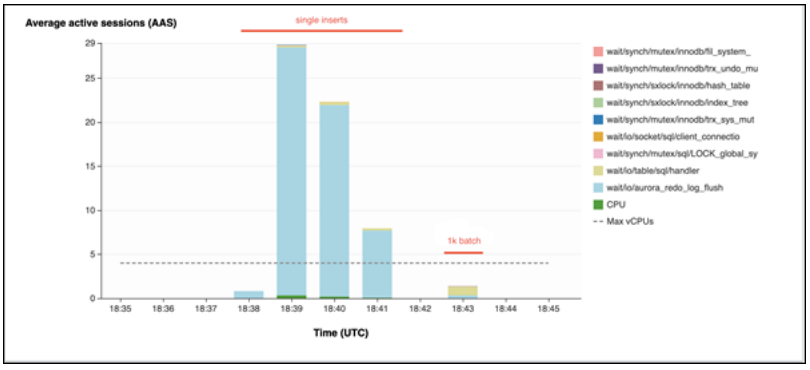


这是由于使用的简单插入语句造成的。



50000 条记录需要 3.5 分钟才能插入。

- 在第二个示例中，插入分 1000 个批次进行，也就是说每个连接执行 10 次提交，而不是 10000 次。性能详情显示，连接不会将大部分时间花在 `io/aurora_redo_log_flush` 等待事件上。



50000 条记录需要 4 秒钟才能插入。

操作

根据等待事件的原因，我们建议采取不同的操作。

主题

- 识别有问题的会话和查询
- 对写入操作进行分组
- 关闭自动提交
- 使用事务
- 使用批处理

识别有问题的会话和查询

如果数据库实例遇到瓶颈，您的首要任务是查找导致瓶颈的会话和查询。对于有用 AWS 数据库博客文章，请参阅利用性能详情分析 Amazon Aurora MySQL 工作负载。

识别导致瓶颈的会话和查询

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择性能详情。
3. 选择您的数据库实例。
4. 在 Database load（数据库负载）中，选择 Slice by wait（按等待切片）。
5. 在页面底部，选择 Top SQL（主要 SQL）。

   列表顶部的查询导致数据库负载最高。

对写入操作进行分组

以下示例会触发 `io/aurora_redo_log_flush` 等待事件。（自动提交已开启。）

```
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
....
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
```

```
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;
....
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;

DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
....
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
```

为了减少等待 io/aurora_redo_log_flush 等待事件所花费的时间，将写入操作按逻辑分组为单个提交，以减少对存储的持久调用。

关闭自动提交

在进行不在事务范围内的大型更改之前，请关闭自动提交，如以下示例所示。

```
SET SESSION AUTOCOMMIT=OFF;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
....
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
-- Other DML statements here
COMMIT;

SET SESSION AUTOCOMMIT=ON;
```

使用事务

您可使用事务，如以下示例所示。

```
BEGIN
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
....
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');

DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
```

```
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
....
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;

-- Other DML statements here
END
```

使用批处理

您还可以对批处理进行更改，如以下示例所示。但是，使用过大的批处理可能会导致性能问题，尤其是在只读副本中或执行时间点恢复 (PITR) 时。

```
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES
('xxxx','xxxxx'),('xxxx','xxxxx'),...,('xxxx','xxxxx'),('xxxx','xxxxx');

UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1 BETWEEN xx AND
 xxx;

DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1<xx;
```

io/aurora_respond_to_client

当线程等待将结果集返回给客户端时，会发生 `io/aurora_respond_to_client` 事件。

主题

- 支持的引擎版本
- 上下文
- 等待次数增加的可能原因
- 操作

支持的引擎版本

以下引擎版本支持此等待事件信息：

- Aurora MySQL 版本 2

上下文

事件 `io/aurora_respond_to_client` 表示线程正等待将结果集返回给客户端。

查询处理已完成，结果将返回应用程序客户端。但是，由于数据库集群上没有足够的网络带宽，线程正在等待返回结果集。

等待次数增加的可能原因

当 io/aurora_respond_to_client 事件的发生率超过正常（可能表示性能问题）时，典型原因包括以下几点：

数据库实例类不足以满足工作负载

数据库集群使用的数据库实例类没有高效处理工作负载所需的网络带宽。

大型结果集

返回的结果集的大小有所增加，因为查询返回的行数越多。结果集越大，消耗的网络带宽越多。

客户端负载增加

客户端上可能存在 CPU 压力、内存压力或网络饱和。客户端负载的增加会延迟从 Aurora MySQL 数据库集群接收数据的过程。

网络延迟增加

Aurora MySQL 数据库集群和客户端之间的网络延迟可能会增加。较高的网络延迟会增加客户端接收数据所需的时间。

操作

根据等待事件的原因，我们建议采取不同的操作。

主题

- 确定导致事件的会话和查询
- 扩展数据库实例类
- 检查工作负载是否有意外结果
- 使用读取器实例分配工作负载
- 使用 SQL_BUFFER_RESULT 修饰符

确定导致事件的会话和查询

您可以使用性能详情显示被 io/aurora_respond_to_client 等待事件阻止的查询。通常，具有中等到大量负载的数据库都会有等待事件。如果性能最佳，等待事件可能是可以接受的。如果性能不佳，

请检查数据库花费最多时间的位置。查看导致最高负载的等待事件，并了解是否可以优化数据库和应用程序以减少这些事件。

查找负责高负载的 SQL 查询

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择性能详情。
3. 选择一个数据库实例。将为该数据库实例显示性能详情控制面板。
4. 在 Database load（数据库负载）图表中，选择 Slice by wait（按等待切片）。
5. 在页面底部，选择 Top SQL（热门 SQL）。

   该图表列出了负责加载的 SQL 查询。排在列表前面的负载负有最大的责任。为了解决瓶颈，请关注这些语句。

有关使用性能详情进行故障排除的有用概览，请参阅 AWS 数据库博客文章使用性能详情分析 Amazon Aurora MySQL 工作负载。

扩展数据库实例类

检查与网络吞吐量相关的 Amazon CloudWatch 指标值的增加，例如 `NetworkReceiveThroughput` 和 `NetworkTransmitThroughput`。如果达到数据库实例类网络带宽，您可以通过修改数据库集群来扩展数据库集群使用的数据库实例类。网络带宽较大的数据库实例类可以更高效地将数据返回给客户端。

有关监控 Amazon CloudWatch 指标的信息，请参阅 在 Amazon RDS 控制台中查看指标。有关数据库实例类的信息，请参阅 Amazon Aurora 数据库实例类。有关修改数据库集群的信息，请参阅修改 Amazon Aurora 数据库集群。

检查工作负载是否有意外结果

检查数据库集群上的工作负载，并确保它不会产生意外的结果。例如，可能有查询返回的行数比预期的要多。在这种情况下，您可以使用性能详情计数器指标，例如 `Innodb_rows_read`。有关更多信息，请参阅 性能详情计数器指标。

使用读取器实例分配工作负载

您可以用 Aurora 副本分配只读工作负载。您可以通过添加更多 Aurora 副本来进行水平扩展。这样做可能会增加网络带宽的节流限制。有关更多信息，请参阅 Amazon Aurora 数据库集群。

使用 SQL_BUFFER_RESULT 修饰符

您可以添加 SQL_BUFFER_RESULT 修饰符到 SELECT 语句中，在将结果返回给客户端之前将结果强制放入临时表中。当 InnoDB 锁没有被释放时，此修饰符可以帮助解决性能问题，因为查询处于 io/aurora_respond_to_client 等待状态。有关更多信息，请参阅 MySQL 文档中的 SELECT 语句。

io/redo_log_flush

在会话向 Amazon Aurora 存储中写入持久数据时，将发生 io/redo_log_flush 事件。

主题

- 支持的引擎版本
- 上下文
- 等待次数增加的可能原因
- 操作

支持的引擎版本

以下引擎版本支持此等待事件信息：

- Aurora MySQL 版本 3

上下文

io/redo_log_flush 事件用于 Aurora MySQL 中的写入输入/输出 (I/O) 操作。

> **Note**
>
> 在 Aurora MySQL 版本 2 中，此等待事件名为 io/aurora_redo_log_flush。

等待次数增加的可能原因

对于数据持久性，提交需要对稳定存储进行持久写入操作。如果数据库执行的提交太多，写入输入/输出操作会出现等待事件，即 io/redo_log_flush 等待事件。

有关此等待事件的行为的示例，请参阅 io/aurora_redo_log_flush。

操作

根据等待事件的原因，我们建议采取不同的操作。

主题

- 识别有问题的会话和查询
- 对写入操作进行分组
- 关闭自动提交
- 使用事务
- 使用批处理

识别有问题的会话和查询

如果数据库实例遇到瓶颈，您的首要任务是查找导致瓶颈的会话和查询。对于有用 AWS 数据库博客文章，请参阅利用性能详情分析 Amazon Aurora MySQL 工作负载。

识别导致瓶颈的会话和查询

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择性能详情。
3. 选择您的数据库实例。
4. 在 Database load（数据库负载）中，选择 Slice by wait（按等待切片）。
5. 在页面底部，选择 Top SQL（主要 SQL）。

   列表顶部的查询导致数据库负载最高。

对写入操作进行分组

以下示例会触发 `io/redo_log_flush` 等待事件。（自动提交已开启。）

```
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
....
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
```

```
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;
....
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;

DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
....
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
```

为了减少等待 io/redo_log_flush 等待事件所花费的时间，将写入操作按逻辑分组为单个提交，以减少对存储的持久调用。

关闭自动提交

在进行不在事务范围内的大型更改之前，请关闭自动提交，如以下示例所示。

```
SET SESSION AUTOCOMMIT=OFF;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
....
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
-- Other DML statements here
COMMIT;

SET SESSION AUTOCOMMIT=ON;
```

使用事务

您可使用事务，如以下示例所示。

```
BEGIN
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
....
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
```

```
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
....
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;

-- Other DML statements here
END
```

使用批处理

您还可以对批处理进行更改，如以下示例所示。但是，使用过大的批处理可能会导致性能问题，尤其是在只读副本中或执行时间点恢复 (PITR) 时。

```
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES
('xxxx','xxxxx'),('xxxx','xxxxx'),...,('xxxx','xxxxx'),('xxxx','xxxxx');

UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1 BETWEEN xx AND
 xxx;

DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1<xx;
```

io/socket/sql/client_connection

当线程正处理新连接时，将发生 `io/socket/sql/client_connection` 事件。

主题

- 支持的引擎版本
- 上下文
- 等待次数增加的可能原因
- 操作

支持的引擎版本

以下引擎版本支持此等待事件信息：

- Aurora MySQL 版本 2 和 3

上下文

事件 io/socket/sql/client_connection 表示 mysqld 正忙于创建线程来处理传入的新客户端连接。在这种情况下，在连接等待分配线程的同时，服务新客户端连接请求的处理速度会减慢。有关更多信息，请参阅 MySQL 服务器 (mysqld)。

等待次数增加的可能原因

当此事件的发生率超过正常（可能表示性能问题）时，典型原因包括以下几点：

- 从应用程序到 Amazon RDS 实例的新用户连接突然增加。
- 由于网络、CPU 或内存受到限制，您的数据库实例无法处理新连接。

操作

如果 io/socket/sql/client_connection 主导着数据库活动，它不一定表示性能问题。在不处于空闲状态的数据库中，等待事件始终处于首位。只有在性能下降时才采取行动。根据等待事件的原因，我们建议采取不同的操作。

主题

- 识别有问题的会话和查询
- 遵循连接管理的最佳实践
- 如果资源受到限制，则纵向扩展实例
- 检查首要主机和主要用户
- 查询 performance_schema 表
- 检查查询的线程状态
- 审核您的请求和查询
- 池化您的数据库连接

识别有问题的会话和查询

如果数据库实例遇到瓶颈，您的首要任务是查找导致瓶颈的会话和查询。有关有用的博客文章，请参阅利用性能详情分析 Amazon Aurora MySQL 工作负载。

识别导致瓶颈的会话和查询

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。

2. 在导航窗格中，选择性能详情。
3. 选择您的数据库实例。
4. 在 Database load（数据库负载）中，选择 Slice by wait（按等待切片）。
5. 在页面底部，选择 Top SQL（主要 SQL）。

   列表顶部的查询导致数据库负载最高。

遵循连接管理的最佳实践

要管理连接，请考虑以下策略：

- 使用连接池。

  您可以根据需要逐步增加连接数量。有关更多信息，请参阅白皮书 Amazon Aurora MySQL 数据库管理员手册。
- 使用读取器节点重新分配只读流量。

  有关更多信息，请参阅 Aurora 副本 和 Amazon Aurora 端点连接。

如果资源受到限制，则纵向扩展实例

在以下资源中查找节流的示例：

- CPU

  检查您的 Amazon CloudWatch 指标是否存在高 CPU 使用率。
- 网络

  检查 CloudWatch 指标 `network receive throughput` 和 `network transmit throughput` 的值是否增加。如果您的实例已达到实例类的网络带宽限制，请考虑将 RDS 实例纵向扩展到更高的实例类类型。有关更多信息，请参阅 Amazon Aurora 数据库实例类。
- 可用内存

  检查 CloudWatch 指标 `FreeableMemory` 是否有下降。另外，请考虑开启增强监控功能。有关更多信息，请参阅 使用增强监控来监控操作系统指标。

检查首要主机和主要用户

使用性能详情查看首要主机和主要用户。有关更多信息，请参阅 使用 Performance Insights 控制面板分析指标。

查询 performance_schema 表

要获得当前连接和总连接数的准确计数，请查询 performance_schema 表。使用此方法，您可以识别负责创建大量连接的源用户或主机。例如，如下所示查询 performance_schema 表。

```
SELECT * FROM performance_schema.accounts;
SELECT * FROM performance_schema.users;
SELECT * FROM performance_schema.hosts;
```

检查查询的线程状态

如果性能问题仍然存在，请检查查询的线程状态。在 mysql 客户端中，发出以下命令。

```
show processlist;
```

审核您的请求和查询

要查看来自用户账户的请求和查询的性质，请使用 AuroraAurora MySQL 高级审核。要了解如何启用审核，请参阅 在 Amazon Aurora MySQL 数据库集群中使用高级审计。

池化您的数据库连接

考虑使用 Amazon RDS 代理进行连接管理。通过使用 RDS 代理，您可以允许您的应用程序池化和共享数据库连接，以提高其扩展能力。RDS 代理通过在保留应用程序连接的同时自动连接到备用数据库实例，使应用程序能够更好地抵御数据库故障。有关更多信息，请参阅 适用于 Aurora 的 Amazon RDS 代理。

io/table/sql/handler

当工作被委派给存储引擎时，会发生 io/table/sql/handler 事件。

主题

- 支持的引擎版本
- 上下文
- 等待次数增加的可能原因
- 操作

支持的引擎版本

以下引擎版本支持此等待事件信息：

- Aurora MySQL 版本 2 和 3

上下文

事件 io/table 表示等待访问表。无论数据是缓存在缓冲池中还是可在磁盘上访问，都会发生此事件。io/table/sql/handler 事件表示工作负载活动增加。

处理程序是专门处理某种类型的数据或专注于某些特殊任务的例程。例如，事件处理程序接收和提取来自操作系统或用户界面的事件和信号。内存处理程序执行与内存相关的任务。文件输入处理程序是接收文件输入并根据上下文对数据执行特殊任务的函数。

performance_schema.events_waits_current 之类的视图在实际的等待是嵌套的等待事件（例如锁定）时经常显示 io/table/sql/handler。当实际的等待不是 io/table/sql/handler 时，性能详情将报告嵌套的等待事件。当性能详情报告 io/table/sql/handler 时，它表示 I/O 请求的 InnoDB 处理，而不是隐藏的嵌套等待事件。有关更多信息，请参阅 MySQL 参考手册中的 Performance Schema“原子”和“分子”事件。

io/table/sql/handler 事件通常显示在具有 io/aurora_redo_log_flush 之类的输入/输出等待的主要等待事件中。

等待次数增加的可能原因

在性能详情中，io/table/sql/handler 事件突然激增表示工作负载活动增加。活动增加意味着输入/输出增加。

当底层嵌套事件为锁定等待时，性能详情会筛选嵌套事件 ID，但不会报告 io/table/sql/handler 等待。例如，如果根本原因事件是 synch/mutex/innodb/aurora_lock_thread_slot_futex，性能详情会将此等待显示在主要等待事件中而不是 io/table/sql/handler 中。

在 performance_schema.events_waits_current 之类的视图中，当实际的等待是嵌套的等待事件（例如锁定）时经常显示 io/table/sql/handler 的等待。当实际等待与 io/table/sql/handler 不同时，性能详情会查找嵌套等待并报告实际等待而不是 io/table/sql/handler。当性能详情报告 io/table/sql/handler 时，实际的等待为 io/table/sql/handler，而不是隐藏的嵌套等待事件。有关更多信息，请参阅 MySQL 5.7 参考手册中的 Performance Schema“原子”和“分子”事件。

操作

如果此等待事件主导着数据库活动，它不一定表示性能问题。当数据库处于活动状态时，等待事件始终显示在最前面。您只需要在性能下降时采取行动。

根据您看到的其他等待事件，我们建议采取不同的操作。

主题

- 确定导致事件的会话和查询
- 检查与性能详情计数器指标的关联性
- 检查其他相关的等待事件

确定导致事件的会话和查询

通常，具有中等到大量负载的数据库都会有等待事件。如果性能最佳，等待事件可能是可以接受的。如果性能不佳，请检查数据库花费最多时间的位置。查看导致最高负载的等待事件，并了解是否可以优化数据库和应用程序以减少这些事件。

查找负责高负载的 SQL 查询

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择性能详情。
3. 选择一个数据库实例。将为该数据库实例显示性能详情控制面板。
4. 在 Database load（数据库负载）图表中，选择 Slice by wait（按等待切片）。
5. 在页面底部，选择 Top SQL（热门 SQL）。

   该图表列出了负责加载的 SQL 查询。排在列表前面的负载负有最大的责任。为了解决瓶颈，请关注这些语句。

有关使用性能详情进行故障排除的有用概览，请参阅博客文章利用性能详情分析 Amazon Aurora MySQL 工作负载。

检查与性能详情计数器指标的关联性

检查性能详情计数器指标，如 `Innodb_rows_changed`。如果计数器指标与 `io/table/sql/handler` 相关，请按照以下步骤操作：

1. 在性能详情中，查找负责 io/table/sql/handler 主要等待事件的 SQL 语句。如果可能，请优化此语句，使其返回的行更少。
2. 从 schema_table_statistics 和 x$schema_table_statistics 视图中查看主要的表。这些视图显示了每个表花费的时间。有关更多信息，请参阅 MySQL 参考手册中的 schema_table_statistics 和 x$schema_table_statistics 视图。

   预设情况下，行按总等待时间降序排序。争用最多的表首先显示。输出表明时间是花在读取、写入、获取、插入、更新还是删除上。

```
mysql> select * from sys.schema_table_statistics limit 1\G

*************************** 1. row ***************************
     table_schema: read_only_db
       table_name: sbtest41
    total_latency: 54.11 m
     rows_fetched: 6001557
    fetch_latency: 39.14 m
    rows_inserted: 14833
   insert_latency: 5.78 m
     rows_updated: 30470
   update_latency: 5.39 m
     rows_deleted: 14833
   delete_latency: 3.81 m
 io_read_requests: NULL
          io_read: NULL
  io_read_latency: NULL
io_write_requests: NULL
         io_write: NULL
 io_write_latency: NULL
 io_misc_requests: NULL
  io_misc_latency: NULL
1 row in set (0.11 sec)
```

检查其他相关的等待事件

如果 synch/sxlock/innodb/btr_search_latch 和 io/table/sql/handler 是造成数据库加载异常的最大因素，检查 innodb_adaptive_hash_index 变量是否已开启。如果是，请考虑增加 innodb_adaptive_hash_index_parts 参数值。

如果自适应哈希索引已关闭，请考虑将其开启。要了解有关 MySQL 自适应哈希索引的更多信息，请参阅以下资源：

- Percona 网站上的文章 InnoDB 中的自适应哈希索引是否适合我的工作负载？
- MySQL 参考手册中的自适应哈希索引
- Percona 网站上的文章 MySQL InnoDB 中的争用：信号灯部分的有用信息

> **Note**
>
> Aurora 读取器数据库实例不支持自适应哈希索引。
> 在某些情况下，当 `synch/sxlock/innodb/btr_search_latch` 和 `io/table/sql/handler` 起主导作用时，读取器实例的性能可能较差。如果是这样，请考虑临时将工作负载重新导向到写入器数据库实例并开启自适应哈希索引。

synch/cond/innodb/row_lock_wait

当一个会话锁定了一个行以进行更新，而另一个会话尝试更新同一行时，将会发生 `synch/cond/innodb/row_lock_wait` 事件。有关更多信息，请参阅 MySQL 文档中的 InnoDB 锁定。

支持的引擎版本

以下引擎版本支持此等待事件信息：

- Aurora MySQL 版本 3

等待次数增加的可能原因

多个数据操作语言 (DML) 语句同时访问相同的一行或多行。

操作

根据您看到的其他等待事件，我们建议采取不同的操作。

主题

- 查找并响应负责此等待事件的 SQL 语句
- 查找并响应阻止会话

查找并响应负责此等待事件的 SQL 语句

使用性能详情可识别应对此等待事件负责的 SQL 语句。请考虑以下策略：

- 如果行锁定是持续存在的问题，请考虑重写应用程序以使用乐观锁。
- 使用多行语句。
- 将工作负载分散到不同的数据库对象上。您可以通过分区执行此操作。
- 检查 innodb_lock_wait_timeout 参数的值。它控制事务在生成超时错误之前需要等待多长时间。

有关使用性能详情进行故障排除的有用概览，请参阅博客文章利用性能详情分析 Amazon Aurora MySQL 工作负载。

查找并响应阻止会话

确定阻止会话处于空闲状态还是活动状态。另外，了解会话是来自应用程序还是活跃用户。

要识别持有锁的会话，您可以运行 SHOW ENGINE INNODB STATUS。下面的示例显示了示例输出。

```
mysql> SHOW ENGINE INNODB STATUS;

---TRANSACTION 1688153, ACTIVE 82 sec starting index read
mysql tables in use 1, locked 1
LOCK WAIT 2 lock struct(s), heap size 1136, 2 row lock(s)
MySQL thread id 4244, OS thread handle 70369524330224, query id 4020834 172.31.14.179
 reinvent executing
select id1 from test.t1 where id1=1 for update
------- TRX HAS BEEN WAITING 24 SEC FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 11 page no 4 n bits 72 index GEN_CLUST_INDEX of table test.t1 trx
 id 1688153 lock_mode X waiting
Record lock, heap no 2 PHYSICAL RECORD: n_fields 5; compact format; info bits 0
```

或者，您可以使用以下查询来提取有关当前锁定的详细信息。

```
mysql> SELECT p1.id waiting_thread,
    p1.user waiting_user,
    p1.host waiting_host,
    it1.trx_query waiting_query,
    ilw.requesting_engine_transaction_id waiting_transaction,
    ilw.blocking_engine_lock_id blocking_lock,
    il.lock_mode blocking_mode,
    il.lock_type blocking_type,
    ilw.blocking_engine_transaction_id blocking_transaction,
    CASE it.trx_state
        WHEN 'LOCK WAIT'
```

```
        THEN it.trx_state
        ELSE p.state end blocker_state,
    concat(il.object_schema,'.', il.object_name) as locked_table,
    it.trx_mysql_thread_id blocker_thread,
    p.user blocker_user,
    p.host blocker_host
FROM performance_schema.data_lock_waits ilw
JOIN performance_schema.data_locks il
ON ilw.blocking_engine_lock_id = il.engine_lock_id
AND ilw.blocking_engine_transaction_id = il.engine_transaction_id
JOIN information_schema.innodb_trx it
ON ilw.blocking_engine_transaction_id = it.trx_id join information_schema.processlist p
ON it.trx_mysql_thread_id = p.id join information_schema.innodb_trx it1
ON ilw.requesting_engine_transaction_id = it1.trx_id join
 information_schema.processlist p1
ON it1.trx_mysql_thread_id = p1.id\G

*************************** 1. row ***************************
waiting_thread: 4244
waiting_user: reinvent
waiting_host: 123.456.789.012:18158
waiting_query: select id1 from test.t1 where id1=1 for update
waiting_transaction: 1688153
blocking_lock: 70369562074216:11:4:2:70369549808672
blocking_mode: X
blocking_type: RECORD
blocking_transaction: 1688142
blocker_state: User sleep
locked_table: test.t1
blocker_thread: 4243
blocker_user: reinvent
blocker_host: 123.456.789.012:18156
1 row in set (0.00 sec)
```

当您识别会话时，您的选项包括：

- 联系应用程序拥有者或用户。
- 如果阻止会话处于空闲状态，请考虑结束阻止会话。此操作可能会触发长时间回滚。要了解如何结束会话，请参阅结束会话或查询。

有关识别阻止事务的更多信息，请参阅 MySQL 文档中的 Using InnoDB transaction and locking information。

synch/cond/innodb/row_lock_wait_cond

当一个会话锁定了一个行以进行更新，而另一个会话尝试更新同一行时，将会发生 `synch/cond/innodb/row_lock_wait_cond` 事件。有关更多信息，请参阅 MySQL 文档中的 InnoDB 锁定。

支持的引擎版本

以下引擎版本支持此等待事件信息：

- Aurora MySQL 版本 2

等待次数增加的可能原因

多个数据操作语言 (DML) 语句同时访问相同的一行或多行。

操作

根据您看到的其他等待事件，我们建议采取不同的操作。

主题

- 查找并响应负责此等待事件的 SQL 语句
- 查找并响应阻止会话

查找并响应负责此等待事件的 SQL 语句

使用性能详情可识别应对此等待事件负责的 SQL 语句。请考虑以下策略：

- 如果行锁定是持续存在的问题，请考虑重写应用程序以使用乐观锁。
- 使用多行语句。
- 将工作负载分散到不同的数据库对象上。您可以通过分区执行此操作。
- 检查 `innodb_lock_wait_timeout` 参数的值。它控制事务在生成超时错误之前需要等待多长时间。

有关使用性能详情进行故障排除的有用概览，请参阅博客文章利用性能详情分析 Amazon Aurora MySQL 工作负载。

查找并响应阻止会话

确定阻止会话处于空闲状态还是活动状态。另外，了解会话是来自应用程序还是活跃用户。

要识别持有锁的会话，您可以运行 SHOW ENGINE INNODB STATUS。下面的示例显示了示例输出。

```
mysql> SHOW ENGINE INNODB STATUS;

---TRANSACTION 2771110, ACTIVE 112 sec starting index read
mysql tables in use 1, locked 1
LOCK WAIT 2 lock struct(s), heap size 1136, 1 row lock(s)
MySQL thread id 24, OS thread handle 70369573642160, query id 13271336 172.31.14.179
 reinvent Sending data
select id1 from test.t1 where id1=1 for update
------- TRX HAS BEEN WAITING 43 SEC FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 11 page no 3 n bits 0 index GEN_CLUST_INDEX of table test.t1 trx
 id 2771110 lock_mode X waiting
Record lock, heap no 2 PHYSICAL RECORD: n_fields 5; compact format; info bits 0
```

或者，您可以使用以下查询来提取有关当前锁定的详细信息。

```
mysql> SELECT p1.id waiting_thread,
              p1.user waiting_user,
              p1.host waiting_host,
              it1.trx_query waiting_query,
              ilw.requesting_trx_id waiting_transaction,
              ilw.blocking_lock_id blocking_lock,
              il.lock_mode blocking_mode,
              il.lock_type blocking_type,
              ilw.blocking_trx_id blocking_transaction,
              CASE it.trx_state
                WHEN 'LOCK WAIT'
                THEN it.trx_state
                ELSE p.state
              END blocker_state,
              il.lock_table locked_table,
              it.trx_mysql_thread_id blocker_thread,
              p.user blocker_user,
              p.host blocker_host
       FROM information_schema.innodb_lock_waits ilw
       JOIN information_schema.innodb_locks il
         ON ilw.blocking_lock_id = il.lock_id
        AND ilw.blocking_trx_id = il.lock_trx_id
       JOIN information_schema.innodb_trx it
         ON ilw.blocking_trx_id = it.trx_id
       JOIN information_schema.processlist p
         ON it.trx_mysql_thread_id = p.id
```

```
       JOIN information_schema.innodb_trx it1
         ON ilw.requesting_trx_id = it1.trx_id
       JOIN information_schema.processlist p1
         ON it1.trx_mysql_thread_id = p1.id\G

*************************** 1. row ***************************
      waiting_thread: 3561959471
        waiting_user: reinvent
        waiting_host: 123.456.789.012:20485
       waiting_query: select id1 from test.t1 where id1=1 for update
 waiting_transaction: 312337314
       blocking_lock: 312337287:261:3:2
       blocking_mode: X
       blocking_type: RECORD
blocking_transaction: 312337287
       blocker_state: User sleep
        locked_table: `test`.`t1`
      blocker_thread: 3561223876
        blocker_user: reinvent
        blocker_host: 123.456.789.012:17746
1 row in set (0.04 sec)
```

当您识别会话时，您的选项包括：

- 联系应用程序拥有者或用户。
- 如果阻止会话处于空闲状态，请考虑结束阻止会话。此操作可能会触发长时间回滚。要了解如何结束会话，请参阅结束会话或查询。

有关识别阻止事务的更多信息，请参阅 MySQL 文档中的 Using InnoDB transaction and locking information。

synch/cond/sql/MDL_context::COND_wait_status

当有正等待表元数据锁定的线程时，会发生 `synch/cond/sql/MDL_context::COND_wait_status` 事件。

主题

- 支持的引擎版本
- 上下文
- 等待次数增加的可能原因

- 操作

支持的引擎版本

以下引擎版本支持此等待事件信息：

- Aurora MySQL 版本 2 和 3

上下文

事件 synch/cond/sql/MDL_context::COND_wait_status 表示有正等待表元数据锁定的线程。在某些情况下，一个会话在表上保持元数据锁定，而另一个会话则试图在同一个表上获取同一个锁定。在这种情况下，第二个会话将等待 synch/cond/sql/MDL_context::COND_wait_status 等待事件。

MySQL 使用元数据锁定来管理对数据库对象的并发访问并确保数据一致性。元数据锁定适用于通过 get_lock 函数获取的表、架构、计划事件、表空间和用户锁定，以及存储的程序。存储的程序包括过程、函数和触发器。有关更多信息，请参阅 MySQL 文档中的元数据锁定。

MySQL 进程列表显示此会话处于状态 waiting for metadata lock。在性能详情中，如果 Performance_schema 已开启，将会显示事件 synch/cond/sql/MDL_context::COND_wait_status。

等待元数据锁定的查询的原定设置超时时间取决于 lock_wait_timeout 参数的值，该值原定设置为 31536000 秒（365 天）。

有关不同 InnoDB 锁定以及可能导致冲突的锁定类型的更多详细信息，请参阅 MySQL 文档中的 InnoDB 锁定。

等待次数增加的可能原因

当 synch/cond/sql/MDL_context::COND_wait_status 事件的发生率超过正常（可能表示性能问题）时，典型原因包括以下几点：

长时间运行的事务

一个或多个事务正在修改大量数据并在表上保持锁定很长一段时间。

空闲事务

一个或多个事务长时间保持打开状态，而不会被提交或回退。

大型表格上的 DDL 语句

一个或多个数据定义语句 (DDL) 语句（例如 ALTER TABLE 命令）在非常大的表上运行。

显式表锁定

表上存在显式锁定，但没有及时释放。例如，应用程序可能会不适当运行 LOCK TABLE 语句。

操作

根据等待事件的原因以及 Aurora MySQL 数据库集群的版本，我们建议采取不同的操作。

主题

- 确定导致事件的会话和查询
- 检查过去的事件
- 在 Aurora MySQL 版本 2 上运行查询
- 响应阻止会话

确定导致事件的会话和查询

您可以使用性能详情显示被 synch/cond/sql/MDL_context::COND_wait_status 等待事件阻止的查询。但是，要识别阻止的会话，请从中数据库集群上的 performance_schema 和 information_schema 中查询元数据表。

通常，具有中等到大量负载的数据库都会有等待事件。如果性能最佳，等待事件可能是可以接受的。如果性能不佳，请检查数据库花费最多时间的位置。查看导致最高负载的等待事件，并了解是否可以优化数据库和应用程序以减少这些事件。

查找负责高负载的 SQL 查询

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择性能详情。
3. 选择一个数据库实例。该数据库实例的性能详情控制面板出现。
4. 在 Database load（数据库负载）图表中，选择 Slice by wait（按等待切片）。
5. 在页面底部，选择 Top SQL（热门 SQL）。

   该图表列出了负责加载的 SQL 查询。排在列表前面的负载负有最大的责任。为了解决瓶颈，请关注这些语句。

有关使用性能详情进行故障排除的有用概览，请参阅 AWS 数据库博客文章使用性能详情分析 Amazon Aurora MySQL 工作负载。

检查过去的事件

您可以深入了解此等待事件，以检查过去发生的事件。为此，请完成以下操作：

- 检查数据操作语言 (DML) 和 DDL 吞吐量和延迟，以查看工作负载是否有任何变化。

  您可以使用性能详情查找问题发生时等待此事件的查询。此外，您还可以查看接近发布时间运行的查询的摘要。
- 如果为数据库集群开启了审计日志或常规日志，则可以检查在等待事务中涉及的对象 (schema.table) 上运行的所有查询。您还可以检查在事务之前已完成运行的查询。

可用于排除过去事件故障的信息有限。执行这些检查不会显示哪个对象正在等待信息。但是，您可以识别事件发生时负载大的表以及在发布时导致冲突的频繁操作行集。然后，您可以使用此信息在测试环境中重现问题并提供有关其原因的洞察。

在 Aurora MySQL 版本 2 上运行查询

在 Aurora MySQL 版本 2 中，您可以通过查询 `performance_schema` 表或 `sys` 架构视图来直接识别被阻止的会话。一个例子可以说明如何查询表以识别阻止的查询和会话。

在以下进程列表输出中，连接 ID 89 正在等待元数据锁定，它正在运行 `TRUNCATE TABLE` 命令。在 `performance_schema` 表或 `sys` 架构视图上的查询中，输出显示阻止会话为 76。

```
MySQL [(none)]> select @@version, @@aurora_version;
+-----------+------------------+
| @@version | @@aurora_version |
+-----------+------------------+
| 5.7.12    | 2.11.5           |
+-----------+------------------+
1 row in set (0.01 sec)

MySQL [(none)]> show processlist;
+----+-----------------+--------------------+-----------+---------+------
+---------------------------------+------------------------------+
| Id | User            | Host               | db        | Command | Time | State
                    | Info                         |
+----+-----------------+--------------------+-----------+---------+------
+---------------------------------+------------------------------+
```

```
|  2 | rdsadmin        | localhost          | NULL      | Sleep   |    0 | NULL
                    | NULL                           |
|  4 | rdsadmin        | localhost          | NULL      | Sleep   |    2 | NULL
                    | NULL                           |
|  5 | rdsadmin        | localhost          | NULL      | Sleep   |    1 | NULL
                    | NULL                           |
| 20 | rdsadmin        | localhost          | NULL      | Sleep   |    0 | NULL
                    | NULL                           |
| 21 | rdsadmin        | localhost          | NULL      | Sleep   |  261 | NULL
                    | NULL                           |
| 66 | auroramysql5712 | 172.31.21.51:52154 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                           |
| 67 | auroramysql5712 | 172.31.21.51:52158 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                           |
| 68 | auroramysql5712 | 172.31.21.51:52150 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                           |
| 69 | auroramysql5712 | 172.31.21.51:52162 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                           |
| 70 | auroramysql5712 | 172.31.21.51:52160 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                           |
| 71 | auroramysql5712 | 172.31.21.51:52152 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                           |
| 72 | auroramysql5712 | 172.31.21.51:52156 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                           |
| 73 | auroramysql5712 | 172.31.21.51:52164 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                           |
| 74 | auroramysql5712 | 172.31.21.51:52166 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                           |
| 75 | auroramysql5712 | 172.31.21.51:52168 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                           |
| 76 | auroramysql5712 | 172.31.21.51:52170 | NULL      | Query   |    0 | starting
                    | show processlist               |
| 88 | auroramysql5712 | 172.31.21.51:52194 | NULL      | Query   |   22 | User sleep
                    | select sleep(10000)            |
| 89 | auroramysql5712 | 172.31.21.51:52196 | NULL      | Query   |    5 | Waiting for
 table metadata lock | truncate table sbtest.sbtest1 |
+----+-----------------+--------------------+-----------+---------+------
+---------------------------------+-------------------------------+
18 rows in set (0.00 sec)
```

接下来，performance_schema 表或 sys 架构视图上的查询会显示阻止会话为 76。

```
MySQL [(none)]> select * from sys.schema_table_lock_waits;

+---------------+-------------+-------------------+-------------
+-------------------------------+-------------------+-----------------------
+-------------------------------+--------------------+-----------------------------
+-------------------------------+--------------------+-------------
+-------------------------------+--------------------+-----------------------
+-------------------------+------------------------------+
| object_schema | object_name | waiting_thread_id | waiting_pid | waiting_account
           | waiting_lock_type | waiting_lock_duration | waiting_query
   | waiting_query_secs | waiting_query_rows_affected | waiting_query_rows_examined |
 blocking_thread_id | blocking_pid | blocking_account              | blocking_lock_type
 | blocking_lock_duration | sql_kill_blocking_query | sql_kill_blocking_connection |
+---------------+-------------+-------------------+-------------
+-------------------------------+-------------------+-----------------------
+-------------------------------+--------------------+-----------------------------
+-------------------------------+--------------------+-------------
+-------------------------------+--------------------+-----------------------
+-------------------------+------------------------------+
| sbtest        | sbtest1     |               121 |          89 |
 auroramysql5712@192.0.2.0    | EXCLUSIVE         | TRANSACTION           | truncate
 table sbtest.sbtest1 |                 10 |                            0 |
                0 |                108 |           76 | auroramysql5712@192.0.2.0    |
 SHARED_READ        | TRANSACTION            | KILL QUERY 76           | KILL 76
               |
+---------------+-------------+-------------------+-------------
+-------------------------------+-------------------+-----------------------
+-------------------------------+--------------------+-----------------------------
+-------------------------------+--------------------+-------------
+-------------------------------+--------------------+-----------------------
+-------------------------+------------------------------+
1 row in set (0.00 sec)
```

响应阻止会话

当您识别会话时，您的选项包括：

- 联系应用程序拥有者或用户。
- 如果阻止会话处于空闲状态，请考虑结束阻止会话。此操作可能会触发长时间回滚。要了解如何结束会话，请参阅结束会话或查询。

有关识别阻止事务的更多信息，请参阅 MySQL 文档中的使用 InnoDB 事务和锁定信息。

synch/mutex/innodb/aurora_lock_thread_slot_futex

当一个会话锁定了一个行以进行更新，而另一个会话尝试更新同一行时，将会发生 `synch/mutex/innodb/aurora_lock_thread_slot_futex` 事件。有关更多信息，请参阅 MySQL 参考中的 InnoDB 锁定。

支持的引擎版本

以下引擎版本支持此等待事件信息：

- Aurora MySQL 版本 2

等待次数增加的可能原因

多个数据操作语言 (DML) 语句同时访问相同的一行或多行。

操作

根据您看到的其他等待事件，我们建议采取不同的操作。

主题

- 查找并响应负责此等待事件的 SQL 语句
- 查找并响应阻止会话

查找并响应负责此等待事件的 SQL 语句

使用性能详情可识别应对此等待事件负责的 SQL 语句。请考虑以下策略：

- 如果行锁定是持续存在的问题，请考虑重写应用程序以使用乐观锁。
- 使用多行语句。
- 将工作负载分散到不同的数据库对象上。您可以通过分区执行此操作。
- 检查 `innodb_lock_wait_timeout` 参数的值。它控制事务在生成超时错误之前需要等待多长时间。

有关使用性能详情进行故障排除的有用概览，请参阅博客文章利用性能详情分析 Amazon Aurora MySQL 工作负载。

查找并响应阻止会话

确定阻止会话处于空闲状态还是活动状态。另外，了解会话是来自应用程序还是活跃用户。

要识别持有锁的会话，您可以运行 SHOW ENGINE INNODB STATUS。下面的示例显示了示例输出。

```
mysql> SHOW ENGINE INNODB STATUS;

----------------------TRANSACTION 302631452, ACTIVE 2 sec starting index read
mysql tables in use 1, locked 1
LOCK WAIT 2 lock struct(s), heap size 376, 1 row lock(s)
MySQL thread id 80109, OS thread handle 0x2ae915060700, query id 938819 10.0.4.12
 reinvent updating
UPDATE sbtest1 SET k=k+1 WHERE id=503
------- TRX HAS BEEN WAITING 2 SEC FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 148 page no 11 n bits 30 index `PRIMARY` of table
 `sysbench2`.`sbtest1` trx id 302631452 lock_mode X locks rec but not gap waiting
Record lock, heap no 30 PHYSICAL RECORD: n_fields 6; compact format; info bits 0
```

或者，您可以使用以下查询来提取有关当前锁定的详细信息。

```
mysql> SELECT p1.id waiting_thread,
              p1.user waiting_user,
              p1.host waiting_host,
              it1.trx_query waiting_query,
              ilw.requesting_trx_id waiting_transaction,
              ilw.blocking_lock_id blocking_lock,
              il.lock_mode blocking_mode,
              il.lock_type blocking_type,
              ilw.blocking_trx_id blocking_transaction,
              CASE it.trx_state
                WHEN 'LOCK WAIT'
                THEN it.trx_state
                ELSE p.state
              END blocker_state,
              il.lock_table locked_table,
              it.trx_mysql_thread_id blocker_thread,
              p.user blocker_user,
              p.host blocker_host
       FROM information_schema.innodb_lock_waits ilw
       JOIN information_schema.innodb_locks il
         ON ilw.blocking_lock_id = il.lock_id
        AND ilw.blocking_trx_id = il.lock_trx_id
```

```
       JOIN information_schema.innodb_trx it
         ON ilw.blocking_trx_id = it.trx_id
       JOIN information_schema.processlist p
         ON it.trx_mysql_thread_id = p.id
       JOIN information_schema.innodb_trx it1
         ON ilw.requesting_trx_id = it1.trx_id
       JOIN information_schema.processlist p1
         ON it1.trx_mysql_thread_id = p1.id\G

*************************** 1. row ***************************
      waiting_thread: 3561959471
        waiting_user: reinvent
        waiting_host: 123.456.789.012:20485
       waiting_query: select id1 from test.t1 where id1=1 for update
 waiting_transaction: 312337314
       blocking_lock: 312337287:261:3:2
       blocking_mode: X
       blocking_type: RECORD
blocking_transaction: 312337287
       blocker_state: User sleep
        locked_table: `test`.`t1`
      blocker_thread: 3561223876
        blocker_user: reinvent
        blocker_host: 123.456.789.012:17746
1 row in set (0.04 sec)
```

当您识别会话时，您的选项包括：

- 联系应用程序拥有者或用户。
- 如果阻止会话处于空闲状态，请考虑结束阻止会话。此操作可能会触发长时间回滚。要了解如何结束会话，请参阅结束会话或查询。

有关识别阻止事务的更多信息，请参阅 MySQL 参考手册中的使用 InnoDB 事务和锁定信息。

synch/mutex/innodb/buf_pool_mutex

当线程在 InnoDB 缓冲池上获取锁定以访问内存中的页面时，将发生 `synch/mutex/innodb/buf_pool_mutex` 事件。

主题

- 相关引擎版本

- 上下文
- 等待次数增加的可能原因
- 操作

相关引擎版本

以下引擎版本支持此等待事件信息：

- Aurora MySQL 版本 2

上下文

buf_pool 互斥锁是保护缓冲池的控制数据结构的单个互斥锁。

有关更多信息，请参阅 MySQL 文档中的使用性能架构监控 InnoDB 互斥锁等待。

等待次数增加的可能原因

这是特定于工作负载的等待事件。synch/mutex/innodb/buf_pool_mutex 显示在主要等待事件中的常见原因包括以下各项：

- 缓冲池不够大，无法容纳工作数据集。
- 工作负载更加特定于数据库中特定表中的某些页面，从而导致缓冲池中发生争用。

操作

根据等待事件的原因，我们建议采取不同的操作。

主题

- 确定导致事件的会话和查询
- 使用性能详情
- 创建 Aurora 副本
- 检查缓冲池大小
- 监控全局状态历史记录

确定导致事件的会话和查询

通常，具有中等到大量负载的数据库都会有等待事件。如果性能最佳，等待事件可能是可以接受的。如果性能不佳，请检查数据库花费最多时间的位置。查看导致最高负载的等待事件，并了解是否可以优化数据库和应用程序以减少这些事件。

在 AWS 管理控制台中查看主要的 SQL 图表

1. 通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Performance Insights。
3. 选择一个数据库实例。将为该数据库实例显示性能详情控制面板。
4. 在 Database load（数据库负载）图表中，选择 Slice by wait（按等待切片）。
5. 在 Database load（数据库加载）图表下，选择 Top SQL（主要 SQL）。

   该图表列出了负责加载的 SQL 查询。排在列表前面的负载负有最大的责任。为了解决瓶颈，请关注这些语句。

有关使用性能详情进行故障排除的有用概览，请参阅博客文章利用性能详情分析 Amazon Aurora MySQL 工作负载。

使用性能详情

此事件与工作负载有关。您可以使用性能详情执行以下操作：

- 确定等待事件的开始时间，以及在那段时间内，应用程序日志或相关来源的工作负载是否有任何变化。
- 识别应对此等待事件负责的 SQL 语句。检查查询的执行计划，以确保这些查询经过优化并且在使用适当的索引。

  如果负责等待事件的主要查询与同一数据库对象或表相关，则考虑对该对象或表进行分区。

创建 Aurora 副本

您可以创建 Aurora 副本以提供只读流量。您还可以使用 Aurora Auto Scaling 来处理读取流量的激增。确保在 Aurora 副本上运行计划的只读任务和逻辑备份。

有关更多信息，请参阅 Amazon Aurora Auto Scaling 与 Aurora 副本结合使用。

检查缓冲池大小

通过查看指标 innodb_buffer_pool_wait_free 来检查缓冲池大小是否足以应付工作负载。如果此指标的值很高且持续增加，则表明缓冲池的大小不足以处理工作负载。如果 innodb_buffer_pool_size 设置正确，innodb_buffer_pool_wait_free 的值应该很小。有关更多信息，请参阅 MySQL 文档中的 Innodb_buffer_pool_wait_free。

如果数据库实例有足够的内存用于会话缓冲区和操作系统任务，则增加缓冲池大小。如果没有，请将数据库实例更改为较大的数据库实例类，以获取可分配给缓冲池的额外内存。

> **Note**
>
> Aurora MySQL 基于配置的 innodb_buffer_pool_size 自动调整 innodb_buffer_pool_instances 的值。

监控全局状态历史记录

通过监控状态变量的更改率，您可以检测数据库实例的锁定或内存问题。如果尚未开启“全局状态历史记录”(GoSH)，请将其打开。有关 GoSH 的更多信息，请参阅管理全局状态历史记录。

您还可以创建自定义 Amazon CloudWatch 指标以监控状态变量。有关更多信息，请参阅发布自定义指标。

synch/mutex/innodb/fil_system_mutex

当会话等待访问表空间内存缓存时，会发生 synch/mutex/innodb/fil_system_mutex 事件。

主题

- 支持的引擎版本
- 上下文
- 等待次数增加的可能原因
- 操作

支持的引擎版本

以下引擎版本支持此等待事件信息：

- Aurora MySQL 版本 2 和 3

上下文

InnoDB 使用表空间来管理表和日志文件的存储区域。表空间内存缓存是一种全局内存结构，用于维护有关表空间的信息。MySQL 使用 synch/mutex/innodb/fil_system_mutex 等待控制对表空间内存缓存的并发访问。

事件 synch/mutex/innodb/fil_system_mutex 表示当前有多个操作需要检索和操作同一表空间的表空间内存缓存中的信息。

等待次数增加的可能原因

当 synch/mutex/innodb/fil_system_mutex 事件的发生率超过正常（可能表示性能问题）时，该情况通常会在以下所有条件均存在时发生：

- 更新或删除同一表中的数据的并发数据操作语言 (DML) 操作增加。
- 此表的表空间非常大，有很多数据页面。
- 这些数据页面的填充系数很低。

操作

根据等待事件的原因，我们建议采取不同的操作。

主题

- 确定导致事件的会话和查询
- 在非高峰时段重新组织大型表

确定导致事件的会话和查询

通常，具有中等到大量负载的数据库都会有等待事件。如果性能最佳，等待事件可能是可以接受的。如果性能不佳，请检查数据库花费最多时间的位置。查看导致最高负载的等待事件，并了解是否可以优化数据库和应用程序以减少这些事件。

查找负责高负载的 SQL 查询

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择性能详情。
3. 选择一个数据库实例。该数据库实例的性能详情控制面板出现。

4. 在 Database load（数据库负载）图表中，选择 Slice by wait（按等待切片）。
5. 在页面底部，选择 Top SQL（热门 SQL）。

   该图表列出了负责加载的 SQL 查询。排在列表前面的负载负有最大的责任。为了解决瓶颈，请关注这些语句。

有关使用性能详情进行故障排除的有用概览，请参阅博客文章利用性能详情分析 Amazon Aurora MySQL 工作负载。

找出哪些查询导致了大量 synch/mutex/innodb/fil_system_mutex 等待的另一个方法是检查 performance_schema，如以下示例所示。

```
mysql> select * from performance_schema.events_waits_current where EVENT_NAME='wait/
synch/mutex/innodb/fil_system_mutex'\G
*************************** 1. row ***************************
            THREAD_ID: 19
             EVENT_ID: 195057
         END_EVENT_ID: 195057
           EVENT_NAME: wait/synch/mutex/innodb/fil_system_mutex
               SOURCE: fil0fil.cc:6700
          TIMER_START: 1010146190118400
            TIMER_END: 1010146196524000
           TIMER_WAIT: 6405600
                SPINS: NULL
        OBJECT_SCHEMA: NULL
          OBJECT_NAME: NULL
           INDEX_NAME: NULL
          OBJECT_TYPE: NULL
OBJECT_INSTANCE_BEGIN: 47285552262176
     NESTING_EVENT_ID: NULL
   NESTING_EVENT_TYPE: NULL
            OPERATION: lock
      NUMBER_OF_BYTES: NULL
                FLAGS: NULL
*************************** 2. row ***************************
            THREAD_ID: 23
             EVENT_ID: 5480
         END_EVENT_ID: 5480
           EVENT_NAME: wait/synch/mutex/innodb/fil_system_mutex
               SOURCE: fil0fil.cc:5906
          TIMER_START: 995269979908800
            TIMER_END: 995269980159200
```

```
           TIMER_WAIT: 250400
                SPINS: NULL
        OBJECT_SCHEMA: NULL
          OBJECT_NAME: NULL
           INDEX_NAME: NULL
          OBJECT_TYPE: NULL
OBJECT_INSTANCE_BEGIN: 47285552262176
     NESTING_EVENT_ID: NULL
   NESTING_EVENT_TYPE: NULL
            OPERATION: lock
      NUMBER_OF_BYTES: NULL
                FLAGS: NULL
*************************** 3. row ***************************
            THREAD_ID: 55
             EVENT_ID: 23233794
         END_EVENT_ID: NULL
           EVENT_NAME: wait/synch/mutex/innodb/fil_system_mutex
               SOURCE: fil0fil.cc:449
          TIMER_START: 1010492125341600
            TIMER_END: 1010494304900000
           TIMER_WAIT: 2179558400
                SPINS: NULL
        OBJECT_SCHEMA: NULL
          OBJECT_NAME: NULL
           INDEX_NAME: NULL
          OBJECT_TYPE: NULL
OBJECT_INSTANCE_BEGIN: 47285552262176
     NESTING_EVENT_ID: 23233786
   NESTING_EVENT_TYPE: WAIT
            OPERATION: lock
      NUMBER_OF_BYTES: NULL
                FLAGS: NULL
```

在非高峰时段重新组织大型表

重新组织您在生产时间之外的维护时段内识别为大量 synch/mutex/innodb/fil_system_mutex 等待事件的来源的大型表。这样做可以确保在快速访问表至关重要时不会进行内部表空间映射清理。有关重新组织表的信息，请参阅 MySQL 参考中的 OPTIMIZE TABLE 语句。

synch/mutex/innodb/trx_sys_mutex

当存在大量事务处理的大量数据库活动时，会发生 synch/mutex/innodb/trx_sys_mutex 事件。

主题

- 相关引擎版本
- 上下文
- 等待次数增加的可能原因
- 操作

相关引擎版本

以下引擎版本支持此等待事件信息：

- Aurora MySQL 版本 2 和 3

上下文

在内部，InnoDB 数据库引擎使用可重复的读取隔离级别和快照来提供读取一致性。这为您提供了创建快照时数据库的时间点视图。

在 InnoDB 中，所有更改都将在数据库到达后立即应用到数据库，无论它们是否已提交。这种方法意味着，如果没有多版本并发控制 (MVCC)，连接到数据库的所有用户都会看到所有更改和最新的行。因此，InnoDB 需要一种方法来跟踪更改，以了解必要时要回滚的内容。

为此，InnoDB 使用了一个事务系统 (`trx_sys`) 来跟踪快照。事务系统执行以下操作：

- 跟踪撤消日志中每行的事务 ID。
- 使用一个名为 `ReadView` 的内部 InnoDB 结构，以帮助确定哪些事务 ID 对于快照可见。

等待次数增加的可能原因

任何需要对事务 ID 进行一致和受控处理（创建、读取、更新和删除）的数据库操作都会生成从 `trx_sys` 到互斥锁的调用。

这些调用发生在三个函数中：

- `trx_sys_mutex_enter` – 创建互斥锁。
- `trx_sys_mutex_exit` – 释放互斥锁。
- `trx_sys_mutex_own` – 测试是否拥有互斥锁。

InnoDB 性能架构工具跟踪所有的 `trx_sys` 互斥锁调用。追踪包括但不限于，在数据库启动或关闭、回滚操作、撤消清理、行读取访问和缓冲池加载时管理 `trx_sys`。具有大量事务的高数据库活动导致 `synch/mutex/innodb/trx_sys_mutex` 出现在主要等待事件之列。

有关更多信息，请参阅 MySQL 文档中的使用性能架构监控 InnoDB 互斥锁等待。

操作

根据等待事件的原因，我们建议采取不同的操作。

主题

- 确定导致事件的会话和查询
- 检查其他等待事件

确定导致事件的会话和查询

通常，具有中等到大量负载的数据库都会有等待事件。如果性能最佳，等待事件可能是可以接受的。如果性能不佳，请检查数据库花费最多时间的位置。查看导致最高负载的等待事件。了解是否可以优化数据库和应用程序以减少这些事件。

要在 AWS Management Console 中查看 Top SQL（主要 SQL）图表

1. 通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Performance Insights。
3. 选择一个数据库实例。将为该数据库实例显示性能详情控制面板。
4. 在 Database load（数据库负载）图表中，选择 Slice by wait（按等待切片）。
5. 在 Database load（数据库加载）图表下，选择 Top SQL（主要 SQL）。

   该图表列出了负责加载的 SQL 查询。排在列表前面的负载负有最大的责任。为了解决瓶颈，请关注这些语句。

有关使用性能详情进行故障排除的有用概览，请参阅博客文章利用性能详情分析 Amazon Aurora MySQL 工作负载。

检查其他等待事件

检查与 `synch/mutex/innodb/trx_sys_mutex` 等待事件相关的其他等待事件。执行此操作可以提供有关工作负载性质的更多信息。大量事务可能会降低吞吐量，但是工作负载也可能使此变得必要。

有关如何优化事务的更多信息，请参阅 MySQL 文档中的优化 InnoDB 事务管理。

synch/sxlock/innodb/hash_table_locks

当必须从存储中读取在缓冲池中找不到的页面时，会发生 `synch/sxlock/innodb/hash_table_locks` 事件。

主题

- 支持的引擎版本
- 上下文
- 等待次数增加的可能原因
- 操作

支持的引擎版本

以下版本支持此等待事件信息：

- Aurora MySQL 版本 2 和 3

上下文

事件 `synch/sxlock/innodb/hash_table_locks` 表示工作负载经常访问未存储在缓冲池中的数据。此等待事件与添加新页面和从缓冲池中移出旧数据有关。存储在缓冲池中的过时数据和新数据必须缓存，因此移出过时的页面以允许缓存新页面。MySQL 使用最近最少使用的 (LRU) 算法将页面从缓冲池中移出。工作负载试图访问尚未加载到缓冲池中的数据或已从缓冲池中移出的数据。

如果工作负载必须访问磁盘上的文件中的数据，或者从缓冲池的 LRU 列表中释放数据块或将数据块添加到缓冲池的 LRU 列表中，就会发生此等待事件。这些操作等待获取共享的排除锁 (SX-lock)。此 SX-lock 用于通过哈希表进行同步，哈希表是内存中的一个表，旨在提高缓冲池访问性能。

有关更多信息，请参阅 MySQL 文档中的缓存池。

等待次数增加的可能原因

当 `synch/sxlock/innodb/hash_table_locks` 等待事件显示超出正常情况，可能表示性能问题，典型原因包括以下几点：

缓冲池大小不足

缓冲池太小，无法将所有经常访问的页面保留在内存中。

繁重的工作负载

工作负载导致缓冲区缓存中频繁发生移出和数据页面重新加载操作。

读取页面时出错

读取缓冲池中的页面存在错误，这可能表明数据损坏。

操作

根据等待事件的原因，我们建议采取不同的操作。

主题

- 增加缓冲池的大小
- 改进数据访问模式
- 减少或避免完整的表扫描
- 检查错误日志是否有页面损坏

增加缓冲池的大小

确保缓冲池的大小适合工作负载。为此，您可以检查缓冲池缓存命中率。通常，如果该值降至 95% 以下，请考虑增加缓冲池大小。较大的缓冲池可以将经常访问的页面在内存中保留更长时间。要增加缓冲池的大小，请修改 `innodb_buffer_pool_size` 参数的值。该参数的原定设置值取决于数据库实例类的大小。有关更多信息，请参阅 Amazon Aurora MySQL 数据库配置的最佳实践。

改进数据访问模式

检查受此等待影响的查询及其执行计划。考虑改进数据访问模式。例如，如果您使用的是 mysqli_result::fetch_array，您可以尝试增加数组获取大小。

您可以使用性能详情来显示可能导致 `synch/sxlock/innodb/hash_table_locks` 等待事件的查询和会话。

查找负责高负载的 SQL 查询

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择性能详情。
3. 选择一个数据库实例。将为该数据库实例显示性能详情控制面板。

4. 在 Database load（数据库负载）图表中，选择 Slice by wait（按等待切片）。
5. 在页面底部，选择 Top SQL（热门 SQL）。

   该图表列出了负责加载的 SQL 查询。排在列表前面的负载负有最大的责任。为了解决瓶颈，请关注这些语句。

有关使用性能详情进行故障排除的有用概览，请参阅 AWS 数据库博客文章使用性能详情分析 Amazon Aurora MySQL 工作负载。

减少或避免完整的表扫描

监控您的工作负载以查看它是否正在运行完整的表扫描，如果是，则减少或避免它们。例如，您可以监控状态变量，例如 `Handler_read_rnd_next`。有关更多信息，请参阅 MySQL 文档中的服务器状态变量。

检查错误日志是否有页面损坏

您可以检查 mysql-error.log 是否有在问题出现时检测到的与损坏相关的消息。错误日志中会显示您可以用来解决问题的消息。您可能需要重新创建报告为已损坏的对象。

使用线程状态优化 Aurora MySQL

下表总结了 Aurora MySQL 的最常见一般线程状态。

一般线程状态	描述
???	此线程状态表示线程正在处理要求使用内部临时表对数据进行排序的 SELECT 语句。
???	此线程状态表示线程正在读取和筛选查询的行以确定正确的结果集。

创建排序索引

`creating sort index` 线程状态表示线程正在处理要求使用内部临时表对数据进行排序的 SELECT 语句。

主题

- 支持的引擎版本
- 上下文
- 等待次数增加的可能原因
- 操作

支持的引擎版本

以下版本支持此线程状态信息：

- Aurora MySQL 版本 2，最高 2.09.2

上下文

当具有 `ORDER BY` 或 `GROUP BY` 子句的查询不能使用现有索引来执行操作时，将出现 `creating sort index` 状态。在这种情况下，MySQL 需要执行更昂贵的 `filesort` 操作。如果结果集不太大，通常在内存中执行此操作。否则，它涉及在磁盘上创建文件。

等待次数增加的可能原因

`creating sort index` 的外观本身并不表明存在问题。如果性能不佳，且您看到频繁的 `creating sort index` 实例，最有可能的原因是使用 `ORDER BY` 或 `GROUP BY` 运算符进行的查询缓慢。

操作

一般指南是查找带有与 `creating sort index` 状态增加相关的 `ORDER BY` 或 `GROUP BY` 子句的查询。然后看看是添加索引还是增加排序缓冲区大小解决了问题。

主题

- 如果性能架构未开启，请打开它
- 识别问题查询
- 检查文件排序使用的解释计划
- 提高排序缓冲区大小

如果性能架构未开启，请打开它

仅在性能架构工具未打开时，性能详情才会报告线程状态。启用性能架构工具后，性能详情会报告等待事件。在调查潜在的性能问题时，性能架构工具可以提供更多洞察和更好的工具。因此，建议您开启性能架构。有关更多信息，请参阅 Aurora MySQL 上性能详情的性能架构概述。

识别问题查询

要识别导致增加 `creating sort index` 状态增加的当前查询，请运行 `show processlist` 并了解是否有任何查询为 `ORDER BY` 或 `GROUP BY`。或者，请运行 `explain for connection N`，其中 N 是具有 `filesort` 的查询的进程列表 ID。

要识别导致这些增加的过去查询，请打开慢查询日志并查找具有 `ORDER BY` 的查询。在慢查询上运行 EXPLAIN 并查找“using filesort”。有关更多信息，请参阅 检查文件排序使用的解释计划。

检查文件排序使用的解释计划

识别具有导致 `creating sort index` 状态的 `ORDER BY` 或 `GROUP BY` 子句的语句。

以下示例显示了如何运行在查询上运行 `explain`。`Extra` 列显示此查询使用 `filesort`。

```
mysql> explain select * from mytable order by c1 limit 10\G
*************************** 1. row ***************************
           id: 1
```

```
  select_type: SIMPLE
        table: mytable
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 2064548
     filtered: 100.00
        Extra: Using filesort
1 row in set, 1 warning (0.01 sec)
```

以下示例显示了在列 c1 上创建索引后在同一查询上运行 EXPLAIN 的结果。

```
mysql> alter table mytable add index (c1);
```

```
mysql> explain select * from mytable order by c1 limit 10\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: mytable
   partitions: NULL
         type: index
possible_keys: NULL
          key: c1
      key_len: 1023
          ref: NULL
         rows: 10
     filtered: 100.00
        Extra: Using index
1 row in set, 1 warning (0.01 sec)
```

有关使用索引进行排序顺序优化的信息，请参阅 MySQL 文档中的 ORDER BY 优化。

提高排序缓冲区大小

要查看特定查询是否需要在磁盘上创建文件的 filesort 进程，请在运行查询后检查 sort_merge_passes 变量值。下面是一个示例。

```
mysql> show session status like 'sort_merge_passes';
+-------------------+-------+
```

```
| Variable_name     | Value |
+-------------------+-------+
| Sort_merge_passes | 0     |
+-------------------+-------+
1 row in set (0.01 sec)

--- run query
mysql> select * from mytable order by u limit 10;
--- run status again:

mysql> show session status like 'sort_merge_passes';
+-------------------+-------+
| Variable_name     | Value |
+-------------------+-------+
| Sort_merge_passes | 0     |
+-------------------+-------+
1 row in set (0.01 sec)
```

如果 sort_merge_passes 的值很高，请考虑增加排序缓冲区的大小。在会话级别应用增加，因为在全局范围内增加它可以显著增加 RAM MySQL 的使用量。以下示例说明如何在运行查询之前更改排序缓冲区的大小。

```
mysql> set session sort_buffer_size=10*1024*1024;
Query OK, 0 rows affected (0.00 sec)
-- run query
```

发送数据

sending data 线程状态表示线程正在读取和筛选查询的行以确定正确的结果集。这个名称具有误导性，因为它意味着该状态为正在传输数据，而不是收集和准备以后发送的数据。

主题

- 支持的引擎版本
- 上下文
- 等待次数增加的可能原因
- 操作

支持的引擎版本

以下版本支持此线程状态信息：

- Aurora MySQL 版本 2，最高 2.09.2

上下文

许多线程状态都是短期的。`sending data` 期间发生的操作倾向于执行大量磁盘或缓存读取。因此，`sending data` 通常是在给定查询生命周期内运行时间最长的状态。当 Aurora MySQL 执行以下操作时会出现此状态：

- 读取和处理 SELECT 语句的行
- 从磁盘或内存中执行大量读取
- 完成特定查询中的所有数据的完整读取
- 从表、索引或存储过程的工作中读取数据
- 对数据进行排序、分组

当 `sending data` 状态完成准备数据后，线程状态 `writing to net` 表示向客户端返回数据。通常，仅当结果集非常大或严重的网络延迟正在减慢传输速度时才捕获 `writing to net`。

等待次数增加的可能原因

`sending data` 的外观本身并不表明存在问题。如果性能不佳，且您看到频繁 `sending data` 实例，最可能的原因如下所示。

主题

- 低效的查询
- 不理想的服务器配置

低效的查询

在大多数情况下，造成此状态的原因是查询没有使用适当的索引来查找特定查询的结果集。例如，考虑查询读取在加利福尼亚州下的所有订单的 1000 万条记录表，其中的状态列没有索引或索引不良。在后一种情况下，索引可能存在，但由于基数较低，优化器忽略了它。

不理想的服务器配置

如果多个查询显示在 `sending data` 状态，数据库服务器可能配置不佳。具体来说，服务器可能存在以下问题：

- 数据库服务器没有足够的计算容量：磁盘输入/输出、磁盘类型和速度、CPU 或 CPU 数量。

- 服务器急需获得分配的资源，例如 InnoDB 表的 InnoDB 缓冲池或 MyIsam 表的密钥缓冲区。
- 每个线程的内存设置（例如 `sort_buffer`、`read_buffer` 和 `join_buffer`）消耗的 RAM 超出了所需的数量，使物理服务器无法获得内存资源。

操作

一般指南是通过检查性能架构来查找返回大量行的查询。如果启用了不使用索引的记录查询，您还可以检查慢日志的结果。

主题

- 如果性能架构未开启，请打开它
- 检查内存设置
- 检查索引使用的解释计划
- 检查返回的数据量
- 检查并发问题
- 检查您的查询结构

如果性能架构未开启，请打开它

仅在性能架构工具未打开时，性能详情才会报告线程状态。启用性能架构工具后，性能详情会报告等待事件。在调查潜在的性能问题时，性能架构工具可以提供更多洞察和更好的工具。因此，建议您开启性能架构。有关更多信息，请参阅 Aurora MySQL 上性能详情的性能架构概述。

检查内存设置

检查主缓冲池的内存设置。确保这些缓冲池的大小适合工作负载。如果您的数据库使用多个缓冲池实例，请确保它们没有被划分为许多小型缓冲池。线程一次只能使用一个缓冲池。

确保用于每个线程的以下内存设置的大小正确：

- read_buffer
- read_rnd_buffer
- sort_buffer
- join_buffer
- binlog_cache

除非您有特定原因要修改设置，否则请使用原定设置值。

检查索引使用的解释计划

对于 sending data 线程状态中的查询，检查计划以确定是否使用了适当的索引。如果查询没有使用有用的索引，请考虑添加 USE INDEX 或 FORCE INDEX 之类的提示。提示可以大大增加或减少运行查询所需的时间，因此在添加提示之前请小心。

检查返回的数据量

检查正在查询的表以及它们包含的数据量。这些数据都可以存档吗？ 在许多情况下，查询执行时间不佳的原因不是查询计划的结果，而是要处理的数据量。许多开发人员可以非常有效地向数据库添加数据，但在设计和开发阶段，很少会考虑数据集生命周期。

查找在低容量数据库中表现良好但在当前系统中表现不佳的查询。有时，设计特定查询的开发人员可能没有意识到这些查询返回了 350000 行。开发人员可能是在数据集小于生产环境数据集的较小容量环境中开发了查询。

检查并发问题

检查是否同时运行同一类型的多个查询。有些形式的查询在单独运行时会有效运行。但是，如果类似形式的查询一起运行，或者运行量大，它们可能会导致并发问题。通常，这些问题是在数据库使用临时表渲染结果时引起的。限制性事务隔离级别也可能导致并发问题。

如果同时读取和写入表，则数据库可能正在使用锁定。为了帮助确定性能不佳的时期，请通过大规模批处理过程来检查数据库的使用情况。要查看最近的锁定和回滚，请检查 SHOW ENGINE INNODB STATUS 命令的输出。

检查您的查询结构

检查从这些状态捕获的查询是否使用子查询。这种类型的查询通常会导致性能不佳，因为数据库会在内部编译结果，然后将其替换回查询中以渲染数据。这个过程是数据库的额外步骤。在许多情况下，在高度并发的加载条件下，此步骤可能会导致性能不佳。

同时，检查您的查询是否使用了大量 ORDER BY 和 GROUP BY 子句。在此类操作中，数据库通常必须首先在内存中形成整个数据集。然后，它必须以特定的方式对数据集进行排序或分组，然后才能将其返回给客户端。

使用 Amazon DevOps Guru 主动见解优化 Aurora MySQL

DevOps Guru 主动见解可在您的 Aurora MySQL 数据库集群出现已知问题状况之前检测到它们。DevOps Guru 可以执行以下操作：

- 通过对照常见的建议设置交叉检查数据库配置，可以防止许多常见的数据库问题。
- 提醒您注意实例集中的关键问题，如果不加以检查，以后可能会导致更大的问题。
- 提醒您注意新发现的问题。

每项主动见解都包含对问题原因的分析和纠正措施建议。

主题

- InnoDB 历史记录列表长度显著增加
- 数据库正在磁盘上创建临时表

InnoDB 历史记录列表长度显著增加

从 *date* 开始，您的行更改历史记录列表显著增加，最大达到 *db-instance* 上的 *length*。这一增加会影响查询和数据库关闭性能。

主题

- 支持的引擎版本
- 上下文
- 这个问题的可能原因
- 操作
- 相关指标

支持的引擎版本

Aurora MySQL 的所有版本都支持这些见解信息。

上下文

InnoDB 事务系统维护多版本并发控制（MVCC）。修改行时，正在修改的数据的预修改版本将作为撤消记录存储在撤消日志中。每条撤消记录都引用其先前的重做记录，形成一个链接的列表。

InnoDB 历史记录列表是已提交事务的撤消日志的全局列表。当事务不再需要历史记录时，MySQL 使用历史记录列表来清除记录和日志页面。历史记录列表长度是包含历史记录列表中的修改的撤消日志总数。每个日志包含一个或多个修改。如果 InnoDB 历史记录列表长度过大，表明有大量的旧行版本，则查询和数据库关闭会变得更慢。

这个问题的可能原因

历史记录列表较长的典型原因包括以下几点：

- 长时间运行的事务，无论是读取还是写入
- 繁重的写入负载

操作

根据见解的原因，我们建议采取不同的操作。

主题

- 在 InnoDB 历史记录列表减小之前，不要开始任何涉及数据库关闭的操作
- 识别并结束长时间运行的事务
- 使用性能详情确定首要主机和主要用户。

在 InnoDB 历史记录列表减小之前，不要开始任何涉及数据库关闭的操作

由于 InnoDB 历史记录列表较长会减慢数据库关闭速度，因此请在启动涉及数据库关闭的操作之前减小列表大小。这些操作包括主要版本数据库升级。

识别并结束长时间运行的事务

您可以通过查询 `information_schema.innodb_trx` 来找到长时间运行的事务。

> **Note**
>
> 还要确保在只读副本上查找长时间运行的事务。

识别并结束长时间运行的事务

1. 在 SQL 客户端中，运行以下查询：

```
SELECT a.trx_id,
      a.trx_state,
      a.trx_started,
      TIMESTAMPDIFF(SECOND,a.trx_started, now()) as "Seconds Transaction Has Been
 Open",
      a.trx_rows_modified,
```

```
      b.USER,
      b.host,
      b.db,
      b.command,
      b.time,
      b.state
FROM  information_schema.innodb_trx a,
      information_schema.processlist b
WHERE a.trx_mysql_thread_id=b.id
  AND TIMESTAMPDIFF(SECOND,a.trx_started, now()) > 10
ORDER BY trx_started
```

2. 使用存储过程 mysql.rds_kill 结束每个长时间运行的事务。

使用性能详情确定首要主机和主要用户。

优化事务，以便立即提交大量修改后的行。

相关指标

以下指标与此见解相关：

- `trx_rseg_history_len` – 可以在性能详情以及 `INFORMATION_SCHEMA.INNODB_METRICS` 表中查看此计数器指标。有关更多信息，请参阅 MySQL 文档中的 InnoDB INFORMATION_SCHEMA metrics table。
- `RollbackSegmentHistoryListLength` – 此 Amazon CloudWatch 指标衡量记录已提交事务（带有删除标记的记录）的撤销日志。这些记录安排为由 InnoDB 清除操作进行处理。指标 `trx_rseg_history_len` 具有与 `RollbackSegmentHistoryListLength` 相同的值。
- `PurgeBoundary` – 允许进行 InnoDB 清除的截止事务编号。如果此 CloudWatch 指标在很长一段时间内没有进展，这很好地表明 InnoDB 清除被长时间运行的事务阻止了。要进行调查，请检查 Aurora MySQL 数据库集群上的活跃事务。此指标仅适用于 Aurora MySQL 版本 2.11 及更高版本以及版本 3.08 及更高版本。
- `PurgeFinishedPoint` – 执行 InnoDB 清除的截止事务编号。此 CloudWatch 指标有助于您检查 InnoDB 清除的进展速度。此指标仅适用于 Aurora MySQL 版本 2.11 及更高版本以及版本 3.08 及更高版本。
- `TransactionAgeMaximum` – 最早的活动运行事务的期限。此 CloudWatch 指标仅适用于 Aurora MySQL 版本 3.08 及更高版本。
- `TruncateFinishedPoint` – 执行撤消截断的截止事务编号。此 CloudWatch 指标仅适用于 Aurora MySQL 版本 2.11 及更高版本以及版本 3.08 及更高版本。

有关 CloudWatch 指标的更多信息，请参阅 Amazon Aurora 的实例级指标。

数据库正在磁盘上创建临时表

您最近的磁盘上临时表使用量显著增加，高达 *percentage*。数据库每秒创建大约 *number* 个临时表。这可能会影响性能并增加 *db-instance* 上的磁盘操作。

主题

- 支持的引擎版本
- 上下文
- 这个问题的可能原因
- 操作
- 相关指标

支持的引擎版本

Aurora MySQL 的所有版本都支持这些见解信息。

上下文

有时，MySQL 服务器在处理查询时需要创建内部临时表。Aurora MySQL 可以在内存中保留一个内部临时表，此临时表可在此由 TempTable 或 MEMORY 存储引擎进行处理，也可以由 InnoDB 存储在磁盘上。有关更多信息，请参阅《MySQL 参考手册》中的 MySQL 中的内部临时表使用。

这个问题的可能原因

磁盘上临时表的增加表明使用了复杂的查询。如果配置的内存不足以在内存中存储临时表，则 Aurora MySQL 会在磁盘上创建表。这可能会影响性能并增加磁盘操作。

操作

根据见解的原因，我们建议采取不同的操作。

- 对于 Aurora MySQL 版本 3，我们建议您使用 TempTable 存储引擎。
- 通过仅选择必要的列来优化查询，以返回更少的数据。

  如果在启用所有 `statement` 分析并对其定时的情况下开启性能模式，则可以查询 `SYS.statements_with_temp_tables` 以检索使用临时表的查询列表。有关更多信息，请参阅 MySQL 文档中的使用 sys 模式的先决条件。

- 考虑对参与排序和分组操作的列编制索引。
- 重新编写您的查询以避免使用 BLOB 和 TEXT 列。这些列始终使用磁盘。
- 调整以下数据库参数：tmp_table_size 和 max_heap_table_size。

  这些参数的原定设置值为 16MiB。对内存中的临时表使用 MEMORY 存储引擎时，其最大大小由 tmp_table_size 或 max_heap_table_size 值定义，以较小者为准。当达到此最大大小时，MySQL 会自动将内存中的内部临时表转换为 InnoDB 磁盘上的内部临时表。有关更多信息，请参阅在 Amazon RDS for MySQL 和 Amazon Aurora MySQL 上使用 TempTable 存储引擎。

  > **Note**
  >
  > 使用 CREATE TABLE 显式创建 MEMORY 表时，只有 max_heap_table_size 变量决定表可以增长多大。也不会转换为磁盘上的格式。

相关指标

以下性能详情指标与此见解相关：

- Created_tmp_disk_tables
- Created_tmp_tables

有关更多信息，请参阅 MySQL 文档中的 Created_tmp_disk_tables。

Amazon Aurora MySQL 的并行查询

本主题描述了 Amazon Aurora MySQL 兼容版的并行查询性能优化。该功能在某些数据密集型查询中使用特殊处理路径，从而利用 Aurora 共享存储架构。并行查询非常适合以下 Aurora MySQL 数据库集群：具有包含数百万行的表以及需要数分钟或数小时才能完成的分析查询。

主题

- Aurora MySQL 的并行查询概述
- 在 Aurora MySQL 中创建并行查询数据库集群
- 在 Aurora MySQL 中开启和关闭并行查询
- 在 Aurora MySQL 中优化并行查询
- 验证哪些语句使用 Aurora MySQL 的并行查询
- 监控 Aurora MySQL 的并行查询
- Aurora MySQL 中用于并行查询的 SQL 构造

Aurora MySQL 的并行查询概述

Aurora MySQL 并行查询是一种优化功能，它并行处理在处理数据密集型查询时涉及的一些 I/O 和计算。并行处理的工作包括从存储中检索行，提取列值以及确定哪些行与 `WHERE` 子句和联接子句中的条件匹配。这种数据密集型工作将委派（在数据库优化术语中为向下推送）给 Aurora 分布式存储层中的多个节点。如果不使用并行查询，每个查询将所有扫描的数据传输到 Aurora MySQL 集群中的单个节点（头节点），并在此处执行所有查询处理。

> **Tip**
>
> PostgreSQL 数据库引擎还有一个称为“并行查询”的功能。该功能与 Aurora 并行查询无关。

如果开启了并行查询功能，Aurora MySQL 引擎将自动确定查询何时可以从中受益，而无需进行 SQL 更改（如提示或表属性）。在以下章节中，您可以找到何时将并行查询应用于查询的说明。您还可以了解如何确保在提供最大好处时应用并行查询。

> **Note**
>
> 并行查询优化为需要数分钟或数小时才能完成的长时间运行的查询提供最大优势。Aurora MySQL 通常不会为低开销查询运行并行查询优化。如果另一种优化技术更有意义（如查询缓

存、缓冲池缓存或索引查找），它通常也不会执行并行查询优化。如果发现在需要时未使用并行查询，请参阅 验证哪些语句使用 Aurora MySQL 的并行查询。

主题

- 优势
- 架构
- 先决条件
- 限制
- 并行查询导致的 I/O 成本

优势

使用并行查询，您可以对 Aurora MySQL 表运行数据密集型分析查询。在很多情况下，与传统的查询处理分工相比，性能提高了一个数量级。

并行查询的好处包括：

- 由于跨多个存储节点并行处理物理读取请求，提高了 I/O 性能。
- 降低了网络流量。Aurora 不会将存储节点中的整个数据页面传输到头节点并随后筛选掉不需要的行和列。相反，Aurora 传输仅包含结果集所需的列值的紧凑元组。
- 由于向下推送 WHERE 子句的函数处理、行筛选和列投影，减少了头节点上的 CPU 使用率。
- 减轻了缓冲池上的内存压力。并行查询处理的页面不会添加到缓冲池中。此方法可降低数据密集型扫描从缓冲池中逐出经常使用的数据的可能性。
- 通过使在现有的数据上执行长时间运行的分析查询变得切实可行，可能会在提取、转换和加载 (ETL) 管道中减少重复的数据。

架构

并行查询功能使用 Aurora MySQL 的主要架构准则：将数据库引擎与存储子系统分离，并简化通信协议以减少网络流量。Aurora MySQL 使用这些技术加快写入密集型操作（例如重做日志处理）。并行查询将相同的准则应用于读取操作。

> **Note**
>
> Aurora MySQL 并行查询的架构与其他数据库系统中名称类似的功能架构不同。Aurora MySQL 并行查询不涉及对称多处理 (SMP)，因此不依赖于数据库服务器的 CPU 容量。并行处理是在存储层中发生的，与作为查询协调器的 Aurora MySQL 服务器无关。

默认情况下，如果没有并行查询，Aurora 查询处理涉及将原始数据传输到 Aurora 集群中的单个节点（头节点）。之后，Aurora 会针对该单个节点上单个线程中的该查询执行所有进一步的处理。通过使用并行查询，该 I/O 密集型和 CPU 密集型工作的绝大部分将委派给存储层中的节点。仅将结果集的紧凑行传回到头节点，已筛选行并提取和转换了列值。性能优势来自于网络流量减少、头节点上的 CPU 使用率下降以及跨存储节点并行处理 I/O。并行 I/O、筛选和投影数量与运行查询的 Aurora 集群中的数据库实例数无关。

先决条件

要使用并行查询的所有特征，需要运行版本 2.09 或更高版本的 Aurora MySQL 数据库集群。如果您已有要与并行查询一起使用的集群，可以将其升级到兼容版本并在之后开启并行查询。在这种情况下，请确保遵循 并行查询的升级注意事项中的升级过程，因为这些较新版本中的配置设置名称和默认值不同。

集群中的数据库实例必须使用 `db.r*` 实例类。

确保为集群启用了哈希联接优化。要了解如何操作，请参阅为并行查询集群开启哈希联接。

要自定义参数（如 `aurora_parallel_query` 和 `aurora_disable_hash_join`），您必须具有与集群一起使用的自定义参数组。您可以使用数据库参数组为每个数据库实例单独指定这些参数。但是，我们建议您在数据库集群参数组中指定它们。这样，集群中的所有数据库实例都会继承这些参数的相同设置。

限制

以下限制适用于并行查询功能：

- Aurora I/O-Optimized 数据库集群存储配置不支持并行查询。
- 您不能将并行查询与 db.t2 或 db.t3 实例类一起使用。即使您使用 `aurora_pq_force` 会话变量来请求并行查询，此限制也适用。
- 并行查询不适用于使用 `COMPRESSED` 或 `REDUNDANT` 行格式的表。对于计划与并行查询结合使用的表，请使用 `COMPACT` 或 `DYNAMIC` 行格式。

- Aurora 使用基于成本的算法来确定是否对每个 SQL 语句使用并行查询机制。在语句中使用某些 SQL 结构可以防止并行查询，或使该语句不太可能执行并行查询。有关 SQL 结构与并行查询的兼容性的信息，请参阅 Aurora MySQL 中用于并行查询的 SQL 构造。
- 每个 Aurora 数据库实例每次只能运行一定数量的并行查询会话。如果查询具有多个使用并行查询的部分（例如，子查询、联接或 UNION 运算符），这些阶段将按顺序运行。在任何时候，该语句仅计为一个并行查询会话。您可以使用并行查询状态变量监控活动会话数。您可以查询 `Aurora_pq_max_concurrent_requests` 状态变量以检查给定数据库实例的并发会话数限制。
- 并行查询适用于 Aurora 支持的所有 AWS 区域。对于大多数 AWS 区域，使用并行查询所需的最低 Aurora MySQL 版本为 2.09。
- 并行查询旨在提高数据密集型查询的性能。它不是为轻量级查询而设计的。
- 我们建议您为 SELECT 语句使用读取器节点，尤其是数据密集型语句。

并行查询导致的 I/O 成本

如果您的 Aurora MySQL 集群使用并行查询，您可能会看到 `VolumeReadIOPS` 值出现增长。并行查询不使用缓冲池。因此，尽管查询速度很快，但这种优化的处理可能会导致读取操作和相关费用的增加。

查询的并行查询 I/O 成本在存储层计量，开启并行查询后，该成本将相同或更高。好处是提高了查询性能。并行查询可能导致 I/O 成本更高的原因有两个：

- 即使表中的某些数据位于缓冲池中，并行查询也要求在存储层扫描所有数据，这会产生 I/O 成本。
- 运行并行查询不会预热缓冲池。因此，连续运行同一个并行查询会产生完全 I/O 成本。

在 Aurora MySQL 中创建并行查询数据库集群

要创建具有并行查询的 Aurora MySQL 集群，在其中添加新实例或执行其他管理操作，您可以使用与其他 Aurora MySQL 集群相同的AWS Management Console和 AWS CLI 方法。您可以创建新的集群以使用并行查询。也可以通过从 MySQL 兼容的 Aurora 数据库集群的快照还原，创建一个数据库集群以使用并行查询。如果不熟悉创建新的 Aurora MySQL 集群的过程，您可以在创建 Amazon Aurora 数据库集群中找到背景信息和先决条件。

在选择 Aurora MySQL 引擎版本时，建议您选择可用的最新版本。目前，所有可用的 Aurora MySQL 版本均支持并行查询。如果使用最新版本，则可以更灵活地开启和关闭并行查询，或者将并行查询与现有集群结合使用。

无论是创建新集群还是从快照还原，您都可以使用与其他 Aurora MySQL 集群相同的方法添加新的数据库实例。

可以使用 Amazon RDS 控制台或 AWS CLI 创建并行查询集群。

目录

- 使用控制台创建并行查询集群
- 使用 CLI 创建并行查询集群

使用控制台创建并行查询集群

您可以使用控制台创建新的并行查询集群，如下所述。

使用 AWS Management Console 创建并行查询集群

1. 按照AWS Management Console中的常规 创建 Amazon Aurora 数据库集群过程进行操作。
2. 对于引擎类型，选择“Aurora MySQL”。
3. 对于其他配置，请选择为数据库集群参数组创建的参数组。Aurora MySQL 2.09 和更高版本需要使用这样的自定义参数组。在数据库集群参数组中，指定参数设置 `aurora_parallel_query=ON` 和 `aurora_disable_hash_join=OFF`。这样做会为集群开启并行查询，并开启与并行查询结合使用的哈希联接优化。

验证新集群是否可以使用并行查询

1. 使用上述方法创建集群。
2. （对于 Aurora MySQL 版本 2 或 3）检查 `aurora_parallel_query` 配置设置是否为 true。

```
mysql> select @@aurora_parallel_query;
+--------------------------+
| @@aurora_parallel_query |
+--------------------------+
|                        1 |
+--------------------------+
```

3. （对于 Aurora MySQL 版本 2）检查 `aurora_disable_hash_join` 设置是否为 false。

```
mysql> select @@aurora_disable_hash_join;
+---------------------------+
```

```
| @@aurora_disable_hash_join |
+----------------------------+
|                          0 |
+----------------------------+
```

4. 对于一些大型表和数据密集型查询，请检查查询计划以确认某些查询正在使用并行查询优化。为此，请按照验证哪些语句使用 Aurora MySQL 的并行查询中的过程操作。

使用 CLI 创建并行查询集群

您可以使用 CLI 创建新的并行查询集群，如下所述。

使用 AWS CLI 创建并行查询集群

1. （可选）检查哪些 Aurora MySQL 版本与并行查询集群兼容。为此，请使用 `describe-db-engine-versions` 命令并检查 `SupportsParallelQuery` 字段的值。有关示例，请参阅检查并行查询的 Aurora MySQL 版本兼容性。
2. （可选）使用设置 `aurora_parallel_query=ON` 和 `aurora_disable_hash_join=OFF` 创建自定义数据库集群参数组。使用如下命令。

```
aws rds create-db-cluster-parameter-group --db-parameter-group-family aurora-
mysql8.0 --db-cluster-parameter-group-name pq-enabled-80-compatible
aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-name pq-
enabled-80-compatible \
  --parameters
 ParameterName=aurora_parallel_query,ParameterValue=ON,ApplyMethod=pending-reboot
aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-name pq-
enabled-80-compatible \
  --parameters
 ParameterName=aurora_disable_hash_join,ParameterValue=OFF,ApplyMethod=pending-
reboot
```

   如果执行此步骤，请在后续 `--db-cluster-parameter-group-name` *`my_cluster_parameter_group`* 语句中指定 `create-db-cluster` 选项。替换您自己的参数组的名称。如果省略此步骤，则创建参数组并稍后将其与集群关联，如 在 Aurora MySQL 中开启和关闭并行查询中所述。
3. 按照AWS CLI中的常规 创建 Amazon Aurora 数据库集群过程进行操作。
4. 指定以下选项集：

- 对于 `--engine` 选项，请使用 `aurora-mysql`。这些值生成的并行查询集群与 MySQL 5.7 或 8.0 兼容。
- 对于 `--db-cluster-parameter-group-name` 选项，请指定您创建并指定参数值 `aurora_parallel_query=ON` 的数据库集群参数组的名称。如果省略此选项，则可以使用默认参数组创建集群，然后对其进行修改以使用此类自定义参数组。
- 对于 `--engine-version` 选项，请使用与并行查询兼容的 Aurora MySQL 版本。如有必要，请使用 在 Aurora MySQL 中优化并行查询中的过程获取版本列表。

  以下代码示例显示了操作方法。用您自己的值替换每个环境变量，如 *$CLUSTER_ID*。此示例还指定了生成主用户密码并在 Secrets Manager 中对其进行管理的 `--manage-master-user-password` 选项。有关更多信息，请参阅 使用 Amazon Aurora 和 AWS Secrets Manager 管理密码。或者，您可以使用 `--master-password` 选项自行指定和管理密码。

```
aws rds create-db-cluster --db-cluster-identifier $CLUSTER_ID \
  --engine aurora-mysql --engine-version 8.0.mysql_aurora.3.04.1 \
  --master-username $MASTER_USER_ID --manage-master-user-password \
  --db-cluster-parameter-group-name $CUSTOM_CLUSTER_PARAM_GROUP

aws rds create-db-instance --db-instance-identifier ${INSTANCE_ID}-1 \
  --engine same_value_as_in_create_cluster_command \
  --db-cluster-identifier $CLUSTER_ID --db-instance-class $INSTANCE_CLASS
```

5. 验证您创建或还原的集群是否具有可用的并行查询功能。

   检查 `aurora_parallel_query` 配置设置是否存在。如果此设置的值为 1，则可以使用并行查询。如果此设置的值为 0，请先将其设置为 1，然后才能使用并行查询。无论使用哪种方式，集群都能够执行并行查询。

```
mysql> select @@aurora_parallel_query;
+------------------------+
| @@aurora_parallel_query|
+------------------------+
|                      1 |
+------------------------+
```

使用 AWS CLI将快照还原到并行查询集群

1. 检查哪些 Aurora MySQL 版本与并行查询集群兼容。为此，请使用 `describe-db-engine-versions` 命令并检查 `SupportsParallelQuery` 字段的值。有关示例，请参阅检查并行查询的 Aurora MySQL 版本兼容性。确定要用于还原的集群的版本。
2. 找到与 Aurora MySQL 兼容的集群快照。
3. 按照AWS CLI中的常规 从数据库集群快照还原过程进行操作。

```
aws rds restore-db-cluster-from-snapshot \
  --db-cluster-identifier mynewdbcluster \
  --snapshot-identifier mydbclustersnapshot \
  --engine aurora-mysql
```

4. 验证您创建或还原的集群是否具有可用的并行查询功能。使用与 使用 CLI 创建并行查询集群中相同的验证过程。

在 Aurora MySQL 中开启和关闭并行查询

如果开启了并行查询，Aurora MySQL 确定是否在运行时为每个查询使用该功能。对于联接、联合和子查询等，Aurora MySQL 确定每个查询块是否在运行时使用并行查询。有关详细信息，请参阅 验证哪些语句使用 Aurora MySQL 的并行查询 和 Aurora MySQL 中用于并行查询的 SQL 构造。

您可以使用 aurora_parallel_query 选项，在数据库实例的全局和会话级别动态开启和关闭并行查询。默认情况下，您可以更改数据库集群组中的 `aurora_parallel_query` 设置，以启用或关闭并行查询。

```
mysql> select @@aurora_parallel_query;
+-------------------------+
| @@aurora_parallel_query|
+-------------------------+
|                       1 |
+-------------------------+
```

要在会话级别切换 `aurora_parallel_query` 参数，请使用标准方法更改客户端配置设置。例如，可以通过 `mysql` 命令行或在 JDBC 或 ODBC 应用程序中执行此操作。标准 MySQL 客户端上的命令是 `set session aurora_parallel_query = {'ON'/'OFF'}`。您还可以将会话级参数添加到 JDBC 配置或应用程序代码中，以动态开启或关闭并行查询。

您可以为特定数据库实例或整个集群永久更改 aurora_parallel_query 参数的设置。如果您在数据库参数组中指定参数值，则该值仅适用于集群中的特定数据库实例。如果在数据库集群参数组中指定参数值，则集群中的所有数据库实例都将继承相同的设置。要切换 aurora_parallel_query 参数，请使用处理参数组的方法，如Amazon Aurora 的参数组中所述。按照以下步骤进行操作：

1. 创建自定义集群参数组（推荐）或自定义数据库参数组。
2. 在此参数组中，将 parallel_query 更新为所需的值。
3. 根据您创建的是数据库集群参数组还是数据库参数组，将参数组附加到 Aurora 集群或计划使用并行查询功能的特定数据库实例。

> **Tip**
>
> 因为 aurora_parallel_query 是一个动态参数，所以在更改此设置后不需要重启集群。但是，在切换选项之前使用并行查询的任何连接都将继续执行此操作，直到连接关闭或实例重启。

您可以使用 ModifyDBClusterParameterGroup 或 ModifyDBParameterGroup API 操作或 AWS Management Console修改并行查询参数。

可以为并行查询集群开启哈希联接，使用 Amazon RDS 控制台或 AWS CLI 开启和关闭并行查询，并覆盖并行查询优化器。

目录

- 为并行查询集群开启哈希联接
- 使用控制台开启和关闭并行查询
- 使用 CLI 开启和关闭并行查询
- 覆盖并行查询优化程序

为并行查询集群开启哈希联接

并行查询通常用于受益于哈希联接优化的各种资源密集型查询。因此，确保为计划使用并行查询的集群开启哈希联接非常有用。有关如何有效使用哈希联接的信息，请参阅 使用哈希联接优化大型 Aurora MySQL 联接查询。

使用控制台开启和关闭并行查询

通过使用参数组，可以在数据库实例级别或数据库集群级别开启或关闭并行查询。

要使用 AWS Management Console 开启或关闭数据库集群的并行查询

1. 创建一个自定义参数组，如Amazon Aurora 的参数组中所述。
2. 将 aurora_parallel_query 更新为 1（开启）或 0（关闭）。对于可以使用并行查询功能的集群，将原定设置关闭 aurora_parallel_query。
3. 如果使用自定义集群参数组，请将其附加到计划使用并行查询功能的 Aurora 数据库集群。如果您使用自定义数据库参数组，请将其附加到集群中的一个或多个数据库实例。我们建议使用集群参数组。这样做可确保集群中的所有数据库实例对并行查询和关联功能（如哈希联接）具有相同的设置。

使用 CLI 开启和关闭并行查询

您可以使用 `modify-db-cluster-parameter-group` 或 `modify-db-parameter-group` 命令修改并行查询参数。根据您是通过数据库集群参数组还是通过数据库参数组指定 `aurora_parallel_query` 的值，选择相应的命令。

要使用 CLI 开启或关闭数据库集群的并行查询

- 使用 `modify-db-cluster-parameter-group` 命令修改并行查询参数。使用如下命令。用适当的名称替换您自己的自定义参数组。用 `ON` 或 `OFF`替换 `ParameterValue` 选项的 `--parameters` 部分。

```
$ aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-
name cluster_param_group_name \
  --parameters
 ParameterName=aurora_parallel_query,ParameterValue=ON,ApplyMethod=pending-reboot
{
    "DBClusterParameterGroupName": "cluster_param_group_name"
}

aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-
name cluster_param_group_name \
  --parameters ParameterName=aurora_pq,ParameterValue=ON,ApplyMethod=pending-reboot
```

您还可以在会话级别开启或关闭并行查询，例如，通过 mysql 命令行或在 JDBC 或 ODBC 应用程序中。为此，请使用标准方法更改客户端配置设置。例如，对于 Aurora MySQL，标准 MySQL 客户端上的命令是 set session aurora_parallel_query = {'ON'/'OFF'}。

您还可以将会话级参数添加到 JDBC 配置或应用程序代码中，以动态开启或关闭并行查询。

覆盖并行查询优化程序

您可以使用 aurora_pq_force 会话变量覆盖并行查询优化程序并为每个查询请求并行查询。我们建议您仅出于测试目的这样做。以下示例显示如何在会话中使用 aurora_pq_force。

```
set SESSION aurora_parallel_query = ON;
set SESSION aurora_pq_force = ON;
```

要关闭覆盖，请执行以下操作：

```
set SESSION aurora_pq_force = OFF;
```

在 Aurora MySQL 中优化并行查询

要优化数据库集群来进行并行查询，请考虑哪些数据库集群将从并行查询中受益，以及是否要升级来进行并行查询。然后，调整工作负载并创建用于并行查询的架构对象。

目录

- 规划并行查询集群
  - 检查并行查询的 Aurora MySQL 版本兼容性
- 并行查询的升级注意事项
  - 将并行查询集群升级到 Aurora MySQL 版本 3
  - 升级到 Aurora MySQL 2.09 及更高版本
- 并行查询的性能优化
- 创建架构对象以利用并行查询

规划并行查询集群

规划开启并行查询的数据库集群需要做出一些选择。其中包括执行设置步骤（创建或还原完整 Aurora MySQL 集群），以及确定在整个数据库集群中开启并行查询的范围。

作为规划的一部分，考虑以下事项：

- 如果您使用与 MySQL 5.7 兼容的 Aurora MySQL，则必须创建预置集群。然后，使用 `aurora_parallel_query` 参数开启并行查询。

  如果已有 Aurora MySQL 集群，则无需创建新集群即可使用并行查询。您可以将集群或集群中的特定数据库实例与开启了 `aurora_parallel_query` 参数的参数组相关联。这样，您可以减少设置要与并行查询结合使用的相关数据的时间和精力。
- 规划需要重新组织的任何大型表，以便在访问它们时可以使用并行查询。您可能需要创建一些大型表的新版本，其中并行查询非常有用。例如，您可能需要删除全文搜索索引。有关详细信息，请参阅创建架构对象以利用并行查询。

检查并行查询的 Aurora MySQL 版本兼容性

要检查哪些 Aurora MySQL 版本与并行查询集群兼容，请使用 `describe-db-engine-versions` AWS CLI CLI 命令并检查 `SupportsParallelQuery` 字段的值。以下代码示例说明了如何检查哪些组合适用于指定 AWS 区域中的并行查询集群。确保在单行上指定完整的 `--query` 参数字符串。

```
aws rds describe-db-engine-versions --region us-east-1 --engine aurora-mysql \
--query '*[]|[?SupportsParallelQuery == `true`].[EngineVersion]' --output text
```

上述命令生成类似于以下内容的输出。输出可能因指定AWS区域中可用的 Aurora MySQL 版本而异。

```
5.7.mysql_aurora.2.11.1
5.7.mysql_aurora.2.11.2
5.7.mysql_aurora.2.11.3
5.7.mysql_aurora.2.11.4
5.7.mysql_aurora.2.11.5
5.7.mysql_aurora.2.11.6
5.7.mysql_aurora.2.12.0
5.7.mysql_aurora.2.12.1
5.7.mysql_aurora.2.12.2
5.7.mysql_aurora.2.12.3
5.7.mysql_aurora.2.12.4
8.0.mysql_aurora.3.04.0
8.0.mysql_aurora.3.04.1
8.0.mysql_aurora.3.04.2
8.0.mysql_aurora.3.04.3
8.0.mysql_aurora.3.05.2
8.0.mysql_aurora.3.06.0
8.0.mysql_aurora.3.06.1
8.0.mysql_aurora.3.07.0
```

```
8.0.mysql_aurora.3.07.1
```

开始对集群使用并行查询后，可以监视性能并消除使用并行查询的障碍。有关这些说明，请参阅并行查询的性能优化。

并行查询的升级注意事项

根据升级并行查询集群时的原始版本和目标版本，您可能会发现并行查询可以优化的查询类型的增强功能。您可能还会发现不需要为并行查询指定特殊的引擎模式参数。以下各节介绍了升级已开启并行查询的集群时的注意事项。

将并行查询集群升级到 Aurora MySQL 版本 3

从 Aurora MySQL 版本 3 开始，一些 SQL 语句、子句和数据类型具有新的或改进的并行查询支持。从版本 3 之前的版本升级时，请检查其他查询是否可以从并行查询优化中受益。有关这些并行查询增强的信息，请参阅 列数据类型、分区表 和 聚合函数、GROUP BY 子句和 HAVING 子句。

如果您要从 Aurora MySQL 2.08 或更低版本升级并行查询集群，还可以了解开启并行查询的方式的变化。为此，请阅读 升级到 Aurora MySQL 2.09 及更高版本。

原定设置情况下，哈希联接优化在 Aurora MySQL 版本 3 中处于开启状态。早期版本中的 `aurora_disable_hash_join` 配置选项未使用。

升级到 Aurora MySQL 2.09 及更高版本

在 Aurora MySQL 版本 2.09 及更高版本中，并行查询适用于预调配的集群，但不需要 `parallelquery` 引擎模式参数。因此，您无需创建新集群或从现有快照还原，即可将并行查询与这些版本结合使用。您可以使用 升级 Aurora MySQL 数据库集群的次要版本或补丁程序级别中介绍的升级过程将集群升级到此类版本。您可以升级较旧的集群，无论它是并行查询集群还是预置集群。要减少引擎版本菜单中的选项数，可以选择显示支持并行查询功能的版本来筛选该菜单中的条目。然后，选择 Aurora MySQL 2.09 或更高版本。

将较早的并行查询集群升级到 Aurora MySQL 2.09 或更高版本后，您可以在升级后的集群中开启并行查询。默认情况下，在这些版本中，并行查询处于关闭状态，并且启用该查询的过程不同。预设情况下，哈希联接优化也处于关闭状态，必须单独开启。因此，请确保在升级后再次开启这些设置。有关此操作的说明，请参阅 在 Aurora MySQL 中开启和关闭并行查询和 为并行查询集群开启哈希联接。

特别是，您可以通过使用配置参数 `aurora_parallel_query=ON` 和 `aurora_disable_hash_join=OFF` 而不是 `aurora_pq_supported` 和 `aurora_pq` 来开启并行查询。`aurora_pq_supported` 和 `aurora_pq` 参数在较新 Aurora MySQL 版本中已弃用。

在升级的集群中，EngineMode 属性的值是 provisioned 而不是 parallelquery。要检查并行查询是否可用于指定的引擎版本，现在您检查 SupportsParallelQuery describe-db-engine-versions 命令输出中 AWS CLI 字段的值。在早期 Aurora MySQL 版本中，您已检查 parallelquery 列表中 SupportedEngineModes 是否存在。

升级到 Aurora MySQL 版本 2.09 或更高版本后，您可以利用以下功能。这些功能不适用于运行较旧 Aurora MySQL 版本的并行查询集群。

- Performance Insights。有关更多信息，请参阅“在 Amazon Aurora 上使用性能详情监控数据库负载”。
- 正在回溯。有关更多信息，请参阅“回溯 Aurora 数据库集群”。
- 停止并启动集群。有关更多信息，请参阅“停止和启动 Amazon Aurora 数据库集群”。

并行查询的性能优化

要管理并行查询工作负载的性能，请确保将并行查询用于该优化最有用的查询。

为此，您可以执行以下操作：

- 确保您的最大表与并行查询兼容。您可以更改表属性或重新创建一些表，以便对这些表的查询可以利用并行查询优化。要了解如何操作，请参阅创建架构对象以利用并行查询。
- 监控哪些查询使用并行查询。要了解如何操作，请参阅监控 Aurora MySQL 的并行查询。
- 验证并行查询正用于数据最密集且长时间运行的查询（工作负载具有适当的并发级别）。要了解如何操作，请参阅验证哪些语句使用 Aurora MySQL 的并行查询。
- 优化 SQL 代码以开启并行查询，以应用于您期望的查询。要了解如何操作，请参阅Aurora MySQL 中用于并行查询的 SQL 构造。

创建架构对象以利用并行查询

在创建或修改您计划用于并行查询的表之前，请确保自行熟悉 先决条件和 限制中描述的要求。

由于并行查询要求表使用 ROW_FORMAT=Compact 或 ROW_FORMAT=Dynamic 设置，请检查 Aurora 配置设置以了解对 INNODB_FILE_FORMAT 配置选项的任何更改。发出 SHOW TABLE STATUS 语句以确认数据库中的所有表的行格式。

在更改架构以开启并行查询来处理更多表之前，请确保进行测试。测试应确认并行查询是否会导致这些表的性能净增加。此外，还要确保并行查询的架构要求与您的目标相符。

例如，在从 ROW_FORMAT=Compressed 切换到 ROW_FORMAT=Compact 或 ROW_FORMAT=Dynamic 之前，请针对原始表和新表测试工作负载的性能。此外，还要考虑其他潜在影响，例如，数据量增加。

验证哪些语句使用 Aurora MySQL 的并行查询

在典型操作中，您无需执行任何特殊操作即可利用并行查询。在查询满足并行查询的基本要求后，查询优化程序自动确定是否在每个特定查询中使用并行查询。

如果您在开发或测试环境中运行试验，您可能会发现未使用并行查询，因为您的表的行数或总数据量太少。表的数据也可能完全位于缓冲池中，尤其是最近创建以执行试验的表。

在监控或优化集群性能时，请确保确定是否在相应的上下文中使用并行查询。您可以调整数据库架构、设置、SQL 查询甚至集群拓扑和应用程序连接设置以利用该功能。

要检查查询是否使用并行查询，请运行 EXPLAIN 语句以检查查询计划（也称为“解释计划”）。有关 SQL 语句、子句和表达式如何影响并行查询的 EXPLAIN 输出的示例，请参阅Aurora MySQL 中用于并行查询的 SQL 构造。

以下示例说明了传统查询计划和并行查询计划之间的区别。此解释计划来自 TPC-H 基准中的查询 3。本节中的很多示例查询使用 TPC-H 数据集中的表。您可以从 TPC-H 网站获取生成示例数据的表定义、查询以及 dbgen 程序。

```
EXPLAIN SELECT l_orderkey,
  sum(l_extendedprice * (1 - l_discount)) AS revenue,
  o_orderdate,
  o_shippriority
FROM customer,
  orders,
  lineitem
WHERE c_mktsegment = 'AUTOMOBILE'
AND c_custkey = o_custkey
AND l_orderkey = o_orderkey
AND o_orderdate < date '1995-03-13'
AND l_shipdate > date '1995-03-13'
GROUP BY l_orderkey,
  o_orderdate,
  o_shippriority
ORDER BY revenue DESC,
  o_orderdate LIMIT 10;
```

默认情况下，查询可能具有如下所示的计划。如果您没有看到查询计划中使用的哈希联接，请确保首先开启了优化。

```
+----+-------------+----------+------------+------+---------------+------+---------
+------+----------+----------+----------------------------------------------------+
| id | select_type | table    | partitions | type | possible_keys | key  | key_len |
 ref  | rows     | filtered | Extra                                              |
+----+-------------+----------+------------+------+---------------+------+---------
+------+----------+----------+----------------------------------------------------+
|  1 | SIMPLE      | customer | NULL       | ALL  | NULL          | NULL | NULL    |
 NULL |  1480234 |    10.00 | Using where; Using temporary; Using filesort       |
|  1 | SIMPLE      | orders   | NULL       | ALL  | NULL          | NULL | NULL    |
 NULL | 14875240 |     3.33 | Using where; Using join buffer (Block Nested Loop) |
|  1 | SIMPLE      | lineitem | NULL       | ALL  | NULL          | NULL | NULL    |
 NULL | 59270573 |     3.33 | Using where; Using join buffer (Block Nested Loop) |
+----+-------------+----------+------------+------+---------------+------+---------
+------+----------+----------+----------------------------------------------------+
```

对于 Aurora MySQL 版本 3，您可以通过发出以下语句在会话级别开启哈希联接。

```
SET optimizer_switch='block_nested_loop=on';
```

对于 Aurora MySQL 版本 2.09 及更高版本，您可以将 aurora_disable_hash_join 数据库参数或数据库集群参数设置为 0（关闭）。关闭 aurora_disable_hash_join 会将 optimizer_switch 的值设置为 hash_join=on。

开启哈希连接后，尝试再次运行 EXPLAIN 语句。有关如何有效使用哈希联接的信息，请参阅 使用哈希联接优化大型 Aurora MySQL 联接查询。

在开启哈希联接但关闭并行查询的情况下，查询可能具有如下所示的计划，该计划使用哈希联接而不是并行查询。

```
+----+-------------+----------+...+-----------
+-------------------------------------------------------------------+
| id | select_type | table    |...| rows      | Extra
                          |
+----+-------------+----------+...+-----------
+-------------------------------------------------------------------+
|  1 | SIMPLE      | customer |...|   5798330 | Using where; Using index; Using
 temporary; Using filesort       |
|  1 | SIMPLE      | orders   |...| 154545408 | Using where; Using join buffer (Hash
 Join Outer table orders)   |
|  1 | SIMPLE      | lineitem |...| 606119300 | Using where; Using join buffer (Hash
 Join Outer table lineitem) |
```

```
+----+-------------+----------+...+-----------
+-------------------------------------------------------------------+
```

在开启并行查询后，该查询计划中的两个步骤可以使用并行查询优化，如 EXPLAIN 输出中的 Extra 列所示。这些步骤的 I/O 密集型和 CPU 密集型处理将向下推送到存储层。

```
+----+...
+-------------------------------------------------------------------------------------------------------------
+
| id |...| Extra
                                                    |
+----+...
+-------------------------------------------------------------------------------------------------------------
+
|  1 |...| Using where; Using index; Using temporary; Using filesort
                                                    |
|  1 |...| Using where; Using join buffer (Hash Join Outer table orders); Using
 parallel query (4 columns, 1 filters, 1 exprs; 0 extra)   |
|  1 |...| Using where; Using join buffer (Hash Join Outer table lineitem); Using
 parallel query (4 columns, 1 filters, 1 exprs; 0 extra) |
+----+...
+-------------------------------------------------------------------------------------------------------------
+
```

有关如何解释并行查询的 EXPLAIN 输出以及并行查询可以应用到的 SQL 语句部分的信息，请参阅Aurora MySQL 中用于并行查询的 SQL 构造。

以下示例输出显示在具有冷缓冲池的 db.r4.2xlarge 实例上运行上述查询的结果。在使用并行查询时，查询运行速度要快得多。

> **Note**
>
> 由于计时取决于很多环境因素，因此，您的结果可能会有所不同。请始终执行您自己的性能测试，以便在您自己的环境、工作负载等条件下确认这些结果。

```
-- Without parallel query
+------------+-------------+-------------+----------------+
| l_orderkey | revenue     | o_orderdate | o_shippriority |
+------------+-------------+-------------+----------------+
```

```
|   92511430 | 514726.4896 | 1995-03-06  |              0 |
.
.
|   28840519 | 454748.2485 | 1995-03-08  |              0 |
+------------+-------------+-------------+----------------+
10 rows in set (24 min 49.99 sec)
```

```
-- With parallel query
+------------+-------------+-------------+----------------+
| l_orderkey | revenue     | o_orderdate | o_shippriority |
+------------+-------------+-------------+----------------+
|   92511430 | 514726.4896 | 1995-03-06  |              0 |
.
.
|   28840519 | 454748.2485 | 1995-03-08  |              0 |
+------------+-------------+-------------+----------------+
10 rows in set (1 min 49.91 sec)
```

本节中的很多示例查询使用该 TPC-H 数据集中的表，尤其是具有 2000 万行和以下定义的 PART 表。

```
+---------------+---------------+------+-----+---------+-------+
| Field         | Type          | Null | Key | Default | Extra |
+---------------+---------------+------+-----+---------+-------+
| p_partkey     | int(11)       | NO   | PRI | NULL    |       |
| p_name        | varchar(55)   | NO   |     | NULL    |       |
| p_mfgr        | char(25)      | NO   |     | NULL    |       |
| p_brand       | char(10)      | NO   |     | NULL    |       |
| p_type        | varchar(25)   | NO   |     | NULL    |       |
| p_size        | int(11)       | NO   |     | NULL    |       |
| p_container   | char(10)      | NO   |     | NULL    |       |
| p_retailprice | decimal(15,2) | NO   |     | NULL    |       |
| p_comment     | varchar(23)   | NO   |     | NULL    |       |
+---------------+---------------+------+-----+---------+-------+
```

在您的工作负载条件下试验，以了解各个 SQL 语句是否可以利用并行查询。然后，使用以下监控方法帮助验证在一段时间内在实际工作负载条件下使用并行查询的频率。对于实际工作负载，还存在额外的影响因素，例如，并发限制。

监控 Aurora MySQL 的并行查询

如果您的 Aurora MySQL 集群使用并行查询，您可能会看到 VolumeReadIOPS 值出现增长。并行查询不使用缓冲池。因此，尽管查询速度很快，但这种优化的处理可能会导致读取操作和相关费用的增加。

除了在 Amazon RDS 控制台中查看指标中所述的 Amazon CloudWatch 指标以外，Aurora 还提供了其他全局状态变量。可以使用这些全局状态变量来帮助监视并行查询执行情况。它们可以让您深入了解为什么优化程序在给定情况下可能使用或不使用并行查询。要访问这些变量，您可以使用 SHOW GLOBAL STATUS 命令。您还可以找到在下面列出的这些变量。

并行查询会话不一定与由数据库执行的查询呈一一对应关系。例如，假设您的查询计划具有两个使用并行查询的步骤。在这种情况下，查询涉及两个并行会话，并且尝试的请求和成功请求的计数器增加 2 个。

在执行 EXPLAIN 语句以试验并行查询时，即使查询没有实际运行，也会看到指定为“未选择”的计数器增加。在生产环境中使用并行查询时，您可以检查“未选择”计数器的增加速度是否比预期速度快。此时，您可以进行调整，以便为您期望的查询运行并行查询。为此，您可以更改集群设置、查询组合、开启并行查询的数据库实例等。

将在数据库实例级别跟踪这些计数器。在连接到不同的终端节点时，您可能会看到不同的指标，因为每个数据库实例运行自己的一组并行查询。如果读取器终端节点在每个会话中连接到不同的数据库实例，您可能也会看到不同的指标。

名称	描述
Aurora_pq_bytes_returned	在并行查询期间传输到头节点的元组数据结构的字节数。除以 16,384 以与 Aurora_pq_pages_pushed_down 进行比较。
Aurora_pq_max_concurrent_requests	可以在该 Aurora 数据库实例上并发运行的最大并行查询会话数。这是一个取决于 AWS 数据库实例类的固定数字。
Aurora_pq_pages_pushed_down	并行查询避免通过网络传输到头节点的数据页面数量（每个页面具有 16 KiB 的固定大小）。

名称	描述
Aurora_pq_request_attempted	请求的并行查询会话数。该值可能表示每个查询具有多个会话，具体取决于 SQL 结构，如子查询和联接。
Aurora_pq_request_executed	成功运行的并行查询会话数。
Aurora_pq_request_failed	向客户端返回错误的并行查询会话数。在某些情况下，并行查询请求可能会失败，例如，由于在存储层中出现问题。在这些情况下，将使用非并行查询机制重试失败的查询部分。如果重试的查询也失败，则会向客户端返回错误并增加该计数器。
Aurora_pq_request_in_progress	当前运行的并行查询会话数。该数字适用于您连接到的特定 Aurora 数据库实例，而不适用于整个 Aurora 数据库集群。要查看数据库实例是否接近其并发限制，请将该值与 Aurora_pq_max_concurrent_requests 进行比较。
Aurora_pq_request_not_chosen	未选择并行查询以满足查询条件的次数。该值是几个其他更精细的计数器的总和。即使没有实际执行查询，EXPLAIN 语句也可能增加此计数器。
Aurora_pq_request_not_chosen_below_min_rows	由于表中的行数而未选择并行查询的次数。即使没有实际执行查询，EXPLAIN 语句也可能增加此计数器。
Aurora_pq_request_not_chosen_column_bit	由于投影列的列表中的数据类型不受支持而使用非并行查询处理路径的并行查询请求数。
Aurora_pq_request_not_chosen_column_geometry	由于表具有 GEOMETRY 数据类型的列而使用非并行查询处理路径的并行查询请求数。有关删除此限制的 Aurora MySQL 版本的信息，请参阅将并行查询集群升级到 Aurora MySQL 版本 3。

名称	描述
Aurora_pq_request_not_chosen_column_lob	由于表具有 LOB 数据类型的列或具有（由于声明的长度）而在外部存储的 VARCHAR 列，因此使用非并行查询处理路径的并行查询请求数。有关删除此限制的 Aurora MySQL 版本的信息，请参阅 将并行查询集群升级到 Aurora MySQL 版本 3。
Aurora_pq_request_not_chosen_column_virtual	由于表包含虚拟列而使用非并行查询处理路径的并行查询请求数。
Aurora_pq_request_not_chosen_custom_charset	由于表具有带自定义字符集的列而使用非并行查询处理路径的并行查询请求数。
Aurora_pq_request_not_chosen_fast_ddl	由于表当前正在被快速 DDL ALTER 语句更改而使用非并行查询处理路径的并行查询请求数。
Aurora_pq_request_not_chosen_few_pages_outside_buffer_pool	由于没有足够的未缓冲表数据以值得运行并行查询而未选择并行查询的次数，即使缓冲池中的表数据少于 95%。
Aurora_pq_request_not_chosen_full_text_index	由于表具有全文索引而使用非并行查询处理路径的并行查询请求数。
Aurora_pq_request_not_chosen_high_buffer_pool_pct	由于在缓冲池中具有较高比例的表数据（目前大于 95%）而未选择并行查询的次数。在这些情况下，优化程序确定从缓冲池中读取数据更高效。即使没有实际执行查询，EXPLAIN 语句也可能增加此计数器。
Aurora_pq_request_not_chosen_index_hint	由于查询包含索引提示而使用非并行查询处理路径的并行查询请求数。

名称	描述
Aurora_pq_request_not_chosen_innodb_table_format	由于表使用不受支持的 InnoDB 行格式，因此使用非并行查询处理路径的并行查询请求数。Aurora 并行查询仅适用于 COMPACT、REDUNDANT 和 DYNAMIC 行格式。
Aurora_pq_request_not_chosen_long_trx	由于正在长时间运行的事务中启动查询而使用非并行查询处理路径的并行查询请求数。即使没有实际执行查询，EXPLAIN 语句也可能增加此计数器。
Aurora_pq_request_not_chosen_no_where_clause	由于查询不包含任何 WHERE 子句而使用非并行查询处理路径的并行查询请求数。
Aurora_pq_request_not_chosen_range_scan	由于查询对索引使用范围扫描而使用非并行查询处理路径的并行查询请求数。
Aurora_pq_request_not_chosen_row_length_too_long	由于所有列的总组合长度过长而使用非并行查询处理路径的并行查询请求数。
Aurora_pq_request_not_chosen_small_table	由于表的总大小（由行数和平均行长度确定）而未选择并行查询的次数。即使没有实际执行查询，EXPLAIN 语句也可能增加此计数器。
Aurora_pq_request_not_chosen_temporary_table	由于查询引用了临时表（这些临时表使用不受支持的 MyISAM 或 memory 表类型）而使用非并行查询处理路径的并行查询请求数。
Aurora_pq_request_not_chosen_tx_isolation	由于查询使用不受支持的事务隔离级别而使用非并行查询处理路径的并行查询请求数。在读取器数据库实例上，并行查询仅适用于 REPEATABLE READ 和 READ COMMITTED 隔离级别。
Aurora_pq_request_not_chosen_update_delete_stmts	由于查询是 UPDATE 或 DELETE 语句的一部分而使用非并行查询处理路径的并行查询请求数。

名称	描述
Aurora_pq_request_not_chosen_unsupported_access	由于 WHERE 子句不符合并行查询条件而使用非并行查询处理路径的并行查询请求数。如果查询不需要数据密集型扫描，或者查询是 DELETE 或 UPDATE 语句，则会出现该结果。
Aurora_pq_request_not_chosen_unsupported_storage_type	由于 Aurora MySQL 数据库集群未使用支持的 Aurora 集群存储配置，而使用非并行查询处理路径的并行查询请求的数量。此参数适用于 Aurora MySQL 版本 3.04 及更高版本。有关更多信息，请参阅 限制。
Aurora_pq_request_throttled	由于在特定 Aurora 数据库实例上已运行的最大并发并行查询数而未选择并行查询的次数。

Aurora MySQL 中用于并行查询的 SQL 构造

在下一节中，您可以找到为什么特定 SQL 语句使用或不使用并行查询的更多详细信息。本节还详细介绍了 Aurora MySQL 功能如何与并行查询交互。此信息可以帮助您诊断使用并行查询的集群的性能问题，或了解并行查询如何应用于您的特定工作负载。

确定是否使用并行查询取决于在运行语句时存在的很多因素。因此，某些查询可能始终、从不或仅在某些情况下使用并行查询。

> **Tip**
>
> 当您在 HTML 中查看这些示例时，可以使用每个代码列表右上角的复制小部件来复制 SQL 代码以便自行尝试。使用复制小部件可避免复制 mysql> 提示行和 -> 延续行周围的多余字符。

主题

- EXPLAIN 语句
- WHERE 子句
- 数据定义语言 (DDL)
- 列数据类型

- 分区表
- 聚合函数、GROUP BY 子句和 HAVING 子句
- WHERE 子句中的函数调用
- LIMIT 子句
- 比较运算符
- 联接
- 子查询
- 联合
- 视图
- 数据操作语言 (DML) 语句
- 事务和锁定
- B 树索引
- 全文搜索 (FTS) 索引
- 虚拟列
- 内置缓存机制
- 优化程序提示
- MyISAM 临时表

EXPLAIN 语句

正如本节中的示例所示，EXPLAIN 语句指示查询的每个阶段当前是否适合运行并行查询。它还指示可以将查询的哪些方面向下推送到存储层。查询计划中的最重要项目如下所示：

- NULL 列的 key 以外的值表明可以使用索引查找高效地执行查询，而不会运行并行查询。
- 较小的 rows 列值（不是数百万的值）表明查询没有访问足够的数据以值得运行并行查询。这意味着不太可能使用并行查询。
- Extra 列显示是否需要使用并行查询。该输出类似于以下示例。

```
Using parallel query (A columns, B filters, C exprs; D extra)
```

  columns 数字表示查询块中引用的列数。

filters 数字表示 WHERE 谓词数，它表示列值与常数的简单比较。比较的结果可以是相等、不相等，或者是处于某个范围内。Aurora 可以十分高效地并行处理此类谓词。

exprs 数字表示也可以并行处理但不像筛选条件那样高效的表达式数，例如，函数调用、运算符或其他表达式。

extra 数字表示无法向下推送并由头节点执行的表达式数。

例如，请考虑以下 EXPLAIN 输出。

```
mysql> explain select p_name, p_mfgr from part
    -> where p_brand is not null
    -> and upper(p_type) is not null
    -> and round(p_retailprice) is not null;
+----+-------------+-------+...+----------
+-------------------------------------------------------------------------------+
| id | select_type | table |...| rows     | Extra
                                |
+----+-------------+-------+...+----------
+-------------------------------------------------------------------------------+
|  1 | SIMPLE      | part  |...| 20427936 | Using where; Using parallel query (5
 columns, 1 filters, 2 exprs; 0 extra) |
+----+-------------+-------+...+----------
+-------------------------------------------------------------------------------+
```

Extra 列中的信息显示从每行中提取 5 列以计算查询条件并构建结果集。一个 WHERE 谓词涉及一个筛选条件，即，在 WHERE 子句中直接测试的列。两个 WHERE 子句需要计算更复杂的表达式，在这种情况下，将涉及函数调用。0 extra 字段确认 WHERE 子句中的所有操作将作为并行查询处理的一部分向下推送到存储层。

如果未选择并行查询，您通常可以从 EXPLAIN 输出的其他列中推断出原因。例如，rows 值可能太小，或者 possible_keys 列可能表示查询可以使用索引查找，而不是数据密集型扫描。以下示例显示了一个查询，其中优化程序可以估计查询将仅扫描少量的行。它根据主键的特性执行此操作。在这种情况下，不需要运行并行查询。

```
mysql> explain select count(*) from part where p_partkey between 1 and 100;
+----+-------------+-------+-------+---------------+---------+---------+------+------
+--------------------------+
| id | select_type | table | type  | possible_keys | key     | key_len | ref  | rows |
 Extra                    |
```

```
+----+-------------+-------+-------+---------------+---------+---------+------+------
+--------------------------+
|  1 | SIMPLE      | part  | range | PRIMARY       | PRIMARY | 4       | NULL |   99 |
 Using where; Using index |
+----+-------------+-------+-------+---------------+---------+---------+------+------
+--------------------------+
```

显示是否使用并行查询的输出考虑了在运行 EXPLAIN 语句时存在的所有因素。在实际运行查询时，如果在此期间情况发生变化，优化程序可能会做出不同的选择。例如，EXPLAIN 可能会报告语句将使用并行查询。但在以后实际运行查询时，它可能会根据此时的情况不使用并行查询。此类条件可以包括同时运行其他几个并行查询。此类情况还可能包括从表中删除行，创建新的索引，在打开事务中经过的时间太长，等等。

WHERE 子句

要使查询使用并行查询优化，它必须 包含一个 WHERE 子句。

并行查询优化加快了 WHERE 子句中使用的多种类型的表达式的速度：

- 列值与常数的简单比较，称为筛选条件。这些比较从向下推送到存储层中受益最多。将在 EXPLAIN 输出中报告查询中的筛选条件表达式数。
- 如果可能，WHERE 子句中的其他类型的表达式也会向下推送到存储层。将在 EXPLAIN 输出中报告查询中的此类表达式数。这些表达式可能是函数调用、LIKE 运算符、CASE 表达式，等等。
- 目前，并行查询不会向下推送某些函数和运算符。查询中的此类表达式数将在 extra 输出中报告为 EXPLAIN 计数器。查询的其余部分仍然可以使用并行查询。
- 虽然不会向下推送选择列表中的表达式，但包含此类函数的查询仍然可以从并行查询的中间结果的网络流量减少中受益。例如，在选择列表中调用聚合函数的查询可以从并行查询中受益，即使不向下推送聚合函数。

例如，以下查询执行全表扫描并处理 P_BRAND 列的所有值。不过，它不使用并行查询，因为查询不包含任何 WHERE 子句。

```
mysql> explain select count(*), p_brand from part group by p_brand;
+----+-------------+-------+------+---------------+------+---------+------+----------
+----------------------------------+
| id | select_type | table | type | possible_keys | key  | key_len | ref  | rows     |
 Extra                            |
+----+-------------+-------+------+---------------+------+---------+------+----------
+----------------------------------+
```

```
|  1 | SIMPLE      | part  | ALL  | NULL          | NULL | NULL    | NULL | 20427936 |
 Using temporary; Using filesort |
+----+-------------+-------+------+---------------+------+---------+------+----------
+---------------------------------+
```

相反，以下查询包括筛选结果的 WHERE 谓词，因此，可以应用并行查询：

```
mysql> explain select count(*), p_brand from part where p_name is not null
    ->   and p_mfgr in ('Manufacturer#1', 'Manufacturer#3') and p_retailprice > 1000
    -> group by p_brand;
+----+...+----------
+-------------------------------------------------------------------------------------------------------
+
| id |...| rows     | Extra
                                              |
+----+...+----------
+-------------------------------------------------------------------------------------------------------
+
|  1 |...| 20427936 | Using where; Using temporary; Using filesort; Using parallel
 query (5 columns, 1 filters, 2 exprs; 0 extra) |
+----+...+----------
+-------------------------------------------------------------------------------------------------------
+
```

如果优化程序估计查询块的返回行数很少，则不会在该查询块中使用并行查询。以下示例说明了一种主键列上的大于运算符应用于数百万行的情况，这会导致使用并行查询。估计反向小于测试仅应用于几行，而不使用并行查询。

```
mysql> explain select count(*) from part where p_partkey > 10;
+----+...+----------
+-------------------------------------------------------------------------------+
| id |...| rows     | Extra
          |
+----+...+----------
+-------------------------------------------------------------------------------+
|  1 |...| 20427936 | Using where; Using parallel query (1 columns, 1 filters, 0 exprs;
 0 extra) |
+----+...+----------
+-------------------------------------------------------------------------------+

mysql> explain select count(*) from part where p_partkey < 10;
+----+...+------+-------------------------+
```

```
| id |...| rows | Extra                     |
+----+...+------+---------------------------+
|  1 |...|    9 | Using where; Using index |
+----+...+------+---------------------------+
```

数据定义语言 (DDL)

在 Aurora MySQL 版本 2 中，并行查询仅适用于没有挂起的快速数据定义语言（DDL）操作的表。在 Aurora MySQL 版本 3 中，您可以在表上与即时 DDL 操作同时使用并行查询。

Aurora MySQL 版本 3 中的即时 DDL 取代了 Aurora MySQL 版本 2 中的快速 DDL 功能。有关即时 DDL 的信息，请参阅 即时 DDL（Aurora MySQL 版本 3）。

列数据类型

在 Aurora MySQL 版本 3 中，并行查询可以处理包含具有数据类型 TEXT、BLOB、JSON 和 GEOMETRY 的列的表。它也可以使用最大声明长度超过 768 字节的 VARCHAR 和 CHAR 列。如果您的查询引用任何包含此类大型对象类型的列，则检索它们的额外工作确实会增加查询处理的一些开销。在这种情况下，请检查查询是否可以省略对这些列的引用。如果没有，运行基准测试以确认在开启或关闭并行查询的情况下，此类查询是否更快。

在 Aurora MySQL 版本 2 中，并行查询对于大型对象类型有以下限制：

- TEXT、BLOB、JSON 和 GEOMETRY 数据类型不支持并行查询。引用这些类型的任何列的查询无法使用并行查询。
- 可变长度列（VARCHAR 和 CHAR 数据类型）与并行查询兼容，最大声明长度最多为 768 字节。如果查询引用的任何列具有使用更长最大长度声明的类型，则无法使用并行查询。对于使用多字节字符集的列，字节限制将字符集中的最大字节数考虑在内。例如，对于字符集 utf8mb4（最大字符长度为 4 字节），VARCHAR(192) 列与并行查询兼容，但 VARCHAR(193) 列不兼容。

分区表

在 Aurora MySQL 版本 3 中，您可以将分区表与并行查询结合使用。由于分区表在内部表示为多个较小的表，因此对非分区表使用并行查询的查询可能不会对相同的分区表使用并行查询。Aurora MySQL 考虑每个分区是否足以符合并行查询优化条件，而不是评估整个表的大小。检查是否Aurora_pq_request_not_chosen_small_table如果分区表上的查询在预期时不使用并行查询，则状态变量将递增。

例如，考虑用 PARTITION BY HASH (*column*) PARTITIONS 2 分区的一个表和用 PARTITION BY HASH (*column*) PARTITIONS 10 分区的另一个表。在有两个分区的表中，分区的大小是有

十个分区的表的五倍。因此，并行查询更有可能用于对分区较少的表进行查询。在以下示例中，表 PART_BIG_PARTITIONS 有两个分区，PART_SMALL_PARTITIONS 有十个分区。在数据相同的情况下，并行查询更有可能用于大分区较少的表。

```
mysql> explain select count(*), p_brand from part_big_partitions where p_name is not
 null
    ->   and p_mfgr in ('Manufacturer#1', 'Manufacturer#3') and p_retailprice > 1000
 group by p_brand;
+----+-------------+---------------------+------------
+-------------------------------------------------------------------------------------------
+
| id | select_type | table               | partitions | Extra
                                                                                          |
+----+-------------+---------------------+------------
+-------------------------------------------------------------------------------------------
+
|  1 | SIMPLE      | part_big_partitions | p0,p1      | Using where; Using temporary;
 Using parallel query (4 columns, 1 filters, 1 exprs; 0 extra; 1 group-bys, 1 aggrs) |
+----+-------------+---------------------+------------
+-------------------------------------------------------------------------------------------
+

mysql> explain select count(*), p_brand from part_small_partitions where p_name is not
 null
    ->   and p_mfgr in ('Manufacturer#1', 'Manufacturer#3') and p_retailprice > 1000
 group by p_brand;
+----+-------------+----------------------+------------------------------
+------------------------------+
| id | select_type | table                | partitions                   | Extra
                   |
+----+-------------+----------------------+------------------------------
+------------------------------+
|  1 | SIMPLE      | part_small_partitions | p0,p1,p2,p3,p4,p5,p6,p7,p8,p9 | Using
 where; Using temporary |
+----+-------------+----------------------+------------------------------
+------------------------------+
```

聚合函数、GROUP BY 子句和 HAVING 子句

涉及聚合函数的查询通常是并行查询的理想之选，因为它们涉及扫描大型表中的大量行。

在 Aurora MySQL 3 中，并行查询可以优化选择列表和 HAVING 子句中的聚合函数调用。

在 Aurora MySQL 3 之前，选择列表或 HAVING 子句中的聚合函数调用不会向下推送到存储层。不过，并行查询仍然可以使用聚合函数提高此类查询的性能。为此，它先在存储层中从原始数据页面中并行提取列值。然后，它以紧凑元组格式将这些值发回到头节点，而不是作为完整数据页面。与往常一样，查询需要具有至少一个 WHERE 谓词才能激活并行查询。

以下简单示例说明了可以从并行查询中受益的聚合查询种类。它们以紧凑形式将中间结果返回到头节点以及/或者从中间结果中筛选不匹配的行。

```
mysql> explain select sql_no_cache count(distinct p_brand) from part where p_mfgr =
 'Manufacturer#5';
+----+...+-------------------------------------------------------------------------------+
| id |...| Extra                                                                         |
+----+...+-------------------------------------------------------------------------------+
|  1 |...| Using where; Using parallel query (2 columns, 1 filters, 0 exprs; 0 extra) |
+----+...+-------------------------------------------------------------------------------+

mysql> explain select sql_no_cache p_mfgr from part where p_retailprice > 1000 group by
 p_mfgr having count(*) > 100;
+----+...
+-----------------------------------------------------------------------------------------------------
+
| id |...| Extra
                                  |
+----+...
+-----------------------------------------------------------------------------------------------------
+
|  1 |...| Using where; Using temporary; Using filesort; Using parallel query (3
 columns, 0 filters, 1 exprs; 0 extra) |
+----+...
+-----------------------------------------------------------------------------------------------------
+
```

WHERE 子句中的函数调用

Aurora 可以将并行查询优化应用于 WHERE 子句中的大多数内置函数调用。并行处理这些函数调用将会从头节点中卸载一些 CPU 负载。通过在最早的查询阶段并行计算谓词函数，有助于 Aurora 最大限度减少在后续阶段传输和处理的数据量。

目前，并行处理不适用于选择列表中的函数调用。头节点计算这些函数，即使在 WHERE 子句中出现相同的函数调用。来自相关列的原始值包含在从存储节点发回到头节点的元组中。头节点执行任何转换（如 UPPER、CONCATENATE 等）以生成结果集的最终值。

在以下示例中，并行查询将并行处理对 LOWER 的调用，因为它出现在 WHERE 子句中。并行查询不会影响对 SUBSTR 和 UPPER 的调用，因为它们出现在选择列表中。

```
mysql> explain select sql_no_cache distinct substr(upper(p_name),1,5) from part
    -> where lower(p_name) like '%cornflower%' or lower(p_name) like '%goldenrod%';
+----+...
+-------------------------------------------------------------------------------------------
+
| id |...| Extra
                |
+----+...
+-------------------------------------------------------------------------------------------
+
|  1 |...| Using where; Using temporary; Using parallel query (2 columns, 0 filters, 1
 exprs; 0 extra) |
+----+...
+-------------------------------------------------------------------------------------------
+
```

相同的注意事项适用于其他表达式，例如，CASE 表达式或 LIKE 运算符。例如，以下示例显示并行查询计算 CASE 子句中的 LIKE 表达式和 WHERE 运算符。

```
mysql> explain select p_mfgr, p_retailprice from part
    -> where p_retailprice > case p_mfgr
    ->   when 'Manufacturer#1' then 1000
    ->   when 'Manufacturer#2' then 1200
    ->   else 950
    -> end
    -> and p_name like '%vanilla%'
    -> group by p_retailprice;
+----+...
+-------------------------------------------------------------------------------------------
+
| id |...| Extra
                                  |
+----+...
+-------------------------------------------------------------------------------------------
+
|  1 |...| Using where; Using temporary; Using filesort; Using parallel query (4
 columns, 0 filters, 2 exprs; 0 extra) |
```

```
+----+...
+-------------------------------------------------------------------------------------------------------
+
```

LIMIT 子句

目前，并行查询不用于包含 LIMIT 子句的任何查询块。并行查询可能仍用于具有 GROUP by、ORDER BY 或联接的早期查询阶段。

比较运算符

优化程序估计要扫描的行数以计算比较运算符，并根据该估计确定是否使用并行查询。

下面的第一个示例显示，可以在没有并行查询的情况下高效地执行与主键列的相等比较。下面的第二个示例显示，与未编制索引的列进行类似比较需要扫描数百万行，因此，可以从并行查询中受益。

```
mysql> explain select * from part where p_partkey = 10;
+----+...+------+-------+
| id |...| rows | Extra |
+----+...+------+-------+
|  1 |...|    1 | NULL  |
+----+...+------+-------+

mysql> explain select * from part where p_type = 'LARGE BRUSHED BRASS';
+----+...+----------
+-------------------------------------------------------------------------------+
| id |...| rows     | Extra
          |
+----+...+----------
+-------------------------------------------------------------------------------+
|  1 |...| 20427936 | Using where; Using parallel query (9 columns, 1 filters, 0 exprs;
 0 extra) |
+----+...+----------
+-------------------------------------------------------------------------------+
```

同样的注意事项适用于不等于测试和范围比较，例如小于、大于或等于或 BETWEEN。优化程序估计要扫描的行数，并根据 I/O 总量确定是否值得运行并行查询。

联接

大型表的联接查询通常涉及数据密集型操作，这些操作将从并行查询优化中受益。目前，不会并行处理多个表之间的列值比较（即，联接谓词本身）。不过，并行查询可以向下推送其他联接阶段的一些内部

处理，例如，在哈希联接期间构建 Bloom 筛选条件。即使没有 WHERE 子句，并行查询也可以应用于联接查询。因此，对于需要使用 WHERE 子句以使用并行查询的规则，联接查询是一种例外情况。

将计算联接处理的每个阶段以检查它是否符合并行查询条件。如果多个阶段可以使用并行查询，将按顺序执行这些阶段。因此，每个联接查询在并发限制中计为单个并行查询会话。

例如，如果联接查询包含 WHERE 谓词以筛选联接的某个表中的行，该筛选选项可以使用并行查询。作为另一个示例，假设联接查询使用哈希联接机制，例如，将大表与小表联接在一起。在这种情况下，用于生成 Bloom 筛选条件数据结构的表扫描或许可以使用并行查询。

> **Note**
>
> 并行查询通常用于受益于哈希联接优化的各种资源密集型查询。开启哈希联接优化的方法取决于 Aurora MySQL 版本。有关各版本的详细信息，请参阅 为并行查询集群开启哈希联接。有关如何有效使用哈希联接的信息，请参阅 使用哈希联接优化大型 Aurora MySQL 联接查询。

```
mysql> explain select count(*) from orders join customer where o_custkey = c_custkey;
+----+...+----------+-------+--------------+-------------+...+-----------
+-------------------------------------------------------------------------------------------------
+
| id |...| table    | type  | possible_keys | key         |...| rows      | Extra

              |
+----+...+----------+-------+--------------+-------------+...+-----------
+-------------------------------------------------------------------------------------------------
+
|  1 |...| customer | index | PRIMARY       | c_nationkey |...|  15051972 | Using index

              |
|  1 |...| orders   | ALL   | o_custkey     | NULL        |...| 154545408 | Using join
 buffer (Hash Join Outer table orders); Using parallel query (1 columns, 0 filters, 1
 exprs; 0 extra) |
+----+...+----------+-------+--------------+-------------+...+-----------
+-------------------------------------------------------------------------------------------------
+
```

对于使用嵌套循环机制的联接查询，最外层的嵌套循环块可能会使用并行查询。是否使用并行查询取决于与往常相同的因素，例如，是否在 WHERE 子句中存在其他筛选条件。

```
mysql> -- Nested loop join with extra filter conditions can use parallel query.
```

```
mysql> explain select count(*) from part, partsupp where p_partkey != ps_partkey and
 p_name is not null and ps_availqty > 0;
+----+-------------+----------+...+----------
+-------------------------------------------------------------------------------+
| id | select_type | table    |...| rows     | Extra
                                 |
+----+-------------+----------+...+----------
+-------------------------------------------------------------------------------+
|  1 | SIMPLE      | part     |...| 20427936 | Using where; Using parallel query (2
 columns, 1 filters, 0 exprs; 0 extra) |
|  1 | SIMPLE      | partsupp |...| 78164450 | Using where; Using join buffer (Block
 Nested Loop)                       |
+----+-------------+----------+...+----------
+-------------------------------------------------------------------------------+
```

子查询

外部查询块和内部子查询块可能分别使用并行查询，也可能不使用。它们是否这样做取决于针对每个块的表、 WHERE 子句等的常用特征。例如，以下查询在子查询块中使用并行查询，但在外部块中不使用并行查询。

```
mysql> explain select count(*) from part where
   --> p_partkey < (select max(p_partkey) from part where p_name like '%vanilla%');
+----+-------------+...+----------
+-------------------------------------------------------------------------------+
| id | select_type |...| rows     | Extra
                       |
+----+-------------+...+----------
+-------------------------------------------------------------------------------+
|  1 | PRIMARY     |...|     NULL | Impossible WHERE noticed after reading const tables
                      |
|  2 | SUBQUERY    |...| 20427936 | Using where; Using parallel query (2 columns, 0
 filters, 1 exprs; 0 extra) |
+----+-------------+...+----------
+-------------------------------------------------------------------------------+
```

目前，关联子查询无法使用并行查询优化。

联合

对于 UNION 的每个部分，根据表和 WHERE 子句等的通常特性，UNION 查询中的每个查询块可能会使用或不使用并行查询。

```
mysql> explain select p_partkey from part where p_name like '%choco_ate%'
    -> union select p_partkey from part where p_name like '%vanil_a%';
+----+---------------+...+----------
+-------------------------------------------------------------------------------+
| id | select_type    |...| rows     | Extra
                          |
+----+---------------+...+----------
+-------------------------------------------------------------------------------+
|  1 | PRIMARY        |...| 20427936 | Using where; Using parallel query (2 columns, 0
 filters, 1 exprs; 0 extra) |
|  2 | UNION          |...| 20427936 | Using where; Using parallel query (2 columns, 0
 filters, 1 exprs; 0 extra) |
| NULL | UNION RESULT | <union1,2> |...|     NULL | Using temporary
                        |
+----+--------------+...+----------
+-------------------------------------------------------------------------------+
```

> **Note**
>
> 查询中的每个 UNION 子句是按顺序运行的。即使查询包含的多个阶段均使用并行查询，它在任何时间也仅运行单个并行查询。因此，甚至是复杂的多阶段查询也仅在并发并行查询限制中计为 1 个。

视图

优化程序将视图作为使用基础表的较长查询以重写任何查询。因此，无论表引用是视图还是实际表，并行查询的工作方式都是相同的。有关在查询中是否使用并行查询以及向下推送哪些部分的所有相同注意事项适用于最终重写的查询。

例如，以下查询计划显示通常不使用并行查询的视图定义。在使用额外的 WHERE 子句查询视图时，Aurora MySQL 使用并行查询。

```
mysql> create view part_view as select * from part;
mysql> explain select count(*) from part_view where p_partkey is not null;
+----+...+----------
+-------------------------------------------------------------------------------+
| id |...| rows     | Extra
          |
+----+...+----------
+-------------------------------------------------------------------------------+
```

```
|  1 |...| 20427936 | Using where; Using parallel query (1 columns, 0 filters, 0 exprs;
 1 extra) |
+----+...+----------
+-------------------------------------------------------------------------------+
```

数据操作语言 (DML) 语句

如果 INSERT 部分满足并行查询的其他条件，则 SELECT 语句可以在 SELECT 处理阶段中使用并行查询。

```
mysql> create table part_subset like part;
mysql> explain insert into part_subset select * from part where p_mfgr =
 'Manufacturer#1';
+----+...+----------
+-------------------------------------------------------------------------------+
| id |...| rows     | Extra
          |
+----+...+----------
+-------------------------------------------------------------------------------+
|  1 |...| 20427936 | Using where; Using parallel query (9 columns, 1 filters, 0 exprs;
 0 extra) |
+----+...+----------
+-------------------------------------------------------------------------------+
```

> **Note**
>
> 通常，在执行 INSERT 语句后，新插入的行的数据将位于缓冲池中。因此，在插入大量行后，表可能不符合并行查询条件。以后，在正常运行期间从缓冲池中移出数据后，针对表的查询可能会再次开始使用并行查询。

CREATE TABLE AS SELECT 语句不使用并行查询，即使该语句的 SELECT 部分符合并行查询条件。该语句的 DDL 特性导致它与并行查询处理不兼容。相反，在 INSERT ... SELECT 语句中，SELECT 部分可以使用并行查询。

在 DELETE 或 UPDATE 语句中从不使用并行查询，而无论表的大小和 WHERE 子句中的谓词如何。

```
mysql> explain delete from part where p_name is not null;
+----+-------------+...+----------+-------------+
| id | select_type |...| rows     | Extra       |
+----+-------------+...+----------+-------------+
```

```
|  1 | SIMPLE      |...| 20427936 | Using where |
+----+-------------+...+----------+-------------+
```

事务和锁定

您可以在 Aurora 主实例上使用所有隔离级别。

在 Aurora 读取器数据库实例上，并行查询适用于 REPEATABLE READ 隔离级别下执行的语句。Aurora MySQL 版本 2.09 或更高版本也可以在读取器数据库实例上使用 READ COMMITTED 隔离级别。REPEATABLE READ 是 Aurora 读取器数据库实例的原定设置隔离级别。要在读取器数据库实例上使用 READ COMMITTED 隔离级别，需要在会话级别设置 aurora_read_replica_read_committed 配置选项。读取器实例的 READ COMMITTED 隔离级别符合 SQL 标准行为。但是，与查询在写入器实例上使用 READ COMMITTED 隔离级别时相比，对读取器实例的隔离没有那么严格。

有关 Aurora 隔离级别的更多信息，特别是写入器实例与读取器实例之间的 READ COMMITTED 区别，请参阅 Aurora MySQL 隔离级别。

在较大的事务完成后，表统计数据可能会过时。这种过时的统计数据可能要求使用 ANALYZE TABLE 语句，然后 Aurora 才能准确估计行数。大型 DML 语句可能还会将大部分的表数据放入缓冲池中。将该数据放入缓冲池可能会导致不常为该表选择并行查询，直到将数据从池中移出。

如果您的会话位于长时间运行的事务中（默认为 10 分钟），则该会话中的其他查询不会使用并行查询。在单个长时间运行的查询期间，也可能会发生超时。如果查询在并行查询处理开始之前运行的时间超过最大间隔（当前为 10 分钟），则可能会发生这种超时。

您可以在执行临时（一次）查询的 autocommit=1 会话中设置 mysql，以降低意外启动长时间运行的事务的可能性。甚至针对表的 SELECT 语句也可以创建读取视图以开始运行事务。读取视图 是一个用于后续查询的一致数据集，将在提交事务之前保留该数据集。在 Aurora 中使用 JDBC 或 ODBC 应用程序时，也要注意该限制，因为此类应用程序可能会在禁用 autocommit 设置的情况下运行。

以下示例显示对表运行查询如何创建一个读取视图以隐式开始运行事务（在禁用了 autocommit 设置的情况下）。稍后运行的查询仍然可以使用并行查询。不过，在暂停几分钟后，查询不再符合并行查询条件。如果将事务以 COMMIT 或 ROLLBACK 结尾，将恢复并行查询条件。

```
mysql> set autocommit=0;

mysql> explain select sql_no_cache count(*) from part where p_retailprice > 10.0;
+----+...+---------
+-------------------------------------------------------------------------+
```

```
| id |...| rows    | Extra
         |
+----+...+---------
+-----------------------------------------------------------------------------+
|  1 |...| 2976129 | Using where; Using parallel query (1 columns, 1 filters, 0 exprs;
 0 extra) |
+----+...+---------
+-----------------------------------------------------------------------------+

mysql> select sleep(720); explain select sql_no_cache count(*) from part where
 p_retailprice > 10.0;
+------------+
| sleep(720) |
+------------+
|          0 |
+------------+
1 row in set (12 min 0.00 sec)

+----+...+---------+-------------+
| id |...| rows    | Extra       |
+----+...+---------+-------------+
|  1 |...| 2976129 | Using where |
+----+...+---------+-------------+

mysql> commit;

mysql> explain select sql_no_cache count(*) from part where p_retailprice > 10.0;
+----+...+---------
+-----------------------------------------------------------------------------+
| id |...| rows    | Extra
         |
+----+...+---------
+-----------------------------------------------------------------------------+
|  1 |...| 2976129 | Using where; Using parallel query (1 columns, 1 filters, 0 exprs;
 0 extra) |
+----+...+---------
+-----------------------------------------------------------------------------+
```

要查看查询由于位于长时间运行的事务中而不符合并行查询条件的次数，请检查 Aurora_pq_request_not_chosen_long_trx 状态变量。

```
mysql> show global status like '%pq%trx%';
+---------------------------------------+-------+
```

```
| Variable_name                          | Value |
+----------------------------------------+-------+
| Aurora_pq_request_not_chosen_long_trx | 4     |
+-------------------------------+-------+
```

获取锁定的任何 SELECT 语句（如 SELECT FOR UPDATE 或 SELECT LOCK IN SHARE MODE 语法）无法使用并行查询。

并行查询可以用于 LOCK TABLES 语句锁定的表。

```
mysql> explain select o_orderpriority, o_shippriority from orders where o_clerk =
 'Clerk#000095055';
+----+...+-----------
+-------------------------------------------------------------------------------+
| id |...| rows      | Extra
           |
+----+...+-----------
+-------------------------------------------------------------------------------+
|  1 |...| 154545408 | Using where; Using parallel query (3 columns, 1 filters, 0
 exprs; 0 extra) |
+----+...+-----------
+-------------------------------------------------------------------------------+

mysql> explain select o_orderpriority, o_shippriority from orders where o_clerk =
 'Clerk#000095055' for update;
+----+...+-----------+-------------+
| id |...| rows      | Extra       |
+----+...+-----------+-------------+
|  1 |...| 154545408 | Using where |
+----+...+-----------+-------------+
```

B 树索引

根据每个列的数据特性，ANALYZE TABLE 语句收集的统计数据可以帮助优化程序确定何时使用并行查询或索引查找。在执行对表中的数据进行重大更改的 DML 操作后，请运行 ANALYZE TABLE 以将统计数据保持最新状态。

如果索引查找可以在没有数据密集型扫描的情况下高效地执行查询，则 Aurora 可能会使用索引查找。这样做可以避免并行查询处理的开销。可以在任何 Aurora 数据库集群上同时运行的并行查询数也存在并发限制。确保使用最佳实践为表编制索引，以便最频繁和最高并发性的查询使用索引查找。

全文搜索 (FTS) 索引

目前，并行查询不用于包含全文搜索索引的表，而无论查询是引用此类索引列，还是使用 MATCH 运算符。

虚拟列

目前，并行查询不适用于包含虚拟列的表，无论查询是否引用任何虚拟列。

内置缓存机制

Aurora 包括内置缓存机制，即，缓冲池和查询缓存。Aurora 优化程序在这些缓存机制和并行查询之间进行选择，具体取决于哪一个对特定查询最有效。

在并行查询筛选行并转换和提取列值时，数据将作为元组而不是数据页面传回到头节点。因此，运行并行查询不会将任何页面添加到缓冲池中，也不会移出已位于缓冲池中的页面。

Aurora 会检查位于缓冲池中的表数据页数，以及该数字所代表的表数据比例。Aurora 会通过该信息确定使用并行查询是否更加高效，并且决定是否需要绕过缓冲池中的数据。或者，Aurora 可能使用非并行查询处理路径，这会使用缓冲池中缓存的数据。缓存哪些页面以及数据密集型查询如何影响缓存和移出取决于与缓冲池相关的配置设置。因此，很难预测任何特定查询是否使用并行查询，因为这取决于缓冲池中不断变化的数据。

此外，Aurora 对并行查询施加并发限制。由于并非每个查询都使用并行查询，多个查询同时访问的表通常将大部分数据放在缓冲池中。因此，Aurora 通常不会选择这些表以运行并行查询。

在同一个表上运行一系列非并行查询时，由于数据没有位于缓冲池中，第一个查询的速度可能很慢。由于缓冲池现已“预热”，第二次和后续查询要快得多。并行查询通常从针对表的第一个查询开始就具有一致的性能。在进行性能测试时，将使用冷缓冲池和热缓冲池对非并行查询进行基准测试。在某些情况下，使用热缓冲池的结果可能与并行查询时间接近。在这些情况下，请考虑诸如针对该表进行查询的频率等因素。还要考虑是否值得将该表的数据保留在缓冲池中。

在提交相同的查询以及未更改基础表数据时，查询缓存可以避免重新运行查询。并行查询功能优化的查询可以放入查询缓存中，实际上可以在再次运行时立即运行它们。

> **Note**
>
> 在进行性能比较时，查询缓存可能会生成虚假的低计时数。因此，在与基准测试类似的情况下，您可以使用 `sql_no_cache` 提示。该提示可以防止从查询缓存中提供结果，即使以前运行了相同的查询。该提示直接位于查询中的 SELECT 语句后面。本主题中的很多并行查询示例包括该提示，以使开启和关闭并行查询的查询版本具有类似的查询时间。

在生产环境中使用并行查询时，请确保从源中删除该提示。

优化程序提示

控制优化程序的另一种方法是使用优化程序提示，可以在单个语句中指定优化程序提示。例如，您可以对语句中的一个表开启优化，然后对另一个表关闭优化。有关这些提示的更多信息，请参阅《MySQL 参考手册》中的优化程序提示。

您可以将 SQL 提示与 Aurora MySQL 查询结合使用来微调性能。您还可以使用提示来防止重要查询的执行计划由于不可预知的条件而发生变化。

我们扩展了 SQL 提示功能，以帮助您控制查询计划的优化程序选择。这些提示适用于使用并行查询优化的查询。有关更多信息，请参阅 Aurora MySQL 提示。

MyISAM 临时表

并行查询优化仅适用于 InnoDB 表。由于 Aurora MySQL 在后台的临时表中使用 MyISAM，因此，涉及临时表的内部查询阶段从不使用并行查询。这些查询阶段由 `Using temporary` 输出中的 `EXPLAIN` 指示。

在 Amazon Aurora MySQL 数据库集群中使用高级审计

您可以在 Amazon Aurora MySQL 中使用高性能的高级审核功能来审核数据库活动。要启用该功能，您可以通过设置多个数据库集群参数来启用审核日志的收集。在启用了高级审核时，您可以用它来记录任意支持事件的组合。

通过查看或下载审计日志，您可以一次查看一个数据库实例的审计信息。为此，您可以使用 监控 Amazon Aurora 日志文件 中的过程。

> **Tip**
>
> 对于包含多个数据库实例的 Aurora 数据库集群，您会发现检查集群中所有实例的审计日志更方便。为此，您可以使用 CloudWatch Logs。您可以启用集群级别的设置，将 Aurora MySQL 审计日志数据发布到 CloudWatch 中的日志组。然后，通过 CloudWatch 界面查看、筛选和搜索审计日志。有关更多信息，请参阅 将 Amazon Aurora MySQL 日志发布到 Amazon CloudWatch Logs。

启用高级审核

可以使用本节中介绍的参数为数据库集群启用和配置高级审核。

使用 `server_audit_logging` 参数启用或禁用高级审计。

使用 `server_audit_events` 参数指定要记录的事件。

使用 `server_audit_incl_users` 和 `server_audit_excl_users` 参数来指定审核的对象。默认情况下会审计所有用户。有关一个或两个参数留空或者两者均指定相同用户名时这些参数工作方式的详细信息，请参阅 server_audit_incl_users 和 server_audit_excl_users。

通过在数据库集群使用的参数组中设置这些参数来配置高级审核。您可以使用在 Amazon Aurora 中修改数据库参数组中的参数中所示的过程，通过 AWS Management Console来修改数据库集群参数。您可以使用 modify-db-cluster-parameter-group AWS CLI 命令或 ModifyDBClusterParameterGroup Amazon RDS API 操作，以编程方式修改数据库集群参数。

如果参数组已与集群关联，无需重新启动数据库集群即可修改这些参数。首次将参数组与集群关联时，需要重新启动集群。

主题

- server_audit_logging

- server_audit_events
- server_audit_incl_users
- server_audit_excl_users

server_audit_logging

启用或禁用高级审核。该参数默认为 OFF；将其设置为 ON 可启用高级审核。

日志不会显示审计数据，除非您还使用 `server_audit_events` 参数对要审计的一种或多种类型的事件进行定义。

若要确认数据库实例的审计数据是否已记录，请检查该实例的某些日志文件的名称格式是否为 `audit/audit.log.`*`other_identifying_information`*。要查看日志文件的名称，请按照 查看和列出数据库日志文件 中的过程执行操作。

server_audit_events

包含要记录的事件列表，以逗号分隔。事件必须以全大写形式指定，列表元素之间不应有空格，例如：`CONNECT,QUERY_DDL`。该参数默认为空字符串。

您可以记录以下事件的任意组合：

- CONNECT – 记录成功和失败的连接以及断开连接。此事件包括用户信息。
- QUERY – 以纯文本记录所有查询，包括由于语法或权限错误而失败的查询。

> **Tip**
>
> 启用此事件类型后，审计数据将包括有关 Aurora 自动执行的持续监控与运行状况检查信息的信息。如果只对特定类型的操作感兴趣，可以使用更具体的事件类型。您还可以使用 CloudWatch 界面，在日志中搜索与特定数据库、表或用户相关的事件。

- QUERY_DCL – 类似于 QUERY 事件，不过仅返回数据控制语言 (DCL) 查询 (GRANT、REVOKE 等)。
- QUERY_DDL – 类似于 QUERY 事件，不过仅返回数据定义语言 (DDL) 查询 (CREATE、ALTER 等)。
- QUERY_DML – 类似于 QUERY 事件，但仅返回数据操作语言 (DML) 查询（INSERT、UPDATE 等，也包括 SELECT）。
- TABLE – 记录受查询执行影响的表。

> **Note**
>
> Aurora 中没有任何筛选条件可将某些查询排除在审计日志之外。要排除 SELECT 查询，您必须排除所有 DML 语句。
> 如果某个用户在审计日志中报告了这些内部 SELECT 查询，您可以通过设置 server_audit_excl_users 数据库集群参数来排除该用户。但是，如果该用户也在其他活动中使用并且不能省略，则没有其他选项可以排除 SELECT 查询。

server_audit_incl_users

包含已记录其活动的用户的用户名列表，以逗号分隔。列表元素之间不应有空格，例如：user_3,user_4。该参数默认为空字符串。最大长度为 1024 个字符。指定用户名必须与 User 表的 mysql.user 列中的对应值匹配。有关用户名的更多信息，请参阅 MySQL 文档中的账户用户名和密码。

如果 server_audit_incl_users 和 server_audit_excl_users 均为空（默认），则审计所有用户。

如果您将用户添加到 server_audit_incl_users 并将 server_audit_excl_users 留空，则只审核这些用户。

如果将用户添加到 server_audit_excl_users 并将 server_audit_incl_users 留空，会对所有用户进行审计，但 server_audit_excl_users 中列出的用户除外。

如果将同一用户同时添加到 server_audit_excl_users 和 server_audit_incl_users，则会审计这些用户。如果两个设置中均列出同一用户，server_audit_incl_users 获得的优先级更高。

Connect 和 disconnect 事件不受此变量的影响；只要指定就始终记录这些事件。即使在 server_audit_excl_users 参数中也指定了某个用户，由于 server_audit_incl_users 具有更高的优先级，仍会记录该用户。

server_audit_excl_users

包含未记录其活动的用户的用户名列表，以逗号分隔。列表元素之间不应有空格，例如：rdsadmin,user_1,user_2。该参数默认为空字符串。最大长度为 1024 个字符。指定用户名必须与 User 表的 mysql.user 列中的对应值匹配。有关用户名的更多信息，请参阅 MySQL 文档中的账户用户名和密码。

如果 server_audit_incl_users 和 server_audit_excl_users 均为空（默认），则审计所有用户。

如果将用户添加到 server_audit_excl_users 并将 server_audit_incl_users 留空，则除在 server_audit_excl_users 中列出的那些用户外，其他所有用户均会被审计。

如果将同一用户同时添加到 server_audit_excl_users 和 server_audit_incl_users，则会审计这些用户。如果两个设置中均列出同一用户，server_audit_incl_users 获得的优先级更高。

Connect 和 disconnect 事件不受此变量的影响；只要指定就始终记录这些事件。如果在 server_audit_incl_users 参数中也指定了某个用户，由于该设置的优先级高于 server_audit_excl_users，则记录该用户。

查看审核日志

您可以使用控制台查看并下载审核日志。在 Databases (数据库) 页面上，选择该数据库实例以显示其详细信息，然后滚动到 Logs (日志) 部分。高级审计功能生成的审计日志的名称格式为 audit/audit.log.*other_identifying_information*。

要下载日志文件，请在 Logs (日志) 部分选择该文件并选择 Download (下载)。

您还可以使用 describe-db-log-files AWS CLI 命令获取日志文件列表。您可以使用 download-db-log-file-portion AWS CLI 命令下载日志文件内容。有关更多信息，请参阅“查看和列出数据库日志文件”和“下载数据库日志文件”。

审计日志详细信息

日志文件表示为 UTF-8 格式的逗号分隔变量 (CSV) 文件。查询也用单引号 (') 引起来。

审计日志单独存储在每个 Aurora MySQL 数据库实例的本地存储中。每个实例一次将写入分布在四个日志文件中。日志文件的最大大小为 100 MB。当达到这个不可配置的限制时，Aurora 将轮换文件并生成一个新文件。

> **Tip**
>
> 日志文件条目不按先后顺序排列。要对条目进行排序，请使用时间戳值。要查看最新事件，您可能需要查看所有日志文件。为了更加灵活地对日志数据进行排序和搜索，请启用设置将审计日志上传到 CloudWatch，再使用 CloudWatch 界面进行查看。

若要查看包含更多字段类型和 JSON 格式输出的审计数据，还可以使用数据库活动流功能。有关更多信息，请参阅 使用数据库活动流监控 Amazon Aurora。

审核日志文件的各行按照指定顺序包含以下逗号分隔的信息：

字段	描述
timestamp	所记录事件的 Unix 时间戳，精度为微秒。
serverhost	记录了其事件的实例的名称。
username	已连接用户的用户名。
host	用户发起连接时所在的主机。
connectionid	所记录操作的连接 ID 号。
queryid	查询 ID 号，可用于查找关系表事件和相关查询。对于 TABLE 事件，添加多行。
operation	记录的操作类型。可能值为：CONNECT、QUERY、READ、WRITE、CREATE、ALTER、RENAME 和 DROP。
database	活动数据库，由 USE 命令设置。
object	对于 QUERY 事件，此值指示数据库执行的查询。对于 TABLE 事件，它指示表名。
retcode	所记录操作的返回代码。

使用 Amazon Aurora MySQL 进行复制

Aurora MySQL 复制特征是提高集群可用性与性能的关键所在。Aurora 可以帮助您轻松创建集群并调整大小（最多可创建 15 个 Aurora 副本）。

所有副本采用相同的基础数据。如果某些数据库实例离线，其他处于可用状态的数据库实例将继续处理查询，或者在需要时作为写入器接管这些实例。Aurora 会自动将您的只读连接分配到多个数据库实例中，从而帮助 Aurora 集群更好地支持查询密集型工作负载。

在以下主题中，您可以了解 Aurora MySQL 复制的工作方式以及如何微调复制设置以获得最佳的可用性和性能。

主题

- 使用 Aurora 副本
- Amazon Aurora MySQL 的复制选项
- Amazon Aurora MySQL 复制的性能注意事项
- Amazon Aurora MySQL 的零停机重启 (ZDR)
- 使用 Aurora MySQL 配置复制筛选条件
- 监控 Amazon Aurora MySQL 复制
- 跨 AWS 区域复制 Amazon Aurora MySQL 数据库集群
- Aurora 与 MySQL 之间或 Aurora 与其他 Aurora 数据库集群之间的复制（二进制日志复制）
- 使用基于 GTID 的复制

使用 Aurora 副本

Aurora 副本是 Aurora 数据库集群中的独立终端节点，最适合用于扩展读取操作以及提高可用性。对于数据库集群在 AWS 区域中所跨的多个可用区，最多可以分配 15 个 Aurora 副本。虽然数据库集群卷由数据库集群的多个数据副本组成，但集群卷中的数据表示为数据库集群中的主实例和 Aurora 副本的单个逻辑卷。有关 Aurora 副本的更多信息，请参阅 Aurora 副本。

Aurora 副本十分适用于读取扩展，因为它们完全专用于集群卷上的读取操作。写入操作由主实例进行管理。由于集群卷是在 Aurora MySQL 数据库集群中的所有实例之间共享的，因此，无需执行额外的操作以复制每个 Aurora 副本的数据副本。相比之下，MySQL 只读副本必须在单一线程上，重放从源数据库实例向其本地数据存储的所有写入操作。此限制会影响到 MySQL 只读副本支持海量读取流量的能力。

对于 Aurora MySQL，在删除 Aurora 副本时，将立即删除其实例终端节点，并将 Aurora 副本从读取器终端节点中删除。如果在正待删除的 Aurora 副本上运行语句，则有 3 分钟宽限期。现有语句可在此宽限期内正常完成。当此宽限期结束后，将关闭并删除 Aurora 副本。

> **Important**
>
> 对于在 InnoDB 表上执行的操作，Aurora MySQL 的 Aurora 副本始终使用 `REPEATABLE READ` 默认事务隔离级别。仅对于 Aurora MySQL 数据库集群的主实例，您可以使用 `SET TRANSACTION ISOLATION LEVEL` 命令更改事务级别。该限制避免在 Aurora 副本上出现用户级锁定，并允许 Aurora 副本扩展以支持数千个活动用户连接，同时仍保持最小的副本滞后时间。

> **Note**
>
> 在主实例上运行的 DDL 语句可能会中断关联的 Aurora 副本上的数据库连接。如果 Aurora 副本连接主动使用数据库对象（如表），并且使用 DDL 语句在主实例上修改该对象，则会中断 Aurora 副本连接。

> **Note**
>
> 中国 (宁夏) 区域不支持跨区域只读副本。

Amazon Aurora MySQL 的复制选项

您可以在以下任意选项之间设置复制：

- 不同 AWS 区域中的两个 Aurora MySQL 数据库集群（通过创建 Aurora MySQL 数据库集群的跨区域只读副本）。

  有关更多信息，请参阅 跨 AWS 区域复制 Amazon Aurora MySQL 数据库集群。
- 同一 AWS 区域中的两个 Aurora MySQL 数据库集群 [通过使用 MySQL 二进制日志（binlog）复制]。

  有关更多信息，请参阅 Aurora 与 MySQL 之间或 Aurora 与其他 Aurora 数据库集群之间的复制（二进制日志复制）。

- 一个 RDS for MySQL 数据库实例（作为源）和一个 Aurora MySQL 数据库集群（通过创建 RDS for MySQL 数据库实例的 Aurora 只读副本）。

  您可以使用此方法在迁移到 Aurora 期间将现有和持续的数据更改纳入 Aurora MySQL 中。有关更多信息，请参阅 使用 Aurora 只读副本将数据从 RDS for MySQL 数据库实例迁移到 Amazon Aurora MySQL 数据库集群。

  您还可以使用此方法提高数据读取查询的可扩展性。您可以通过使用只读 Aurora MySQL 集群中的一个或多个数据库实例查询数据来执行此操作。有关更多信息，请参阅 使用 Amazon Aurora 扩展 MySQL 数据库的读取。
- 一个 AWS 区域中的一个 Aurora MySQL 数据库集群和不同区域中最多五个 Aurora 只读 Aurora MySQL 数据库集群（通过创建 Aurora 全局数据库）。

  您可以使用 Aurora 全部数据库来支持覆盖全球范围的应用程序。主 Aurora MySQL 数据库集群有一个写入器实例和最多 15 个 Aurora 副本。每一个只读辅助 Aurora MySQL 数据库集群最多可以由 16 个 Aurora 副本组成。有关更多信息，请参阅“使用 Amazon Aurora Global Database”。

> **Note**
>
> 重新引导 Amazon Aurora 数据库集群的主实例还会自动重新引导该数据库集群的 Aurora 副本，以便重新建立入口点来确保数据库集群中的读/写一致性。

Amazon Aurora MySQL 复制的性能注意事项

以下特征可以帮助您微调 Aurora MySQL 复制性能。

副本日志压缩特征自动降低复制消息的网络带宽。由于每条消息将传输到所有 Aurora 副本，因此，较大的集群的好处更大。该特征在写入方节点上产生一些 CPU 开销以执行压缩。在 Aurora MySQL 版本 2 和版本 3 中，始终启用它。

二进制日志筛选特征自动降低复制消息的网络带宽。由于 Aurora 副本不使用复制消息中包含的二进制日志信息，因此，将从发送到这些节点的消息中省略该数据。

在 Aurora MySQL 版本 2 中，您可以通过更改 `aurora_enable_repl_bin_log_filtering` 参数来控制此特征。默认情况下，该参数为 on。由于该优化应该是透明的，因此，您只能在诊断或故障排除期间禁用该设置以解决与复制相关的问题。例如，与无法使用该特征的旧 Aurora MySQL 集群的行为相符。

在 Aurora MySQL 版本 3 中，二进制日志筛选始终处于启用状态。

Amazon Aurora MySQL 的零停机重启 (ZDR)

零停机重启 (ZDR) 特征可以在某些类型的重启期间保留与数据库实例的部分或全部活动连接。ZDR 适用于 Aurora 自动执行以解决错误条件的重启，例如，当副本开始远远落后于源时。

> **Important**
>
> ZDR 机制运作以尽力而为作为原则。Aurora MySQL 版本、实例类、错误条件、兼容的 SQL 操作以及确定 ZDR 应用位置的其他因素随时可能会发生变化。

Aurora MySQL 2.x 的 ZDR 需要版本 2.10 及更高版本。ZDR 在 Aurora MySQL 3.x 的所有次要版本中都可用。在 Aurora MySQL 版本 2 和 3 中，ZDR 机制默认处于开启状态，且 Aurora 不使用 `aurora_enable_zdr` 参数。

Aurora 会在 Event (事件) 页面中报告与零停机时间重新启动相关的活动。Aurora 在尝试使用 ZDR 机制重新启动时会记录一个事件。此事件说明了 Aurora 重启的原因。然后，在重启完成后，Aurora 会记录另一个事件。这最后一个事件报告了进程所用时长，以及在重启期间保留或丢弃的连接数。您可以查看数据库错误日志，了解有关重启期间所发生情况的更多详细信息。

尽管成功执行 ZDR 操作后连接保持不变，但一些变量和特征会重新初始化。通过零停机重启进行重启后，以下类型的信息将不会保留：

- 全局变量。Aurora 将恢复会话变量，但重启后不会恢复全局变量。
- 状态变量。特别是，引擎状态报告的正常运行时间值将重置。
- `LAST_INSERT_ID`.
- 表的内存中 `auto_increment` 状态。重新初始化内存中的自动增量状态。有关自动增量值的更多信息，请参阅 MySQL 参考手册。
- 来自 `INFORMATION_SCHEMA` 和 `PERFORMANCE_SCHEMA` 表的诊断信息。这些诊断信息也会显示在 `SHOW PROFILE` 和 `SHOW PROFILES` 等命令的输出中。

下表显示了版本、实例角色以及确定在重启集群中的数据库实例时 Aurora 是否可以使用 ZDR 机制的其他情况。

Aurora MySQL version	ZDR 适用于写入器吗？	ZDR 适用于读取器吗？	ZDR 始终处于启用状态？	注意
2.x，低于 2.10.0	否	否	不适用	ZDR 不适用于这些版本。
2.10.0–2.11.0	是	是	是	Aurora 回滚活动连接上正在执行的所有事务。您的应用程序必须重试事务。 Aurora 取消任何使用 TLS/SSL、临时表、表锁定或用户锁定的连接。
2.11.1 及更高版本	是	是	是	Aurora 回滚活动连接上正在执行的所有事务。您的应用程序必须重试事务。 Aurora 取消任何使用临时表、表锁定或用户锁定的连接。
3.01–3.03	是	是	是	Aurora 回滚活动连接上正在执行的所有事务。您的应用程序必须重试事务。 Aurora 取消任何使用 TLS/SSL、临时表、表锁定或用户锁定的连接。
3.04 及更高版本	是	是	是	Aurora 回滚活动连接上正在执行的所有事务。您的应用程序必须重试事务。 Aurora 取消任何使用临时表、表锁定或用户锁定的连接。

使用 Aurora MySQL 配置复制筛选条件

您可以使用复制筛选条件来指定使用只读副本的数据库和表。复制筛选条件可以将数据库和表包含在复制之中或排除在复制之外。

以下是复制筛选条件的一些使用案例：

- 缩减只读副本的大小。使用复制筛选，您可以排除只读副本上不需要的数据库和表。

- 出于安全原因，将数据库和表从只读副本中排除。
- 在不同只读副本中为特定使用案例复制不同的数据库和表。例如，您可以使用特定的只读副本进行分析或分片。
- 对于在不同 AWS 区域中具有只读副本的数据库集群，要在不同的 AWS 区域中复制不同的数据库或表。
- 指定使用 Aurora MySQL 数据库集群复制哪些数据库和表，该集群配置为入站复制拓扑中的副本。有关此配置的更多信息，请参阅Aurora 与 MySQL 之间或 Aurora 与其他 Aurora 数据库集群之间的复制（二进制日志复制）。

主题

- 设置 Aurora MySQL 的复制筛选参数
- Aurora MySQL 的复制筛选限制
- Aurora MySQL 的复制筛选示例
- 查看只读副本的复制筛选条件

设置 Aurora MySQL 的复制筛选参数

要配置复制筛选条件，请设置以下参数：

- `binlog-do-db` –将更改复制到指定的二进制日志。为二进制日志源集群设置此参数时，仅复制在此参数中指定的二进制日志。
- `binlog-ignore-db` –不将更改复制到指定的二进制日志。为二进制日志源集群设置 `binlog-do-db` 参数时，不会评估此参数。
- `replicate-do-db` –将更改复制到指定的数据库。为二进制日志副本集群设置此参数时，仅复制在参数中指定的数据库。
- `replicate-ignore-db` –不将更改复制到指定的数据库。为二进制日志副本集群设置 `replicate-do-db` 参数时，不会评估此参数。
- `replicate-do-table` –将更改复制到指定的表。为只读副本设置此参数时，仅复制参数中指定的表。此外，设置 `replicate-do-db` 或 `replicate-ignore-db` 参数时，请确保包含指定表的数据库包含在使用二进制日志副本集群的复制中。
- `replicate-ignore-table` –不将更改复制到指定的表。为二进制日志副本集群设置 `replicate-do-table` 参数时，不会评估此参数。

- `replicate-wild-do-table` – 根据指定的数据库和表名模式复制表。支持 % 和 _ 通配符。设置 `replicate-do-db` 或 `replicate-ignore-db` 参数时，请确保包含指定表的数据库包含在使用二进制日志副本集群的复制中。
- `replicate-wild-ignore-table` –不基于指定的数据库和表名模式复制表。支持 % 和 _ 通配符。为二进制日志副本集群设置 `replicate-do-table` 或 `replicate-wild-do-table` 参数时，不会评估此参数。

将按这些参数列出的顺序对其进行评估。有关这些参数如何运行的更多信息，请参阅 MySQL 文档：

- 有关一般信息，请参阅副本服务器选项和变量。
- 有关如何评估数据库复制筛选参数的信息，请参阅评估数据库级复制和二进制日志记录选项。
- 有关如何评估表复制筛选参数的信息，请参阅评估表级复制选项。

默认情况下，这些参数中的每个参数都具有一个空值。在每个二进制日志集群上，您可以使用这些参数来设置、更改和删除复制筛选条件。设置其中一个参数时，请用逗号将各筛选条件分开。

您可以在 % 和 _ 参数中使用 `replicate-wild-do-table` 和 `replicate-wild-ignore-table` 通配符。% 通配符可以匹配任意数量的字符，而 _ 通配符只能匹配一个字符。

源数据库实例的二进制日志记录格式对于复制非常重要，因为它决定了数据更改的记录。`binlog_format` 参数的设置将决定复制是基于行还是基于语句的复制。有关更多信息，请参阅“为单可用区数据库配置 Aurora MySQL 二进制日志记录”。

> **Note**
>
> 无论源数据库实例上的 `binlog_format` 设置如何，所有数据定义语言 (DDL) 语句都将作为语句进行复制。

Aurora MySQL 的复制筛选限制

以下限制适用于 Aurora MySQL 的复制筛选：

- 仅 Aurora MySQL 版本 3 支持复制筛选条件。
- 每个复制筛选参数不得超过 2000 个字符。
- 复制筛选条件中不支持逗号。
- 复制筛选不支持 XA 事务。

有关更多信息，请参阅 MySQL 文档中的XA 事务限制。

Aurora MySQL 的复制筛选示例

要为只读副本配置复制筛选，请修改与只读副本关联的数据库集群参数组中的复制筛选参数。

> **Note**
>
> 您无法修改默认数据库集群参数组。如果只读副本使用默认参数组，请创建新的参数组并将其与只读副本关联。有关数据库集群参数组的更多信息，请参阅Amazon Aurora 的参数组。

您可以使用 AWS Management Console、AWS CLI 或 RDS API 在数据库集群参数组中设置参数。有关设置参数的信息，请参阅 在 Amazon Aurora 中修改数据库参数组中的参数。在数据库集群参数组中设置参数时，与参数组关联的所有数据库集群都使用参数设置。如果在数据库集群参数组中设置复制筛选参数，请确保参数组仅与只读副本集群关联。将源数据库实例的复制筛选参数留空。

以下示例使用 AWS CLI 设置参数。这些示例将 `ApplyMethod` 设置为 `immediate`，以便在 CLI 命令完成后立即发生参数更改。如果希望在只读副本重新启动后应用待处理的更改，请将 `ApplyMethod` 设置为 `pending-reboot`。

以下示例设置了复制筛选条件：

- Including databases in replication
- Including tables in replication
- Including tables in replication with wildcard characters
- Excluding databases from replication
- Excluding tables from replication
- Excluding tables from replication using wildcard characters

Example 将数据库包含在复制之中

以下示例将 `mydb1` 和 `mydb2` 数据库包含在复制之内。

对于 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster-parameter-group \
```

```
  --db-cluster-parameter-group-name myparametergroup \
  --parameters "ParameterName=replicate-do-
db,ParameterValue='mydb1,mydb2',ApplyMethod=immediate"
```

对于 Windows：

```
aws rds modify-db-cluster-parameter-group ^
  --db-cluster-parameter-group-name myparametergroup ^
  --parameters "ParameterName=replicate-do-
db,ParameterValue='mydb1,mydb2',ApplyMethod=immediate"
```

Example 将表包含在复制之中

以下示例将数据库 `table1` 中的 `table2` 和 `mydb1` 表包含在复制之中。

对于 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name myparametergroup \
  --parameters "ParameterName=replicate-do-
table,ParameterValue='mydb1.table1,mydb1.table2',ApplyMethod=immediate"
```

对于 Windows：

```
aws rds modify-db-cluster-parameter-group ^
  --db-cluster-parameter-group-name myparametergroup ^
  --parameters "ParameterName=replicate-do-
table,ParameterValue='mydb1.table1,mydb1.table2',ApplyMethod=immediate"
```

Example 使用通配符将表包含在复制之中

以下示例将数据库 `order` 中名称以 `return` 和 `mydb` 开头的表包含在复制之中。

对于 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name myparametergroup \
  --parameters "ParameterName=replicate-wild-do-table,ParameterValue='mydb.order
%,mydb.return%',ApplyMethod=immediate"
```

对于 Windows：

```
aws rds modify-db-cluster-parameter-group ^
  --db-cluster-parameter-group-name myparametergroup ^
  --parameters "ParameterName=replicate-wild-do-table,ParameterValue='mydb.order
%,mydb.return%',ApplyMethod=immediate"
```

Example 将数据库排除在复制之外

以下示例将 mydb5 和 mydb6 数据库排除在复制之外。

对于 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name myparametergroup \
  --parameters "ParameterName=replicate-ignore-
db,ParameterValue='mydb5,mydb6',ApplyMethod=immediate"
```

对于 Windows：

```
aws rds modify-db-cluster-parameter-group ^
  --db-cluster-parameter-group-name myparametergroup ^
  --parameters "ParameterName=replicate-ignore-
db,ParameterValue='mydb5,mydb6,ApplyMethod=immediate"
```

Example 将表排除在复制之外

以下示例将数据库 mydb5 中的表 table1 和数据库 mydb6 中的表 table2 排除在复制之外。

对于 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name myparametergroup \
  --parameters "ParameterName=replicate-ignore-
table,ParameterValue='mydb5.table1,mydb6.table2',ApplyMethod=immediate"
```

对于 Windows：

```
aws rds modify-db-cluster-parameter-group ^
```

```
  --db-cluster-parameter-group-name myparametergroup ^
  --parameters "ParameterName=replicate-ignore-
table,ParameterValue='mydb5.table1,mydb6.table2',ApplyMethod=immediate"
```

Example 使用通配符将表排除在复制之外

以下示例将数据库 `order` 中名称以 `return` 和 `mydb7` 开头的表排除在复制之外。

对于 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name myparametergroup \
  --parameters "ParameterName=replicate-wild-ignore-table,ParameterValue='mydb7.order
%,mydb7.return%',ApplyMethod=immediate"
```

对于 Windows：

```
aws rds modify-db-cluster-parameter-group ^
  --db-cluster-parameter-group-name myparametergroup ^
  --parameters "ParameterName=replicate-wild-ignore-table,ParameterValue='mydb7.order
%,mydb7.return%',ApplyMethod=immediate"
```

查看只读副本的复制筛选条件

您可以通过以下方式查看只读副本的复制筛选条件：

- 检查与只读副本关联的参数组中复制筛选参数的设置。

  有关说明，请参阅在 Amazon Aurora 中查看数据库参数组的参数值。
- 在 MySQL 客户端中，连接到只读副本并运行 `SHOW REPLICA STATUS` 语句。

  在输出中，以下字段显示了只读副本的复制筛选条件：
  - `Binlog_Do_DB`
  - `Binlog_Ignore_DB`
  - `Replicate_Do_DB`
  - `Replicate_Ignore_DB`
  - `Replicate_Do_Table`
  - `Replicate_Ignore_Table`

- `Replicate_Wild_Do_Table`
- `Replicate_Wild_Ignore_Table`

有关这些字段的更多信息，请参阅 MySQL 文档中的检查复制状态。

监控 Amazon Aurora MySQL 复制

读取扩展和高可用性依赖于尽可能短的滞后时间。您可以通过监控 Amazon CloudWatch `AuroraReplicaLag` 指标来监控 Aurora 副本滞后于 Aurora MySQL 数据库集群主实例的时间。`AuroraReplicaLag` 指标记录在每个 Aurora 副本中。

主数据库实例还记录 `AuroraReplicaLagMaximum` 和 `AuroraReplicaLagMinimum` Amazon CloudWatch 指标。`AuroraReplicaLagMaximum` 指标记录主数据库实例与数据库集群中每个 Aurora 副本之间的最大滞后量。`AuroraReplicaLagMinimum` 指标记录主数据库实例与数据库集群中每个 Aurora 副本之间的最小滞后量。

如果您需要 Aurora 副本滞后的最新值，可在 Amazon CloudWatch 中检查 `AuroraReplicaLag` 指标。有关 Aurora MySQL 数据库集群的每个 Aurora 副本的 Aurora 副本滞后也会记录在 `information_schema.replica_host_status` 表中。有关此表的更多信息，请参阅 information_schema.replica_host_status。

有关监控 RDS 实例和 CloudWatch 指标的更多信息，请参阅监控 Amazon Aurora 集群中的指标。

跨 AWS 区域复制 Amazon Aurora MySQL 数据库集群

您可以在与源数据库集群不同的 AWS 区域中创建 Amazon Aurora MySQL 数据库集群作为只读副本。采用此方法可增强灾难恢复能力，允许您将读取操作扩展到更靠近用户的 AWS 区域，并使从一个 AWS 区域迁移到另一个区域变得更轻松。

您可以创建加密和不加密数据库集群的只读副本。如果加密了源数据库集群，则必须加密只读副本。

对于每个源数据库集群，您最多可以具有 5 个作为只读副本的跨区域数据库集群。

> **Note**
>
> 作为跨区域只读副本的替代方案，您可以使用 Aurora 全局数据库以最小的延迟时间来扩展读取操作。1 个 Aurora Global Database 在 1 个 AWS 区域中有 1 个主 Aurora 数据库集群，在不同区域中最多有 10 个辅助只读数据库集群。每个辅助数据库集群最多可以包含 16 个（而不是 15）Aurora 副本。从主数据库集群到所有辅助集群的复制由 Aurora 存储层而不是由数据库引擎处理，因此复制更改的延迟最短，通常不到 1 秒。将数据库引擎排除在复制过程之外意味着数据库引擎专用于处理工作负载。这还意味着您不需要配置或管理 Aurora MySQL 二进制日志（二进制日志记录）复制。要了解更多信息，请参阅使用 Amazon Aurora Global Database。

当您在另一个 AWS 区域中创建 Aurora MySQL 数据库集群只读副本时，您应了解：

- 源数据库集群和跨区域只读副本数据库集群最多可以具有 15 个 Aurora 副本以及数据库集群的主实例。通过使用该功能，您可以扩展源 AWS 区域和复制目标 AWS 区域的读取操作。
- 在跨区域方案中，由于各 AWS 区域之间的网络通道更长，因此，源数据集群和只读副本之间的滞后时间更长。
- 跨区域复制时传输的数据可产生 Amazon RDS 数据传输费用。以下跨区域复制操作会针对传输到源 AWS 区域以外的数据收取费用：
  - 在创建只读副本时，Amazon RDS 将创建源集群的快照，并将快照传输到保存只读副本的 AWS 区域。
  - 对于源数据库中做出的每项数据修改，Amazon RDS 都会将数据从源区域传输到保存只读副本的 AWS 区域。

  有关 Amazon RDS 数据传输定价的更多信息，请参阅 Amazon Aurora 定价。
- 您可以为引用相同源数据库集群的只读副本运行多个并发创建或删除操作。不过，您必须保持在每个源数据库集群中具有 5 个只读副本的限制。

- 为了有效地进行复制，每个只读副本应具有与源数据库集群相同数量的计算和存储资源。如果扩展源数据库集群，您还应扩展只读副本。

主题

- 开始前的准备工作
- 创建 Aurora MySQL 的跨区域只读副本数据库集群
- 将只读副本提升为 Aurora MySQL 的数据库集群
- Amazon Aurora MySQL 的跨区域副本故障排除

开始前的准备工作

在创建作为跨区域只读副本的 Aurora MySQL 数据库集群之前，必须先在源 Aurora MySQL 数据库集群上开启二进制日志记录。Aurora MySQL 的跨区域复制使用 MySQL 二进制复制来重放对跨区域只读副本数据库集群的更改。

要在 Aurora MySQL 数据库集群上开启二进制日志记录，请更新源数据库集群的 `binlog_format` 参数。`binlog_format` 参数是默认集群参数组中的集群级参数。如果数据库集群使用默认的数据库集群参数组，则需创建新的数据库集群参数组来修改 `binlog_format` 设置。建议您将 `binlog_format` 设置为 `MIXED`。不过，您也可以将 `binlog_format` 设置为 `ROW` 或 `STATEMENT`(如果您需要特定的二进制日志格式)。重启您的 Aurora 数据库集群以使更改生效。

有关对 Aurora MySQL 使用二进制日志记录的更多信息，请参阅 Aurora 与 MySQL 之间或 Aurora 与其他 Aurora 数据库集群之间的复制（二进制日志复制）。有关修改 Aurora MySQL 配置参数的更多信息，请参阅 Amazon Aurora 数据库集群和数据库实例参数 和 Amazon Aurora 的参数组。

创建 Aurora MySQL 的跨区域只读副本数据库集群

您可以使用AWS Management Console、AWS Command Line Interface（AWS CLI）或 Amazon RDS API 创建作为跨区域只读副本的 Aurora 数据库集群。您可以从加密和未加密的数据库集群中创建跨区域只读副本。

在使用 AWS Management Console 为 Aurora MySQL 创建跨区域只读副本时，Amazon RDS 先在目标 AWS 区域中创建一个数据库集群，然后自动为该数据库集群创建一个数据库实例以作为主实例。

使用 AWS CLI 或 RDS API 创建跨区域只读副本时，您需要首先在目标 AWS 区域创建数据库集群，然后等待它变为活动状态。一旦处于活动状态，您就可以为该数据库集群创建一个数据库实例作为主实例。

当只读副本数据库集群的主实例可用时，复制开始。

执行以下过程从 Aurora MySQL 数据库集群创建跨区域只读副本。这些步骤适用于从加密和未加密的数据库集群中创建只读副本。

控制台

使用AWS Management Console创建作为跨区域只读副本的 Aurora MySQL 数据库集群

1. 登录AWS Management Console并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在 AWS Management Console的右上角，选择托管源数据库集群的 AWS 区域。
3. 在导航窗格中，选择 Databases（数据库）。
4. 选择要为其创建跨区域只读副本的数据库集群。
5. 对于 Actions（操作），请选择 Create cross-Region read replica（创建跨区域只读副本）。
6. 在 Create cross region read replica (创建跨区域只读副本) 页面上，选择跨区域只读副本数据库集群的选项设置，如下表中所述。

选项	描述
目标区域	选择要托管新的跨区域只读副本数据库集群的 AWS 区域。
目标数据库子网组	选择要用于跨区域只读副本数据库集群的数据库子网组。
公开访问	选择 Yes (是) 可为跨区域只读副本数据库集群提供一个公有 IP 地址；否则请选择 No (否)。
加密	请选择 Enable Encryption（启用加密）以对该数据库集群开启静态加密。有关更多信息，请参阅 加密 Amazon Aurora 资源。
AWS KMS key	仅当加密设置为启用加密时可用。选择用于加密该数据库集群的 AWS KMS key。有关更多信息，请参阅 加密 Amazon Aurora 资源。

选项	描述
数据库实例类	选择定义数据库集群中主实例的处理和内存要求的数据库实例类。有关数据库实例类选项的更多信息，请参阅Amazon Aurora 数据库实例类。
多可用区部署	选择 Yes（是），可在目标 AWS 区域的另一个可用区中创建新数据库集群的只读副本以支持故障转移。有关多可用区的详细信息，请参阅区域及可用区。
只读副本源	选择要为其创建跨区域只读副本的源数据库集群。
数据库实例标识符	键入跨区域只读副本数据库集群中主实例的名称。此标识符在新数据库集群主实例的终端节点地址中使用。 数据库实例标识符具有以下限制： • 它必须包含 1 到 63 个字母数字字符或连字符。 • 它的第一个字符必须是字母。 • 它不能以连字符结束或包含两个连续连字符。 • 它对于每个 AWS 区域每个 AWS 账户的所有数据库实例必须是唯一的。 由于跨区域只读副本数据库集群是从源数据库集群的快照创建的，因此，只读副本的主用户名和主密码与源数据库集群的主用户名和主密码相同。

选项	描述
数据库集群标识符	键入跨区域只读副本数据库集群的名称，它对于副本的目标 AWS 区域中的账户是唯一的。此标识符在数据库集群的集群端点地址中使用。有关集群端点的信息，请参阅 Amazon Aurora 端点连接。 数据库集群标识符具有以下限制： • 它必须包含 1 到 63 个字母数字字符或连字符。 • 它的第一个字符必须是字母。 • 它不能以连字符结束或包含两个连续连字符。 • 它对于每个 AWS 账户每个 AWS 区域的所有数据库集群必须是唯一的。
优先级	选择新数据库集群的主实例的故障转移优先级。此优先级决定从主实例故障恢复时提升 Aurora 副本的顺序。如果您未选择值，则默认值为 tier-1。有关更多信息，请参阅“Aurora 数据库集群的容错能力”。
数据库端口	指定应用程序和实用程序用来访问数据库的端口。Aurora 数据库集群默认为使用默认 MySQL 端口 3306。有些公司的防火墙不允许连接到此端口。如果您的公司防火墙阻止使用默认端口，请为新数据库集群选择其他端口。
增强监控	选择 Enable enhanced monitoring（启用增强监控）可开启您的数据库集群在其上运行的操作系统的实时指标收集。有关更多信息，请参阅 使用增强监控来监控操作系统指标。
监控角色	仅当增强监控设置为启用增强监控时可用。选择您创建的 IAM 角色以允许 Amazon RDS 与 Amazon CloudWatch Logs 通信，或选择默认以让 RDS 为您创建一个名为 `rds-monitoring-role` 的角色。有关更多信息，请参阅“使用增强监控来监控操作系统指标”。

选项	描述
粒度	仅当增强监控设置为启用增强监控时可用。设置为数据库集群收集指标的时间间隔（以秒为单位）。
自动次要版本升级	该设置不适用于 Aurora MySQL 数据库集群。 有关 Aurora MySQL 引擎更新的更多信息，请参阅Amazon Aurora MySQL 的数据库引擎更新。

7. 选择 Create (创建) 以创建 Aurora 的跨区域只读副本。

AWS CLI

使用 CLI 创建作为跨区域只读副本的 Aurora MySQL 数据库集群

1. 在要创建只读副本数据库集群的 AWS 区域中调用 AWS CLI create-db-cluster 命令。包括 `--replication-source-identifier` 选项并指定要创建只读副本的源数据库集群的 Amazon Resource Name (ARN)。

   对于由 `--replication-source-identifier` 标识的数据库集群已加密的跨区域复制，请指定 `--kms-key-id` 选项和 `--storage-encrypted` 选项。

   > **Note**
   >
   > 通过指定 `--storage-encrypted` 并提供 `--kms-key-id` 的值，您可以设置从未加密数据库集群到加密只读副本的跨区域复制。

   您无法指定 `--master-username` 和 `--master-user-password` 参数。这些值取自源数据库集群。

   以下代码示例从 us-west-2 区域的一个未加密数据库集群快照中创建 us-east-1 区域的只读副本。在 us-east-1 区域调用命令。此示例指定了生成主用户密码并在 Secrets Manager 中对其进行管理的 `--manage-master-user-password` 选项。有关更多信息，请参阅 使用 Amazon Aurora 和 AWS Secrets Manager 管理密码。或者，您可以使用 `--master-password` 选项自行指定和管理密码。

   对于 Linux、macOS 或 Unix：

```
aws rds create-db-cluster \
  --db-cluster-identifier sample-replica-cluster \
  --engine aurora-mysql \
  --engine-version 8.0.mysql_aurora.3.08.0 \
  --replication-source-identifier arn:aws:rds:us-
west-2:123456789012:cluster:sample-master-cluster
```

对于 Windows：

```
aws rds create-db-cluster ^
  --db-cluster-identifier sample-replica-cluster ^
  --engine aurora-mysql ^
  --engine-version 8.0.mysql_aurora.3.08.0 ^
  --replication-source-identifier arn:aws:rds:us-
west-2:123456789012:cluster:sample-master-cluster
```

以下代码示例从 us-west-2 区域的一个加密数据库集群快照中创建 us-east-1 区域的只读副本。在 us-east-1 区域调用命令。

对于 Linux、macOS 或 Unix：

```
aws rds create-db-cluster \
  --db-cluster-identifier sample-replica-cluster \
  --engine aurora-mysql \
  --engine-version 8.0.mysql_aurora.3.08.0 \
  --replication-source-identifier arn:aws:rds:us-
west-2:123456789012:cluster:sample-master-cluster \
  --kms-key-id my-us-east-1-key \
  --storage-encrypted
```

对于 Windows：

```
aws rds create-db-cluster ^
  --db-cluster-identifier sample-replica-cluster ^
  --engine aurora-mysql ^
  --engine-version 8.0.mysql_aurora.3.08.0 ^
  --replication-source-identifier arn:aws:rds:us-
west-2:123456789012:cluster:sample-master-cluster ^
  --kms-key-id my-us-east-1-key ^
  --storage-encrypted
```

`--source-region` 选项对于 AWS GovCloud（美国东部）和 AWS GovCloud（美国西部）区域之间的跨区域复制是必需的，其中由 `--replication-source-identifier` 标识的数据库集群已加密。对于 `--source-region`，指定源数据库集群的 AWS 区域。

如果未指定 `--source-region`，请指定 `--pre-signed-url` 值。预签名 URL 包含签名版本 4 签名的请求，该请求用于在源 `create-db-cluster`中调用的 AWS 区域 命令。要了解有关 `pre-signed-url` 选项的更多信息，请参阅《AWS CLI 命令参考》中的 create-db-cluster。

2. 使用 AWS CLI describe-db-clusters 命令检查数据库集群是否已变为可用状态，如以下示例中所示。

```
aws rds describe-db-clusters --db-cluster-identifier sample-replica-cluster
```

当 **`describe-db-clusters`** 结果显示状态 `available` 时，创建数据库集群的主实例以便复制能够开始。为此，请使用 AWS CLI create-db-instance 命令，如以下示例中所示。

对于 Linux、macOS 或 Unix：

```
aws rds create-db-instance \
  --db-cluster-identifier sample-replica-cluster \
  --db-instance-class db.r5.large \
  --db-instance-identifier sample-replica-instance \
  --engine aurora-mysql
```

对于 Windows：

```
aws rds create-db-instance ^
  --db-cluster-identifier sample-replica-cluster ^
  --db-instance-class db.r5.large ^
  --db-instance-identifier sample-replica-instance ^
  --engine aurora-mysql
```

在数据库实例已创建并可用后，复制将开始。您可以调用 AWS CLI describe-db-instances 命令以确定数据库实例是否可用。

RDS API

使用 API 创建作为跨区域只读副本的 Aurora MySQL 数据库集群

1. 在要创建只读副本数据库集群的 AWS 区域中调用 RDS API CreateDBCluster 操作。包括 `ReplicationSourceIdentifier` 参数并指定要创建只读副本的源数据库集群的 Amazon Resource Name (ARN)。

   对于由 `ReplicationSourceIdentifier` 标识的数据库集群已加密的跨区域复制，应指定 `KmsKeyId` 参数并将 `StorageEncrypted` 参数设置为 `true`。

   > **Note**
   >
   > 您可以将 `StorageEncrypted` 指定为 **`true`** 并提供 `KmsKeyId` 值，以设置从未加密数据库集群到加密只读副本的跨区域复制。在这种情况下，不需要指定 `PreSignedUrl`。

   您无需包含 `MasterUsername` 和 `MasterUserPassword` 参数，因为这些值是从源数据库集群中获取的。

   以下代码示例从 us-west-2 区域的一个未加密数据库集群快照中创建 us-east-1 区域的只读副本。在 us-east-1 区域调用操作。

   ```
   https://rds.us-east-1.amazonaws.com/
     ?Action=CreateDBCluster
     &ReplicationSourceIdentifier=arn:aws:rds:us-west-2:123456789012:cluster:sample-
   master-cluster
     &DBClusterIdentifier=sample-replica-cluster
     &Engine=aurora-mysql
     &SignatureMethod=HmacSHA256
     &SignatureVersion=4
     &Version=2014-10-31
     &X-Amz-Algorithm=AWS4-HMAC-SHA256
     &X-Amz-Credential=AKIADQKE4SARGYLE/20161117/us-east-1/rds/aws4_request
     &X-Amz-Date=20160201T001547Z
     &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
     &X-Amz-Signature=a04c831a0b54b5e4cd236a90dcb9f5fab7185eb3b72b5ebe9a70a4e95790c8b7
   ```

   以下代码示例从 us-west-2 区域的一个加密数据库集群快照中创建 us-east-1 区域的只读副本。在 us-east-1 区域调用操作。

```
https://rds.us-east-1.amazonaws.com/
  ?Action=CreateDBCluster
  &KmsKeyId=my-us-east-1-key
  &StorageEncrypted=true
  &PreSignedUrl=https%253A%252F%252Frds.us-west-2.amazonaws.com%252F
         %253FAction%253DCreateDBCluster
         %2526DestinationRegion%253Dus-east-1
         %2526KmsKeyId%253Dmy-us-east-1-key
         %2526ReplicationSourceIdentifier%253Darn%25253Aaws%25253Ards%25253Aus-
west-2%25253A123456789012%25253Acluster%25253Asample-master-cluster
         %2526SignatureMethod%253DHmacSHA256
         %2526SignatureVersion%253D4
         %2526Version%253D2014-10-31
         %2526X-Amz-Algorithm%253DAWS4-HMAC-SHA256
         %2526X-Amz-Credential%253DAKIADQKE4SARGYLE%252F20161117%252Fus-
west-2%252Frds%252Faws4_request
         %2526X-Amz-Date%253D20161117T215409Z
         %2526X-Amz-Expires%253D3600
         %2526X-Amz-SignedHeaders%253Dcontent-type%253Bhost%253Buser-agent%253Bx-
amz-content-sha256%253Bx-amz-date
         %2526X-Amz-Signature
%253D255a0f17b4e717d3b67fad163c3ec26573b882c03a65523522cf890a67fca613
  &ReplicationSourceIdentifier=arn:aws:rds:us-west-2:123456789012:cluster:sample-
master-cluster
  &DBClusterIdentifier=sample-replica-cluster
  &Engine=aurora-mysql
  &SignatureMethod=HmacSHA256
  &SignatureVersion=4
  &Version=2014-10-31
  &X-Amz-Algorithm=AWS4-HMAC-SHA256
  &X-Amz-Credential=AKIADQKE4SARGYLE/20161117/us-east-1/rds/aws4_request
  &X-Amz-Date=20160201T001547Z
  &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
  &X-Amz-Signature=a04c831a0b54b5e4cd236a90dcb9f5fab7185eb3b72b5ebe9a70a4e95790c8b7
```

对于 AWS GovCloud（美国东部）和 AWS GovCloud（美国西部）区域之间的跨区域复制（其中由 `ReplicationSourceIdentifier` 标识的数据库集群已加密），还应指定 `PreSignedUrl` 参数。预签名 URL 必须是对 `CreateDBCluster` API 操作的有效请求，该操作能够在包含要复制的加密数据库集群的源 AWS 区域中执行。KMS 密钥标识符用于加密只读副本，并且必须是对目标 AWS 区域有效的 KMS 密钥。要自动而非手动生成预签名的 URL，请改用具有 `--source-region` 选项的 AWS CLI create-db-cluster 命令。

2. 如下例所示，通过使用 RDS API DescribeDBClusters 操作检查数据库集群是否可用。

```
https://rds.us-east-1.amazonaws.com/
  ?Action=DescribeDBClusters
  &DBClusterIdentifier=sample-replica-cluster
  &SignatureMethod=HmacSHA256
  &SignatureVersion=4
  &Version=2014-10-31
  &X-Amz-Algorithm=AWS4-HMAC-SHA256
  &X-Amz-Credential=AKIADQKE4SARGYLE/20161117/us-east-1/rds/aws4_request
  &X-Amz-Date=20160201T002223Z
  &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
  &X-Amz-Signature=84c2e4f8fba7c577ac5d820711e34c6e45ffcd35be8a6b7c50f329a74f35f426
```

当 DescribeDBClusters 结果显示状态 available 时，创建数据库集群的主实例以便复制能够开始。为此，请使用 RDS API CreateDBInstance 操作，如以下示例所示。

```
https://rds.us-east-1.amazonaws.com/
  ?Action=CreateDBInstance
  &DBClusterIdentifier=sample-replica-cluster
  &DBInstanceClass=db.r5.large
  &DBInstanceIdentifier=sample-replica-instance
  &Engine=aurora-mysql
  &SignatureMethod=HmacSHA256
  &SignatureVersion=4
  &Version=2014-10-31
  &X-Amz-Algorithm=AWS4-HMAC-SHA256
  &X-Amz-Credential=AKIADQKE4SARGYLE/20161117/us-east-1/rds/aws4_request
  &X-Amz-Date=20160201T003808Z
  &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
  &X-Amz-Signature=125fe575959f5bbcebd53f2365f907179757a08b5d7a16a378dfa59387f58cdb
```

在数据库实例已创建并可用后，复制将开始。您可以调用 AWS CLI DescribeDBInstances 命令以确定数据库实例是否可用。

查看 Amazon Aurora MySQL 跨区域副本

您可以调用 describe-db-clusters AWS CLI 命令或 DescribeDBClusters RDS API 操作以查看 Amazon Aurora MySQL 数据库集群的跨区域复制关系。在响应中，请参阅

`ReadReplicaIdentifiers` 字段，获取任何跨区域只读副本数据库集群的数据库集群标识符。请参阅 `ReplicationSourceIdentifier` 元素，获取作为复制源的源数据库集群的 ARN。

将只读副本提升为 Aurora MySQL 的数据库集群

可将 Aurora MySQL 只读副本提升为独立的数据库集群。提升 Aurora MySQL 只读副本时，其数据库实例将在重启后变得可用。

通常，如果源数据库集群出现故障，可以作为数据恢复方案将 Aurora MySQL 只读副本提升为单独数据库集群。

为此，请先创建只读副本，然后监控源数据库集群的故障。如果出现故障，请执行以下操作：

1. 提升只读副本。
2. 将数据库流量定向到提升的数据库集群。
3. 将提升的数据库集群作为源，创建替代只读副本。

在提升只读副本时，只读副本将成为独立的 Aurora 数据库集群。提升过程可能需要几分钟或更长时间才能完成，具体时间取决于只读副本的大小。将只读副本提升为新数据库集群后，它将与任何其他数据库集群相同。例如，可从中创建只读副本并执行时间点还原操作。还可以为数据库集群创建 Aurora 副本。

由于经过提升的数据库集群不再是只读副本，因此不能再使用它作为复制目标。

以下步骤说明将只读副本提升为数据库集群的一般过程：

1. 停止向只读副本源数据库集群写入任何事务，然后等待对只读副本完成所有更新。在源数据库集群完成数据库更新后，只读副本才会进行数据库更新，且这种复制滞后可能会有很大差异。使用 `ReplicaLag` 指标确定只读副本完成所有更新的时间。`ReplicaLag` 指标记录只读副本数据库实例滞后于源数据库实例的时间量。当 `ReplicaLag` 指标达到 `0` 时，即表示只读副本已赶上源数据库实例进度。
2. 使用 Amazon RDS 控制台上的 Promote（提升）选项、AWS CLI 命令 promote-read-replica-db-cluster 或 PromoteReadReplicaDBCluster Amazon RDS API 操作来提升只读副本。

   选择 Aurora MySQL 数据库实例来提升只读副本。在提升只读副本后，Aurora MySQL 数据库集群将被提升为单独数据库集群。故障转移优先级最高的数据库实例将被提升为数据库集群的主数据库实例。其他数据库实例将变为 Aurora 副本。

> **Note**
>
> 提升过程需要几分钟才能完成。在提升只读副本时，会停止复制并重启数据库实例。完成重启后，只读副本即可作为新数据库集群使用。

控制台

将 Aurora MySQL 只读副本提升为数据库集群

1. 登录AWS Management Console并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在控制台中，选择 Instances (实例)。

   此时将显示实例窗格。
3. 在 Instances (实例) 窗格中，选择要提升的只读副本。

   只读副本将显示为 Aurora MySQL 数据库实例。
4. 对于操作，选择提升只读副本。
5. 在确认页面上，选择 Promote read replica（提升只读副本）。

AWS CLI

要将只读副本升级为数据库集群，请使用 AWS CLI promote-read-replica-db-cluster 命令。

Example

对于 Linux、macOS 或 Unix：

```
aws rds promote-read-replica-db-cluster \
    --db-cluster-identifier mydbcluster
```

对于 Windows：

```
aws rds promote-read-replica-db-cluster ^
    --db-cluster-identifier mydbcluster
```

RDS API

要将只读副本提升为数据库集群，请调用 PromoteReadReplicaDBCluster。

Amazon Aurora MySQL 的跨区域副本故障排除

以下是您在创建 Amazon Aurora 跨区域只读副本时可能遇到的常见错误消息的列表以及指定错误的解决方法。

源集群 [数据库集群 ARN] 未启用二进制日志

要解决该问题，请对源数据库集群开启二进制日志记录。有关更多信息，请参阅 开始前的准备工作。

源集群 [数据库集群 ARN] 没有在写入器上同步的集群参数组

如果您已更新 `binlog_format` 数据库集群参数，但尚未重启数据库集群的主实例，则会收到该错误。重启数据库集群的主实例 (即写入器) 并重试。

源集群 [数据库集群 ARN] 已具有此区域中的只读副本

对于任意 AWS 区域中的每个源数据库集群，您最多可以具有 5 个作为只读副本的跨区域数据库集群。如果在特定 AWS 区域中，您已有数据库集群的最大数量只读副本，则必须先删除一个现有副本，然后才能在该区域中创建新的跨区域数据库集群。

数据库集群 [数据库集群 ARN] 需要数据库引擎升级来支持跨区域复制

要解决该问题，请将源数据库集群中的所有实例的数据库引擎版本升级到最新的数据库引擎版本，然后再次尝试创建跨区域只读副本数据库。

Aurora 与 MySQL 之间或 Aurora 与其他 Aurora 数据库集群之间的复制（二进制日志复制）

Amazon Aurora MySQL 与 MySQL 兼容，因此您可以在 MySQL 数据库与 Amazon Aurora MySQL 数据库集群之间设置复制。此类复制使用 MySQL 二进制日志复制，也称为二进制日志复制。如果将二进制日志复制与 Aurora 结合使用，建议您的 MySQL 数据库运行 MySQL 版本 5.5 或更高版本。您可以在 Aurora MySQL 数据库集群为复制源或副本的位置设置复制。您可以使用 Amazon RDS MySQL 数据库实例、Amazon RDS 外部的 MySQL 数据库或其他 Aurora MySQL 数据库集群进行复制。

> **Note**
>
> 您不能使用二进制日志复制以复制到某些类型的 Aurora 数据库集群或从中进行复制。特别是，二进制日志复制不可用于 Aurora Serverless v1 集群。如果 `SHOW MASTER STATUS`

和 SHOW SLAVE STATUS（Aurora MySQL 版本 2）或 SHOW REPLICA STATUS（Aurora MySQL 版本 3）语句不返回任何输出，请检查您使用的集群是否支持二进制日志复制。

您还可以在其他 AWS 区域中的 RDS for MySQL 数据库实例或 Aurora MySQL 数据库集群之间进行复制。跨 AWS 区域执行复制时，请确保您的数据库集群和数据库实例可公开访问。如果 Aurora MySQL 数据库集群位于您的 VPC 的私有子网中，请在 AWS 区域之间使用 VPC 对等。有关更多信息，请参阅 VPC 中的数据库集群由另一 VPC 中的 EC2 实例访问。

如果要在 Aurora MySQL 数据库集群和另一个 AWS 区域中的 Aurora MySQL 数据库集群之间配置复制，您可以在与源数据库集群不同的 AWS 区域 中创建一个 Aurora MySQL 数据库集群作为只读副本。有关更多信息，请参阅 跨 AWS 区域复制 Amazon Aurora MySQL 数据库集群。

使用 Aurora MySQL 版本 2 和 3，您可以在 Aurora MySQL 与为复制使用全局事务标识符（GTID）的外部源或目标之间进行复制。请确保 Aurora MySQL 数据库集群中的 GTID 相关参数具有与外部数据库的 GTID 状态兼容的设置。若要了解如何执行此操作，请参阅使用基于 GTID 的复制。在 Aurora MySQL 版本 3.01 及更高版本中，您可以选择如何将 GTID 分配给从不使用 GTID 的源复制的事务。有关控制该设置的存储过程的信息，请参阅 mysql.rds_assign_gtids_to_anonymous_transactions（Aurora MySQL 版本 3）。

Warning

在 Aurora MySQL 与 MySQL 之间复制时，请确保仅使用 InnoDB 表。如果有 MyISAM 表需要复制，可以在设置复制之前使用以下命令将其转换为 InnoDB。

```
alter table <schema>.<table_name> engine=innodb, algorithm=copy;
```

在以下各节中，设置复制、停止复制、扩展数据库的读取、优化二进制日志复制以及设置增强型二进制日志。

主题

- 为 Aurora MySQL 设置二进制日志复制
- 停止 Aurora MySQL 的二进制日志复制
- 使用 Amazon Aurora 扩展 MySQL 数据库的读取
- 优化 Aurora MySQL 的二进制日志复制
- 为 Aurora MySQL 设置增强型二进制日志

为 Aurora MySQL 设置二进制日志复制

设置使用 Aurora MySQL 进行 MySQL 复制包括以下步骤（下面将详细讨论）：

目录

- 1. 在复制源上开启二进制日志记录
- 2. 在复制源上保留二进制日志，直到不再需要
- 3. 创建复制源的副本或转储
- 4. 将转储加载到副本目标（如果需要）
- 5. 在您的复制源上创建复制用户
- 6. 在副本目标上开启复制
  - 设置停止复制到只读副本的位置
- 7. 监控副本
- 在复制源和目标之间同步密码

1. 在复制源上开启二进制日志记录

在下面查找有关如何在数据库引擎的复制源上开启二进制日志记录的说明。

数据库引擎	说明
Aurora MySQL	在 Aurora MySQL 数据库集群上开启二进制日志记录 将 `binlog_format` 数据库集群参数设置为 `ROW`、`STATEMENT` 或 `MIXED`。建议设置为 `MIXED`，除非您需要采用特定的二进制日志格式。（默认值为 `OFF`。） 要更改 `binlog_format` 参数，请创建自定义数据库集群参数组，并将该自定义参数组与您的数据库集群关联。您无法更改默认数据库集群参数组中的参数。 如果要将 `binlog_format` 参数从 `OFF` 更改为其他值，请重启 Aurora 数据库集群以使更改生效。 有关更多信息，请参阅Amazon Aurora 数据库集群和数据库实例参数 和Amazon Aurora 的参数组。

数据库引擎	说明
RDS for MySQL	在 Amazon RDS 数据库实例上开启二进制日志记录 您不能直接为 Amazon RDS 数据库实例开启二进制日志记录，但可以通过执行以下操作之一来开启它： • 为数据库实例开启自动备份。您可以在创建数据库实例时开启自动备份，也可以通过修改现有数据库实例开启备份。有关更多信息，请参阅 Amazon RDS 用户指南 中的创建数据库实例。 • 为数据库实例创建只读副本。有关更多信息，请参阅 Amazon RDS 用户指南 中的使用只读副本。

数据库引擎	说明
MySQL (外部)	设置加密复制 要使用 Aurora MySQL 版本 2 安全地复制数据，您可以使用加密复制。 **Note** 如果您不需要使用加密复制，可以跳过以下步骤。 以下是使用加密复制的先决条件： • 必须在外部 MySQL 源数据库上启用安全套接字层 (SSL)。 • 必须为 Aurora MySQL 数据库集群准备客户端密钥和客户端证书。 在加密复制期间，Aurora MySQL 数据库集群充当 MySQL 数据库服务器的客户端。Aurora MySQL 客户端的证书和密钥必须是 .pem 格式的文件。 1. 确保您为加密复制做好以下准备： • 如果您没有在外部 MySQL 源数据库上启用 SSL，并且没有准备客户端密钥和客户端证书，请在 MySQL 数据库服务器上开启 SSL 并生成所需的客户端密钥和客户端证书。 • 如果在外部源上启用了 SSL，请提供 Aurora MySQL 数据库集群的客户端密钥和证书。如果未提供，请为 Aurora MySQL 数据库集群生成新密钥和证书。要对客户端证书进行签名，您必须拥有您用来在外部 MySQL 源数据库上配置 SSL 的证书颁发机构密钥。 有关更多信息，请参阅 MySQL 文档中的使用 openssl 创建 SSL 证书和密钥。 您需要证书颁发机构证书、客户端密钥和客户端证书。 2. 通过 SSL 以主用户身份连接到 Aurora MySQL 数据库集群。 有关通过 SSL 连接到 Aurora MySQL 数据库集群的信息，请参阅与 Aurora MySQL 数据库集群的 TLS 连接。

数据库引擎	说明

3. 运行 `mysql.rds_import_binlog_ssl_material` 存储过程，以将 SSL 信息导入到 Aurora MySQL 数据库集群中。

   对于 `ssl_material_value` 参数，将 Aurora MySQL 数据库集群的 .pem 格式文件中的信息插入到正确的 JSON 负载中。

   以下示例将 SSL 信息导入到 Aurora MySQL 数据库集群中。在 .pem 格式文件中，主体代码通常比示例中显示的主体代码长。

```
call mysql.rds_import_binlog_ssl_material(
'{"ssl_ca":"-----BEGIN CERTIFICATE-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pcj
qP3maAhDFcvBS7O6V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96
xbiFveSFJuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/
i8SeJtjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz22
1CBt5IMucxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END CERTIFICATE-----\n","ssl_cert":"-----BEGIN CERTIFICA
TE-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pcj
qP3maAhDFcvBS7O6V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96
xbiFveSFJuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/
i8SeJtjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz22
1CBt5IMucxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END CERTIFICATE-----\n","ssl_key":"-----BEGIN RSA PRIVATE
 KEY-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pc
jqP3maAhDFcvBS7O6V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSF
JuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJ
tjnV3iAoG/cQk+0FzZ
```

数据库引擎	说明

```
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz221CBt5IMu
cxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END RSA PRIVATE KEY-----\n"}');
```

有关更多信息，请参阅mysql.rds_import_binlog_ssl_material 和与 Aurora MySQL 数据库集群的 TLS 连接。

> **Note**
>
> 运行该过程后，密钥会存储在文件中。稍后如果要删除文件，您可以运行 mysql.rds_remove_binlog_ssl_material 存储过程。

在外部 MySQL 数据库上开启二进制日志记录

1. 从命令 Shell 中停止 mysql 服务。

```
sudo service mysqld stop
```

2. 编辑 my.cnf 文件（此文件通常位于 /etc 下）。

```
sudo vi /etc/my.cnf
```

将 log_bin 和 server_id 选项添加到 [mysqld] 节。log_bin 选项为二进制日志文件提供文件名标识符。server_id 选项为源-副本关系中的服务器提供唯一标识符。

如果不需要加密复制，请确保先为外部 MySQL 数据库启用二进制日志并关闭 SSL。

以下是 /etc/my.cnf 文件中未加密数据的相关条目。

```
log-bin=mysql-bin
server-id=2133421
innodb_flush_log_at_trx_commit=1
```

数据库引擎	说明

```
sync_binlog=1
```

如果需要加密复制，请确保先为外部 MySQL 数据库启用 SSL 和二进制日志。

`/etc/my.cnf` 文件的条目包括 MySQL 数据库服务器的 .pem 文件位置。

```
log-bin=mysql-bin
server-id=2133421
innodb_flush_log_at_trx_commit=1
sync_binlog=1

# Setup SSL.
ssl-ca=/home/sslcerts/ca.pem
ssl-cert=/home/sslcerts/server-cert.pem
ssl-key=/home/sslcerts/server-key.pem
```

此外，必须将 MySQL 数据库实例的 `sql_mode` 选项设置为 0，否则不得将该选项包含在 my.cnf 文件中。

当连接到外部 MySQL 数据库时，记录外部 MySQL 数据库的二进制日志位置。

```
mysql> SHOW MASTER STATUS;
```

您的输出应类似于以下内容：

```
+------------------+----------+--------------+------------------
+-------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB |
 Executed_Gtid_Set |
+------------------+----------+--------------+------------------
+-------------------+
| mysql-bin.000031 |      107 |              |                  |
              |
+------------------+----------+--------------+------------------
+-------------------+
1 row in set (0.00 sec)
```

数据库引擎	说明
	有关更多信息，请参阅 MySQL 文档中的 Setting the replication source configuration。 3. 启动 mysql 服务。 `sudo service mysqld start`

2. 在复制源上保留二进制日志，直到不再需要

使用 MySQL 二进制日志复制时，Amazon RDS 不会对复制过程进行管理。因此，您需要确保复制源上的二进制日志文件保留到更改应用于副本之后。保留这些文件有助于在发生故障时可还原您的源数据库。

使用以下说明为数据库引擎保留二进制日志。

数据库引擎	说明
Aurora MySQL	在 Aurora MySQL 数据库集群上保留二进制日志 您无法访问 Aurora MySQL 数据库集群的二进制日志文件。因此，您必须为复制源上的二进制日志文件选择足够长的保留时间，从而确保在 Amazon RDS 删除二进制日志文件之前将更改应用于副本。Aurora MySQL 数据库集群上的二进制日志文件最多可以保留 90 天。 如果所设置的复制以 MySQL 数据库或 RDS for MySQL 数据库实例作为副本，并且要创建副本的数据库非常大，请为二进制日志文件选择较长的保留时间范围，直到数据库到副本的初始复制完成，并且副本滞后达到 0。 要设置二进制日志保留时间范围，请使用 mysql.rds_set_configuration 过程，并指定 `'binlog retention hours'` 配置参数以及在数据库集群上保留二进制日志文件的小时数。Aurora MySQL 版本 2.11.0 及更高版本和版本 3 的最大值为 2160（90天）。 以下示例将二进制日志文件的保留期设置为 6 天：

数据库引擎	说明
	`CALL mysql.rds_set_configuration('binlog retention hours', 144);` 复制开始之后，可以通过对副本运行 `SHOW SLAVE STATUS`（Aurora MySQL 版本 2）或 `SHOW REPLICA STATUS`（Aurora MySQL 版本 3）命令并检查 `Seconds behind master` 字段来验证更改是否已经应用于副本。如果 `Seconds behind master` 字段为 0，则表示不存在副本滞后。不存在副本滞后时，请通过将 `binlog retention hours` 配置参数设置为更短的时间范围缩短保留二进制日志文件的时间。 如果未指定此设置，则 Aurora MySQL 的默认值为 24 小时（1 天）。 如果您为 `'binlog retention hours'` 指定的值大于最大值，则 Aurora MySQL 使用最大值。
RDS for MySQL	在 Amazon RDS 数据库实例上保留二进制日志 您也可以通过设置二进制日志保留小时数在 Amazon RDS 数据库实例上保留二进制日志文件，就如同 Aurora MySQL 数据库集群一样（如上一行所述）。 还可以通过为数据库实例创建只读副本，在 Amazon RDS 数据库实例上保留二进制日志文件。此只读副本是临时副本，仅用于保留二进制日志文件的目的。在创建只读副本后，在只读副本上调用 mysql.rds_stop_replication 过程。复制停止时，Amazon RDS 不会删除复制源上的任何二进制日志文件。设置与永久副本间的复制之后，可以在复制源与永久副本之间的副本滞后（`Seconds behind master` 字段）达到 0 时删除只读副本。
MySQL (外部)	在外部 MySQL 数据库上保留二进制日志 外部 MySQL 数据库上的二进制日志文件不由 Amazon RDS 进行管理，因此，它们会一直保留，直到由您删除。 复制开始之后，可以通过对副本运行 `SHOW SLAVE STATUS`（Aurora MySQL 版本 2）或 `SHOW REPLICA STATUS`（Aurora MySQL 版本 3）命令并检查 `Seconds behind master` 字段来验证更改是否已经应用于副本。如果 `Seconds behind master` 字段为 0，则表示不存在副本滞后。不存在副本滞后时，可以删除旧的二进制日志文件。

3. 创建复制源的副本或转储

使用复制源的快照、克隆或转储将数据的基准副本加载到您的副本。然后从该点开始复制。

使用以下说明创建数据库引擎的复制源的副本或转储。

数据库引擎	说明
Aurora MySQL	创建 Aurora MySQL 数据库集群的副本 使用以下方法之一： • 从数据库集群快照还原： 1. 创建 Amazon Aurora 数据库集群的数据库集群快照。有关更多信息，请参阅“创建数据库集群快照”。 2. 通过从刚创建的数据库集群快照进行还原来创建新的 Aurora 数据库集群。 确保为存储的数据库集群保留的数据库参数组与为原始数据库集群保留的相同。这样可确保数据库集群的副本已启用二进制日志记录。有关更多信息，请参阅 从数据库集群快照还原。 • 克隆您的数据库集群。有关更多信息，请参阅 克隆 Amazon Aurora 数据库集群卷。 确定二进制日志文件名和位置 使用以下方法之一： • 在 AWS Management Console 中： 1. 选择数据库，然后选择新 Aurora 数据库集群的主实例（写入器）来显示其详细信息。 2. 滚动到近期事件。显示的事件消息包括二进制日志文件名和位置。事件消息采用以下格式。 `Binlog position from crash recovery is binlog-file-name  binlog-position` 3. 在开始复制时，保存二进制日志文件名和位置的值。 • 调用 describe-events AWS CLI 命令，如以下示例所示。

数据库引擎	说明

```
aws rds describe-events

{
    "Events": [
        {
            "EventCategories": [],
            "SourceType": "db-instance",
            "SourceArn": "arn:aws:rds:us-west-2:123456789012:
db:sample-restored-instance",
            "Date": "2016-10-28T19:43:46.862Z",
            "Message": "Binlog position from crash recovery is mysql-
bin-changelog.000003 4278",
            "SourceIdentifier": "sample-restored-instance"
        }
    ]
}
```

- 在 MySQL 错误日志中检查上一个 MySQL 二进制日志文件位置。

创建 Aurora MySQL 数据库集群的转储

如果副本目标是外部 MySQL 数据库或 RDS for MySQL 数据库实例，则必须从 Aurora 数据库集群创建转储文件。

请务必针对您创建的源数据库集群的副本运行 `mysqldump` 命令。这是为了避免在进行转储时考虑锁定问题。如果转储是直接在源数据库集群上进行的，则需要锁定源表，用于防止在转储过程中对这些表进行并发写入。

1. 使用 MySQL 客户端连接到数据库集群。
2. 发出 `mysqldump` 命令。例如：

```
PROMPT> mysqldump --databases database_name --single-transaction
--order-by-primary -r backup.sql -u local_user s -p
```

3. 创建转储文件后，可以删除数据库集群副本。

数据库引擎	说明
RDS for MySQL	创建 Amazon RDS 数据库实例的快照 创建 Amazon RDS 数据库实例的只读副本。有关更多信息，请参阅 Amazon Relational Database Service 用户指南 中的创建只读副本。 1. 连接到只读副本，然后运行 mysql.rds_stop_replication 过程以停止复制。 2. 当只读副本处于 Stopped（已停止）时，连接到只读副本并运行 SHOW SLAVE STATUS（Aurora MySQL 版本 2）或 SHOW REPLICA STATUS（Aurora MySQL 版本 3）命令。从 Relay_Master_Log_File 字段检索当前二进制日志文件名，并从 Exec_Master_Log_Pos 字段检索日志文件位置。在开始复制时保存这些值。 3. 在只读副本保持 Stopped (已停止) 状态期间，创建只读副本的数据库快照。有关更多信息，请参阅 Amazon Relational Database Service 用户指南 中的创建数据库快照。 4. 删除只读副本。

MySQL (外部)	创建外部 MySQL 数据库的转储
	
	1. 创建转储之前，您需要确保转储的二进制日志位置随源实例中的数据保持最新状态。为此，必须先使用以下命令停止对实例进行的任何写入操作：
	
	`mysql> FLUSH TABLES WITH READ LOCK;`
	
	2. 使用 mysqldump 命令创建 MySQL 数据库的转储，如下所示：
	
	`PROMPT> sudo mysqldump --databases database_name --master-data=2 --single-transaction \`
	`--order-by-primary -r backup.sql -u local_user -p`
	
	3. 创建转储之后，请使用以下命令解锁 MySQL 数据库中的表：
	
	`mysql> UNLOCK TABLES;`

4. 将转储加载到副本目标（如果需要）

如果您打算从 Amazon RDS 外部的 MySQL 数据库转储加载数据，则可能要创建用于将转储文件复制到其中的 EC2 实例。然后，可以从该 EC2 实例将数据加载到数据库集群或数据库实例中。使用此方法时，您可以在将转储文件复制到 EC2 实例之前压缩它们，以便降低与向 Amazon RDS 复制数据关联的网络成本。在网络中进行传输时，还可以加密转储文件以保护数据。

> **Note**
>
> 如果您创建一个新的 Aurora MySQL 数据库集群作为副本目标，则不需要加载转储文件：
>
> - 可以从数据库集群快照进行还原来创建新的数据库集群。有关更多信息，请参阅 从数据库集群快照还原。
> - 可以克隆源数据库集群来创建新的数据库集群。有关更多信息，请参阅 克隆 Amazon Aurora 数据库集群卷。
> - 可以将数据从数据库实例快照迁移到新的数据库集群中。有关更多信息，请参阅 将数据迁移到 Amazon Aurora MySQL 数据库集群。

使用以下说明将复制源的转储加载到数据库引擎的副本目标中。

数据库引擎	说明

Aurora MySQL	将转储加载到 Aurora MySQL 数据库集群
	1. 将 `mysqldump` 命令的输出从复制源复制到也可以连接到 Aurora MySQL 数据库集群的位置。
	2. 使用 `mysql` 命令连接到 Aurora MySQL 数据库集群。以下是示例。
	`PROMPT> mysql -h host_name -port=3306 -u db_master_user -p`
	3. 在 `mysql` 提示符下，运行 `source` 命令并向它传递您的数据库转储文件名，以便将数据加载到 Aurora MySQL 数据库集群中，例如：
	`mysql> source backup.sql;`

数据库引擎	说明

RDS for MySQL	将转储加载到 Amazon RDS 数据库实例
	1. 将 mysqldump 命令的输出从复制源复制到也可以连接到 MySQL 数据库实例的位置。
	2. 使用 mysql 命令连接到 MySQL 数据库实例。以下是示例。
	`PROMPT> mysql -h host_name -port=3306 -u db_master_user -p`
	3. 在 mysql 提示符下，运行 source 命令并向它传递您的数据库转储文件名，以便将数据加载到 MySQL 数据库实例中，例如：
	`mysql> source backup.sql;`

MySQL (外部)	将转储加载到外部 MySQL 数据库
	您无法将数据库快照或数据库集群快照加载到外部 MySQL 数据库中。您必须使用 mysqldump 命令的输出。
	1. 将 mysqldump 命令的输出从复制源复制到也可以连接到 MySQL 数据库的位置。
	2. 使用 mysql 命令连接到 MySQL 数据库。以下是示例。
	`PROMPT> mysql -h host_name -port=3306 -u db_master_user -p`
	3. 在 mysql 提示符下，运行 source 命令并向它传递您的数据库转储文件名，以便将数据加载到 MySQL 数据库中。以下是示例。
	`mysql> source backup.sql;`

5. 在您的复制源上创建复制用户

在源上创建一个专用于复制的用户 ID。以下示例适用于 RDS for MySQL 或外部 MySQL 源数据库。

```
mysql> CREATE USER 'repl_user'@'domain_name' IDENTIFIED BY 'password';
```

对于 Aurora MySQL 源数据库，skip_name_resolve 数据库集群参数设置为 1（ON）且无法修改，因此您必须使用主机的 IP 地址而不是域名。有关更多信息，请参阅 MySQL 文档中的 skip_name_resolve。

```
mysql> CREATE USER 'repl_user'@'IP_address' IDENTIFIED BY 'password';
```

该用户需要 REPLICATION CLIENT 和 REPLICATION SLAVE 权限。向该用户授予这些权限。

如果您需要使用加密复制，则需为复制用户要求 SSL 连接。例如，您可以使用下面的语句之一来要求对用户账户 repl_user 使用 SSL 连接。

```
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'IP_address';
```

```
GRANT USAGE ON *.* TO 'repl_user'@'IP_address' REQUIRE SSL;
```

> **Note**
>
> 如果不包括 REQUIRE SSL，则复制连接可能会无提示地返回到未加密连接。

6. 在副本目标上开启复制

开启复制之前，建议您手动创建 Aurora MySQL 数据库集群或 RDS for MySQL 数据库实例副本目标的快照。如果出现问题，而您需要重新建立与数据库集群或数据库实例副本目标间的复制，则您可以从此快照还原数据库集群或数据库实例，而不必再次将数据导入您的副本目标之中。

使用以下说明对数据库引擎开启复制。

数据库引擎	说明
Aurora MySQL	从 Aurora MySQL 数据库集群中开启复制 1. 找到复制的起点。您需要二进制日志文件名和二进制日志位置。 如果您的数据库集群副本目标是根据以下内容创建的： • 数据库集群快照或克隆 - 从新数据库集群的最近事件中检索二进制日志文件名和位置，如 3. 创建复制源的副本或转储所示。

数据库引擎	说明
	• 数据库快照 – 您在创建复制源的快照时，从 SHOW SLAVE STATUS（Aurora MySQL 版本 2）或 SHOW REPLICA STATUS（Aurora MySQL 版本 3）命令中检索了二进制日志文件名和位置。 2. 连接到数据库集群并调用以下过程，使用上一步骤中的二进制日志文件名和位置启动与复制源之间的复制： • mysql.rds_set_external_source（Aurora MySQL 版本 3） • mysql.rds_set_external_master（Aurora MySQL 版本 2） • mysql.rds_start_replication（所有版本） 以下示例适用于 Aurora MySQL 版本 3。 `CALL mysql.rds_set_external_source ('mydbinstance.123456789012 .us-east-1.rds.amazonaws.com', 3306, 'repl_user', 'password', 'mysql-bin-changelog.000031', 107, 0); CALL mysql.rds_start_replication;` 要使用 SSL 加密，请将最终值设置为 1 而不是 0。

数据库引擎	说明

RDS for MySQL	从 Amazon RDS 数据库实例中开启复制
	1. 如果数据库实例副本目标是从数据库快照创建的，则您需要作为复制起点的二进制日志文件和二进制日志位置。您在创建复制源的快照时，从 SHOW SLAVE STATUS（Aurora MySQL 版本 2）或 SHOW REPLICA STATUS（Aurora MySQL 版本 3）命令中检索了这些值。
	2. 连接数据库实例，并调用 mysql.rds_set_external_master（Aurora MySQL 版本 2）或 mysql.rds_set_external_source（Aurora MySQL 版本 3） 和 mysql.rds_start_replication 过程，以开始与复制源之间的复制。使用上一步中的二进制日志文件名和位置。示例如下：
	`CALL mysql.rds_set_external_master ('mydbcluster.cluster-123456789012.us-east-1.rds.amazonaws.com', 3306, 'repl_user', 'password', 'mysql-bin-changelog.000031', 107, 0);`
	`CALL mysql.rds_start_replication;`
	要使用 SSL 加密，请将最终值设置为 1 而不是 0。

数据库引擎	说明

MySQL (外部)	开启从外部 MySQL 数据库进行复制
	1. 检索作为复制起点的二进制日志文件和二进制日志位置。您在创建复制源的快照时，从 SHOW SLAVE STATUS（Aurora MySQL 版本 2）或 SHOW REPLICA STATUS（Aurora MySQL 版本 3）命令中检索了这些值。如果外部 MySQL 副本目标是从使用 mysqldump 选项的 --master-data=2 命令的输出填充的，则二进制日志文件和二进制日志位置包含在该输出中。示例如下：
	`-- -- Position to start replication or point-in-time recovery from -- -- CHANGE MASTER TO MASTER_LOG_FILE='mysql-bin-changelog.000031', MASTER_LOG_POS=107;`
	2. 连接到外部 MySQL 副本目标，然后发出 CHANGE MASTER TO 和 START SLAVE（Aurora MySQL 版本 2）或 START REPLICA（Aurora MySQL 版本 3），以使用上一步中的二进制日志文件名和位置开始与复制源之间的复制，例如：
	`CHANGE MASTER TO MASTER_HOST = 'mydbcluster.cluster-123456789012.us-east-1.rds.amazonaws.com', MASTER_PORT = 3306, MASTER_USER = 'repl_user', MASTER_PASSWORD = 'password', MASTER_LOG_FILE = 'mysql-bin-changelog.000031', MASTER_LOG_POS = 107; -- And one of these statements depending on your engine version: START SLAVE; -- Aurora MySQL version 2 START REPLICA; -- Aurora MySQL version 3`

如果复制失败，可能会导致副本上的意外输入/输出大量增加，从而降低性能。如果复制失败或不再需要复制，则可运行 mysql.rds_reset_external_master（Aurora MySQL 版本 2） 或 mysql.rds_reset_external_source（Aurora MySQL 版本 3） 存储过程来删除复制配置。

设置停止复制到只读副本的位置

在 Aurora MySQL 版本 3.04 及更高版本中，您可以使用 mysql.rds_start_replication_until（Aurora MySQL 版本 3）存储过程启动复制，然后在指定的二进制日志文件位置停止复制。

开始复制到只读副本并在特定位置处停止复制

1. 通过使用 MySQL 客户端，以主用户的身份连接到副本 Aurora MySQL 数据库集群。
2. 运行 mysql.rds_start_replication_until（Aurora MySQL 版本 3）存储过程。

   以下示例将启动复制并复制更改，直到它到达 120 二进制日志文件中的 mysql-bin-changelog.000777 位置。在灾难恢复方案中，假定位置 120 刚好位于灾难之前。

```
call mysql.rds_start_replication_until(
  'mysql-bin-changelog.000777',
  120);
```

当到达停止点时，复制将自动停止。将生成以下 RDS 事件：Replication has been stopped since the replica reached the stop point specified by the rds_start_replication_until stored procedure。

如果使用基于 GTID 的复制，请使用 mysql.rds_start_replication_until_gtid（Aurora MySQL 版本 3）存储过程而不是 mysql.rds_start_replication_until（Aurora MySQL 版本 3）存储过程。有关基于 GTID 的复制的更多信息，请参阅使用基于 GTID 的复制。

7. 监控副本

在设置与 Aurora MySQL 数据库集群的 MySQL 复制时，您必须在 Aurora MySQL 数据库集群作为副本目标时监控它的失效转移事件。如果发生失效转移，则可能会在具有不同网络地址的新主机上重新创建作为副本目标的数据库集群。有关如何监控失效转移事件的信息，请参阅使用 Amazon RDS 事件通知。

您还可以通过连接到副本目标并运行 SHOW SLAVE STATUS（Aurora MySQL 版本 2）或 SHOW REPLICA STATUS（Aurora MySQL 版本 3）命令，监控副本目标落后于复制源的程度。命令输出中的 Seconds Behind Master 字段可以揭示副本目标落后于源的程度。

> **Important**
>
> 如果您升级数据库集群并指定自定义参数组，请确保在升级完成后手动重启集群。这样可以让集群使用新的自定义参数设置，并重新启动二进制日志复制。

在复制源和目标之间同步密码

使用 SQL 语句更改复制源上的用户账户和密码时，这些更改将自动复制到复制目标。

如果您使用 AWS Management Console、AWS CLI 或 RDS API 来更改复制源上的主密码，则这些更改不会自动复制到复制目标。如果要在源系统和目标系统之间同步主用户和主密码，则必须自己对复制目标进行相同的更改。

停止 Aurora MySQL 的二进制日志复制

要停止对 MySQL 数据库实例、外部 MySQL 数据库或其他 Aurora 数据库集群进行的二进制日志复制，请执行以下步骤 (本主题后文有详细讨论)。

1. 在副本目标上停止二进制日志复制

2. 在复制源上关闭二进制日志记录

1. 在副本目标上停止二进制日志复制

按照以下说明停止数据库引擎的二进制日志复制。

数据库引擎	说明
Aurora MySQL	在 Aurora MySQL 数据库集群副本目标上停止二进制日志复制 连接到作为副本目标的 Aurora 数据库集群，然后调用 mysql.rds_stop_replication 过程。
RDS for MySQL	在 Amazon RDS 数据库实例上停止二进制日志复制 连接到作为副本目标的 RDS 数据库实例，然后调用 mysql.rds_stop_replication 过程。
MySQL (外部)	在外部 MySQL 数据库上停止二进制日志复制

数据库引擎	说明
	连接到 MySQL 数据库并运行 STOP SLAVE（版本 5.7）或 STOP REPLICA（版本 8.0）命令。

2. 在复制源上关闭二进制日志记录

使用下表中的说明在数据库引擎的复制源上关闭二进制日志记录。

数据库引擎	说明
Aurora MySQL	在 Amazon Aurora 数据库集群上关闭二进制日志记录 1. 连接到作为复制源的 Aurora 数据库集群。 2. 使用 mysql.rds_set_configuration 过程并指定配置参数 binlog retention hours 以及值 NULL，如以下示例所示。 `CALL mysql.rds_set_configuration('binlog retention hours', NULL);` Note 不能将值 0 用于 binlog retention hours。 3. 在复制源上将 binlog_format 参数设置为 OFF。binlog_format 参数位于与数据库集群相关联的自定义数据库集群参数组中。 更改 binlog_format 参数值后，重启数据库集群以便更改生效。 有关更多信息，请参阅Amazon Aurora 数据库集群和数据库实例参数 和在 Amazon Aurora 中修改数据库参数组中的参数。
RDS for MySQL	在 Amazon RDS 数据库实例上关闭二进制日志记录 您不能直接为 Amazon RDS 数据库实例关闭二进制日志记录，但可以通过执行以下操作来关闭它：

数据库引擎	说明
	1. 为数据库实例关闭自动备份。您可通过修改现有数据库实例并将 Backup Retention Period（备份保留期）设置为 0 来关闭自动备份。有关更多信息，请参阅 Amazon Relational Database Service 用户指南 中的修改 Amazon RDS 数据库实例和使用备份。 2. 删除数据库实例的所有只读副本。有关更多信息，请参阅 Amazon Relational Database Service 用户指南 中的使用 MariaDB、MySQL 和 PostgreSQL 数据库实例的只读副本。

MySQL (外部)	在外部 MySQL 数据库上关闭二进制日志记录
	连接到 MySQL 数据库并调用 `STOP REPLICATION` 命令。
	1. 从命令 shell 中停止 `mysqld` 服务，
	`sudo service mysqld stop`
	2. 编辑 `my.cnf` 文件（此文件通常位于 `/etc` 下）。
	`sudo vi /etc/my.cnf`
	从 `log_bin` 部分中删除 `server_id` 和 `[mysqld]` 选项。
	有关更多信息，请参阅 MySQL 文档中的 Setting the replication source configuration。
	3. 启动 mysql 服务。
	`sudo service mysqld start`

使用 Amazon Aurora 扩展 MySQL 数据库的读取

您可以将 Amazon Aurora 用于 MySQL 数据库实例，以便利用 Amazon Aurora 的读取扩展功能并为 MySQL 数据库实例扩展读取工作负载。要使用 Aurora 对 MySQL 数据库实例扩缩读取，请创建 Amazon Aurora MySQL 数据库集群并使它成为 MySQL 数据库实例的只读副本。这适用于 RDS for MySQL 数据库实例或是在 Amazon RDS 外部运行的 MySQL 数据库。

有关创建 Amazon Aurora 数据库集群的信息，请参阅创建 Amazon Aurora 数据库集群。

在 MySQL 数据库实例与 Amazon Aurora 数据库集群之间设置复制时，请确保遵循以下准则：

- 当您引用 Amazon Aurora 数据库集群时，使用 Amazon Aurora MySQL 数据库集群端点地址。如果发生故障转移，则提升为 Aurora 数据库集群主实例的 Aurora MySQL 副本继续使用数据库集群端点地址。
- 在您的写入器实例上维护二进制日志，直至您确认其已应用于 Aurora 副本。这种维护将确保您可以在发生故障时还原写入器实例。

Important

当使用自管理复制时，您负责监控和解决可能发生的所有复制问题。有关更多信息，请参阅 诊断并解决只读副本之间的滞后。

Note

对 Aurora MySQL 数据库集群启动复制功能所需的权限受到限制且对 Amazon RDS 主用户不可用。为此，您必须使用 mysql.rds_set_external_master（Aurora MySQL 版本 2）或 mysql.rds_set_external_source（Aurora MySQL 版本 3）和 mysql.rds_start_replication 过程来设置 Aurora MySQL 数据库集群和 MySQL 数据库实例之间的复制。

启动外部源实例和 Aurora MySQL 数据库集群之间的复制

1. 将源 MySQL 数据库实例设为只读：

```
mysql> FLUSH TABLES WITH READ LOCK;
mysql> SET GLOBAL read_only = ON;
```

2. 对源 MySQL 数据库实例运行 SHOW MASTER STATUS 命令以确定二进制日志位置。将会收到类似于以下示例的输出：

```
File                          Position
-----------------------------------
 mysql-bin-changelog.000031      107
-----------------------------------
```

3. 使用 mysqldump 将数据库从外部 MySQL 数据库实例复制到 Amazon Aurora MySQL 数据库集群。对于非常大的数据库，您可能希望使用 Amazon Relational Database Service 用户指南 中的将数据导入到 MySQL 或 MariaDB 数据库实例并减少停机时间中的过程。

   对于 Linux、macOS 或 Unix：

```
mysqldump \
    --databases <database_name> \
    --single-transaction \
    --compress \
    --order-by-primary \
    -u local_user \
    -p local_password | mysql \
        --host aurora_cluster_endpoint_address \
        --port 3306 \
        -u RDS_user_name \
        -p RDS_password
```

   对于 Windows：

```
mysqldump ^
    --databases <database_name> ^
    --single-transaction ^
    --compress ^
    --order-by-primary ^
    -u local_user ^
    -p local_password | mysql ^
        --host aurora_cluster_endpoint_address ^
        --port 3306 ^
        -u RDS_user_name ^
        -p RDS_password
```

   > **Note**
   >
   > 确保 -p 选项和输入的密码之间没有空格。

   在 --host 命令中使用 --user (-u)、--port、-p 和 mysql 选项，以指定用于连接到 Aurora 数据库集群的主机名、用户名、端口和密码。主机名是 Amazon Aurora 数据库集群端点中的 DNS

名称，例如 mydbcluster.cluster-123456789012.us-east-1.rds.amazonaws.com。您可以在 Amazon RDS 管理控制台上的集群详细信息中找到端点值。

4. 再次将源 MySQL 数据库实例设为可写：

```
mysql> SET GLOBAL read_only = OFF;
mysql> UNLOCK TABLES;
```

有关生成备份以用于复制的更多信息，请参阅 MySQL 文档中的Backing up a source or replica by making it read only。

5. 在 Amazon RDS 管理控制台中，将托管源 MySQL 数据库的服务器的 IP 地址添加到 Amazon Aurora 数据库集群的 VPC 安全组。有关修改 VPC 安全组的更多信息，请参阅 Amazon Virtual Private Cloud 用户指南 中的您的 VPC 的安全组。

   您可能还需要配置本地网络以允许来自 Amazon Aurora 数据库集群的 IP 地址的连接，以便它能与源 MySQL 实例进行通信。要查找 Amazon Aurora 数据库集群的 IP 地址，请使用 host 命令。

```
host aurora_endpoint_address
```

   主机名是 Amazon Aurora 数据库集群端点中的 DNS 名称。

6. 通过使用所选的客户端，连接到外部 MySQL 实例并创建用于复制的 MySQL 用户。此账户仅用于复制，并且必须仅供您的域使用以增强安全性。以下是示例。

```
CREATE USER 'repl_user'@'example.com' IDENTIFIED BY 'password';
```

7. 对于外部 MySQL 实例，向复制用户授予 REPLICATION CLIENT 和 REPLICATION SLAVE 权限。例如，要为您的域的“REPLICATION CLIENT”用户授予对所有数据库的 REPLICATION SLAVE 和 repl_user 权限，请发出以下命令。

```
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'example.com'
    IDENTIFIED BY 'password';
```

8. 在设置复制之前，请创建 Aurora MySQL 数据库集群的手动快照以作为只读副本。如果您需要将数据库集群作为只读副本来重新建立复制，则可从此快照还原 Aurora MySQL 数据库集群，而不必将 MySQL 数据库实例中的数据导入新的 Aurora MySQL 数据库集群。

9. 使 Amazon Aurora 数据库集群成为副本。以主用户身份连接到 Amazon Aurora 数据库集群，并通过使用 mysql.rds_set_external_master（Aurora MySQL 版本 2）或

mysql.rds_set_external_source（Aurora MySQL 版本 3）和 mysql.rds_start_replication 过程将源 MySQL 数据库确定为复制源。

使用在步骤 2 中确定的二进制日志文件名和位置。示例如下：

```
For Aurora MySQL version 2:
CALL mysql.rds_set_external_master ('mymasterserver.example.com', 3306,
    'repl_user', 'password', 'mysql-bin-changelog.000031', 107, 0);

For Aurora MySQL version 3:
CALL mysql.rds_set_external_source ('mymasterserver.example.com', 3306,
    'repl_user', 'password', 'mysql-bin-changelog.000031', 107, 0);
```

10. 在 Amazon Aurora 数据库集群上，调用 mysql.rds_start_replication 过程以开始复制。

```
CALL mysql.rds_start_replication;
```

在源 MySQL 数据库实例与 Amazon Aurora 数据库集群之间建立复制之后，可以将 Aurora 副本添加到 Amazon Aurora 数据库集群。随后可以连接到 Aurora 副本以对数据进行读取扩展。有关创建 Aurora 副本的信息，请参阅将 Aurora 副本添加到数据库集群。

优化 Aurora MySQL 的二进制日志复制

接下来，您可以了解如何优化二进制日志复制性能和排查 Aurora MySQL 中的相关问题。

> **Tip**
>
> 本讨论假定您熟悉 MySQL 二进制日志复制机制及其工作原理。有关背景信息，请参阅 MySQL 文档中的复制实施。

多线程二进制日志复制

使用多线程二进制日志复制时，SQL 线程会从中继日志中读取事件并将其排队，以便 SQL 工作线程应用。SQL 工作线程由协调器线程管理。尽可能并行应用二进制日志事件。

Aurora MySQL 版本 3 和 Aurora MySQL 2.12.1 及更高版本支持多线程二进制日志复制。

当 Aurora MySQL 数据库实例配置为使用二进制日志复制时，默认情况下，副本实例对低于 3.04 的 Aurora MySQL 版本使用单线程复制。要启用多线程复制，请在您的自定义参数组中将 replica_parallel_workers 参数更新为大于零的值。

对于 Aurora MySQL 版本 3.04 及更高版本，复制默认是多线程的，replica_parallel_workers 设置为 4。您可以在自定义参数组中修改此参数。

以下配置选项可以帮助您优化多线程复制。有关使用信息，请参阅 MySQL 参考手册中的复制和二进制日志记录选项和变量。

最佳配置取决于多个因素。例如，二进制日志复制的性能受数据库工作负载特征和副本运行所在的数据库实例类的影响。因此，我们建议您在将新参数设置应用于生产实例之前，彻底测试对这些配置参数的所有更改：

- binlog_group_commit_sync_delay
- binlog_group_commit_sync_no_delay_count
- binlog_transaction_dependency_history_size
- binlog_transaction_dependency_tracking
- replica_preserve_commit_order
- replica_parallel_type
- replica_parallel_workers

在 Aurora MySQL 版本 3.06 及更高版本中，在为具有多个二级索引的大型表复制事务时，可以提高二进制日志副本的性能。此特征引入了一个线程池，用于在二进制日志副本上并行应用二级索引更改。该特征由 aurora_binlog_replication_sec_index_parallel_workers 数据库集群参数控制，该参数控制可用于应用二级索引更改的并行线程总数。默认情况下，此参数设置为 0（禁用）。启用此特征不需要重启实例。要启用此特征，请停止正在进行的复制，设置所需的并行工作线程数，然后重新开始复制。

您也可以将此参数用作全局变量，其中 *n* 是并行工作线程的数量：

```
SET global aurora_binlog_replication_sec_index_parallel_workers=n;
```

优化二进制日志复制（Aurora MySQL 2.10 及更高版本）

在 Aurora MySQL 2.10 及更高版本中，Aurora 会自动将称为二进制日志 I/O 缓存的优化应用于二进制日志复制。通过缓存最近提交的二进制日志事件，此优化旨在提高二进制日志转储线程性能，同时限制对二进制日志源实例上前台事务的影响。

> **Note**
>
> 用于此特征的内存独立于 MySQL binlog_cache 设置。
> 此特征不适用于使用 db.t2 和 db.t3 实例类的 Aurora 数据库实例。

您无需调整任何配置参数即可启用此优化。特别是，如果在早期 Aurora MySQL 版本中将配置参数 aurora_binlog_replication_max_yield_seconds 调整为非零值，请在 Aurora MySQL 2.10 及更高版本中将其重新设置为零。

Aurora MySQL 2.10 及更高版本中提供了状态变量 aurora_binlog_io_cache_reads 和 aurora_binlog_io_cache_read_requests。这些状态变量可帮助您监控从二进制日志 I/O 缓存读取数据的频率。

- aurora_binlog_io_cache_read_requests 显示来自缓存的二进制日志输入/输出读取请求的数量。
- aurora_binlog_io_cache_reads 显示从缓存检索信息的二进制日志输入/输出读取的数量。

以下 SQL 查询计算利用缓存信息的二进制日志读取请求的百分比。在此情况下，比率越接近 100，就越好。

```
mysql> SELECT
  (SELECT VARIABLE_VALUE FROM INFORMATION_SCHEMA.GLOBAL_STATUS
    WHERE VARIABLE_NAME='aurora_binlog_io_cache_reads')
  / (SELECT VARIABLE_VALUE FROM INFORMATION_SCHEMA.GLOBAL_STATUS
    WHERE VARIABLE_NAME='aurora_binlog_io_cache_read_requests')
  * 100
  as binlog_io_cache_hit_ratio;
+---------------------------+
| binlog_io_cache_hit_ratio |
+---------------------------+
|         99.99847949080622 |
+---------------------------+
```

二进制日志 I/O 缓存特征还包括与二进制日志转储线程相关的新指标。转储线程是当新的二进制日志副本连接到二进制日志源实例时创建的线程。

转储线程指标每 60 秒输出到数据库日志中一次，并带有前缀 [Dump thread metrics]。这些指标包括每个二进制日志副本的信息，例如，Secondary_id、Secondary_uuid、二进制日志文件名以

及每个副本读取的位置。这些指标还包括 Bytes_behind_primary，表示复制源和副本之间的距离（以字节为单位）。此指标衡量副本 I/O 线程的滞后。该数字不同于副本 SQL 应用程序线程的滞后，后者通过二进制日志副本上的 seconds_behind_master 指标表示。您可以通过检查距离是减少还是增加来确定二进制日志副本是跟上源还是落后。

优化二进制日志复制（Aurora MySQL 版本 2 至 2.09）

要优化 Aurora MySQL 的二进制日志复制，您可以调整以下集群级优化参数。这些参数可帮助您在二进制日志源实例的延迟和复制滞后之间指定适当的平衡。

- aurora_binlog_use_large_read_buffer
- aurora_binlog_read_buffer_size
- aurora_binlog_replication_max_yield_seconds

> **Note**
>
> 对于 MySQL 5.7 兼容的集群，您可以在 Aurora MySQL 版本 2 至 2.09.* 中使用这些参数。在 Aurora MySQL 2.10.0 及更高版本中，这些参数被二进制日志 I/O 缓存优化所取代，无需使用这些参数。

主题

- 大型读取缓冲区和最大生成率优化概述
- 相关参数
- 启用二进制日志复制的最大生成率机制
- 关闭二进制日志复制最大生成率优化
- 关闭大型读取缓冲区

大型读取缓冲区和最大生成率优化概述

当集群处理大量事务时，如果二进制日志转储线程访问 Aurora 集群卷，您可能会遇到二进制日志复制性能降低的情况。您可以使用参数 aurora_binlog_use_large_read_buffer、aurora_binlog_replication_max_yield_seconds 和 aurora_binlog_read_buffer_size 帮助最大限度减少这种类型的争用。

假设您有一个情况：aurora_binlog_replication_max_yield_seconds 设置为大于 0 并且转储线程的当前二进制日志文件处于活动状态。在这种情况下，二进制日志转储线程最多会等待指定

的秒数，以便事务填充当前二进制日志文件。此等待期避免了单独复制每个二进制日志事件可能引起的争用。但是，这样做会增加二进制日志副本的副本滞后。这些副本可能落后于源，落后的描述与 aurora_binlog_replication_max_yield_seconds 设置相同。

当前的二进制日志文件表示转储线程当前正在读取以执行复制的二进制日志文件。我们认为一个二进制日志文件在更新或打开以供传入事务更新时，该二进制日志文件处于活动状态。Aurora MySQL 填满活动的二进制日志文件后，MySQL 会创建并切换到新的二进制日志文件。旧二进制日志文件变为非活动状态。传入的事务不再更新它。

> **Note**
>
> 在调整这些参数之前，请衡量一段时间内的事务延迟和吞吐量。即使偶尔出现争用，您也可能会发现二进制日志复制性能稳定且延迟低。

aurora_binlog_use_large_read_buffer

如果此参数设置为 1，Aurora MySQL 会基于参数 aurora_binlog_read_buffer_size 和 aurora_binlog_replication_max_yield_seconds 的设置优化二进制日志复制。如果 aurora_binlog_use_large_read_buffer 为 0，Aurora MySQL 会忽略 aurora_binlog_read_buffer_size 和 aurora_binlog_replication_max_yield_seconds 参数的值。

aurora_binlog_read_buffer_size

具有较大读取缓冲区的二进制日志转储线程可通过读取每个输入/输出的更多事件来最大限度地减少读取输入/输出操作的数量。参数 aurora_binlog_read_buffer_size 设置读取缓冲区大小。大型读取缓冲区可以减少生成大量二进制日志数据的工作负载的二进制日志争用。

> **Note**
>
> 此参数仅在集群也具有 aurora_binlog_use_large_read_buffer=1 设置时才有效。提高读取缓冲区的大小不会影响二进制日志复制的性能。二进制日志转储线程不会等待更新事务填满读缓冲区。

`aurora_binlog_replication_max_yield_seconds`

如果您的工作负载需要较低的事务延迟，并且可以容忍某些复制延迟，您可以提高 `aurora_binlog_replication_max_yield_seconds` 参数。此参数控制集群中的二进制日志复制的最大生成属性。

> **Note**
>
> 此参数仅在集群也具有 `aurora_binlog_use_large_read_buffer=1` 设置时才有效。

Aurora MySQL 可立即识别对 `aurora_binlog_replication_max_yield_seconds` 参数值的任何更改。您无需重新启动数据库实例。但是，开启此设置后，只有当当前二进制日志文件达到 128MB 的最大大小并轮换到新文件时，转储线程才开始生成。

相关参数

使用以下数据库集群参数开启二进制日志优化。

参数	默认值	有效值	描述
`aurora_binlog_use_large_read_buffer`	1	0, 1	切换开启复制改进特征。当其值为 1 时，二进制日志转储线程将使用 `aurora_binlog_read_buffer_size` 进行二进制日志复制；否则使用默认缓冲区大小（8K）。在 Aurora MySQL 版本 3 中不使用。
`aurora_binlog_read_buffer_size`	5242880	8192-536870912	参数 `aurora_binlog_use_large_read_buffer` 设置为 1 时，二进制日志转储线程使用的读取缓

参数	默认值	有效值	描述
			冲区大小。在 Aurora MySQL 版本 3 中不使用。
aurora_binlog_replication_max_yield_seconds	0	0-36000	对于版本 Aurora MySQL 2.07.*，可接受的最大值为 45。您可以在 2.09 及更高版本中将其调整为更高的值。 对于版本 2，仅当参数 aurora_binlog_use_large_read_buffer 设置为 1 时，此参数才有效。

启用二进制日志复制的最大生成率机制

您可以按如下方式开启二进制日志复制最大生成率优化。这样做可以最大限度地减少二进制日志源实例上的事务延迟。但是，您可能会遇到更高的复制滞后。

要为 Aurora MySQL 集群开启最大生成率二进制日志优化

1. 使用以下参数设置创建或编辑数据库集群参数组：
   - aurora_binlog_use_large_read_buffer：使用值 ON 或 1 开启。
   - aurora_binlog_replication_max_yield_seconds：指定一个大于 0 的值。
2. 将数据库集群参数组与作为二进制日志源的 Aurora MySQL 集群相关联。为此，请按照 Amazon Aurora 的参数组 中的过程进行操作。
3. 确认参数更改生效。为此，请在二进制日志源实例上运行以下查询。

```
SELECT @@aurora_binlog_use_large_read_buffer,
 @@aurora_binlog_replication_max_yield_seconds;
```

您的输出应类似于以下内容。

```
+-----------------------------------------
+-------------------------------------------------+
| @@aurora_binlog_use_large_read_buffer |
 @@aurora_binlog_replication_max_yield_seconds |
+-----------------------------------------
+-------------------------------------------------+
|                                       1 |
  45 |
+-----------------------------------------
+-------------------------------------------------+
```

关闭二进制日志复制最大生成率优化

您可以按如下方式关闭二进制日志复制最大生成率优化。这样做可最大限度地减少复制滞后。但是，在二进制日志源实例上，您可能会遇到更高的延迟。

要关闭 Aurora MySQL 集群的最大生成率优化

1. 确保与 Aurora MySQL 集群关联的数据库集群参数组将 `aurora_binlog_replication_max_yield_seconds` 设置为 0。有关使用参数组设置配置参数的更多信息，请参阅 Amazon Aurora 的参数组。
2. 确认参数更改生效。为此，请在二进制日志源实例上运行以下查询。

```
SELECT @@aurora_binlog_replication_max_yield_seconds;
```

您的输出应类似于以下内容。

```
+-------------------------------------------------+
| @@aurora_binlog_replication_max_yield_seconds |
+-------------------------------------------------+
|                                             0 |
+-------------------------------------------------+
```

关闭大型读取缓冲区

您可以按如下方式关闭整个大型读取缓冲区特征。

要关闭 Aurora MySQL 集群的大型二进制日志读缓冲区

1. 将 `aurora_binlog_use_large_read_buffer` 重置为 `OFF` 或 `0`。

   确保与 Aurora MySQL 集群关联的数据库集群参数组将 `aurora_binlog_use_large_read_buffer` 设置为 `0`。有关使用参数组设置配置参数的更多信息，请参阅 Amazon Aurora 的参数组。

2. 在二进制日志源实例上，运行以下查询。

   ```
   SELECT @@ aurora_binlog_use_large_read_buffer;
   ```

   您的输出应类似于以下内容。

   ```
   +----------------------------------------+
   | @@aurora_binlog_use_large_read_buffer |
   +----------------------------------------+
   |                                      0 |
   +----------------------------------------+
   ```

为 Aurora MySQL 设置增强型二进制日志

增强型二进制日志可减少开启二进制日志所导致的计算性能开销，在某些情况下，该开销最高可达 50%。使用增强型二进制日志，这种开销可以减少到大约 13%。为了减少开销，增强型二进制日志将二进制和事务日志并行写入存储，从而最大限度地减少在事务提交时间写入的数据。

与社区 MySQL 二进制日志相比，使用增强型二进制日志还可以将重启和故障转移后的数据库恢复时间缩短多达 99%。增强型二进制日志与现有的基于二进制日志的工作负载兼容，您与其交互的方式和您与社区 MySQL 二进制日志交互的方式相同。

增强型二进制日志在 Aurora MySQL 版本 3.03.1 及更高版本上可用。

主题

- 配置增强型二进制日志参数
- 其他相关参数
- 增强型二进制日志和社区 MySQL 二进制日志之间的区别
- 增强型二进制日志的 Amazon CloudWatch 指标
- 增强型二进制日志限制

配置增强型二进制日志参数

您可以通过开启/关闭增强型二进制日志参数，在社区 MySQL 二进制日志和增强型二进制日志之间切换。现有的二进制日志使用者可以继续读取和使用二进制日志文件，而不会在二进制日志文件序列中出现任何间隔。

要开启增强型二进制日志，请设置以下参数：

参数	默认值	描述
binlog_format	–	将 binlog_format 参数设置为您选择的二进制日志记录格式，以开启增强型二进制日志。确保 binlog_format parameter 未设置为 OFF。有关更多信息，请参阅配置 Aurora MySQL 二进制日志记录。
aurora_enhanced_binlog	0	在与 Aurora MySQL 集群关联的数据库集群参数组中，将此参数的值设置为 1。更改此参数的值时，当 DBClusterParameterGroupStatus 值显示为 pending-reboot 时，必须重启写入器实例。
binlog_backup	1	关闭此参数可开启增强型二进制日志。为此，请将此参数的值设置为 0。
binlog_replication_globaldb	1	关闭此参数可开启增强型二进制日志。为此，请将此参数的值设置为 0。

> **Important**
>
> 只有在使用增强型二进制日志时，才能关闭 binlog_backup 和 binlog_replication_globaldb 参数。

要关闭增强型二进制日志，请设置以下参数：

参数	描述
aurora_enhanced_binlog	在与 Aurora MySQL 集群关联的数据库集群参数组中，将此参数的值设置为 0。每当您更改此参数的值时，当 DBClusterParameterGroupStatus 值显示为 pending-reboot 时，必须重启写入器实例。
binlog_backup	关闭增强型二进制日志时开启此参数。为此，请将此参数的值设置为 1。
binlog_replication_globaldb	关闭增强型二进制日志时开启此参数。为此，请将此参数的值设置为 1。

要检查增强型二进制日志是否已开启，请在 MySQL 客户端中使用以下命令：

```
mysql>show status like 'aurora_enhanced_binlog';

+--------------------------+--------+
| Variable_name            | Value  |
+--------------------------+--------+
| aurora_enhanced_binlog | ACTIVE |
+--------------------------+--------+
1 row in set (0.00 sec)
```

开启增强型二进制日志时，输出将对于 aurora_enhanced_binlog 显示 ACTIVE。

其他相关参数

当您开启增强型二进制日志时，以下参数会受到影响：

- max_binlog_size 参数可见但不可修改。当开启增强型二进制日志时，此参数的默认值 134217728 会自动调整为 268435456。
- 与社区 MySQL 二进制日志不同，当开启增强型二进制日志时，binlog_checksum 不充当动态参数。要使对此参数的更改生效，必须手动重启数据库集群，即使 ApplyMethod 为 immediate 也是如此。
- 开启增强型二进制日志时，您对 binlog_order_commits 参数设置的值对提交顺序没有影响。提交始终是按顺序排列的，不会对性能产生任何进一步的影响。

增强型二进制日志和社区 MySQL 二进制日志之间的区别

与社区 MySQL 二进制日志相比，增强型二进制日志与克隆、备份和 Aurora Global Database 的交互方式不同。我们建议您在使用增强型二进制日志之前了解以下差异。

- 来自源数据库集群的增强型二进制日志文件在克隆的数据库集群上不可用。
- 增强型二进制日志文件不包含在 Aurora 备份中。因此，尽管在数据库集群上设置了任何保留期，但在还原数据库集群后，源数据库集群中的增强型二进制日志文件不可用。
- 与 Aurora 全局数据库一起使用时，主数据库集群的增强二进制日志文件不会复制到辅助区域中的数据库集群。

示例

以下示例说明了增强型二进制日志和社区 MySQL 二进制日志之间的区别。

在还原或克隆的数据库集群上

开启增强型二进制日志后，历史二进制日志文件在还原或克隆的数据库集群中不可用。在执行还原或克隆操作后，如果开启二进制日志，则新的数据库集群将开始写入其自己的二进制日志文件序列，从 1 开始（mysql-bin-changelog.000001）。

要在还原或克隆操作后开启增强型二进制日志，请在还原或克隆的数据库集群上设置所需的数据库集群参数。有关更多信息，请参阅 配置增强型二进制日志参数。

Example 示例：开启增强型二进制日志时执行的克隆或还原操作

源数据库集群：

```
mysql> show binary logs;

+----------------------------+-----------+-----------+
```

```
| Log_name                   | File_size | Encrypted |
+----------------------------+-----------+-----------+
| mysql-bin-changelog.000001 |       156 | No        |
| mysql-bin-changelog.000002 |       156 | No        |
| mysql-bin-changelog.000003 |       156 | No        |
| mysql-bin-changelog.000004 |       156 | No        | --> Enhanced Binlog turned on
| mysql-bin-changelog.000005 |       156 | No        | --> Enhanced Binlog turned on
| mysql-bin-changelog.000006 |       156 | No        | --> Enhanced Binlog turned on
+----------------------------+-----------+-----------+
6 rows in set (0.00 sec)
```

在还原或克隆的数据库集群上，开启增强型二进制日志时不备份二进制日志文件。为避免二进制日志数据出现不连续性，在开启增强型二进制日志之前写入的二进制日志文件也不可用。

```
mysql>show binary logs;

+----------------------------+-----------+-----------+
| Log_name                   | File_size | Encrypted |
+----------------------------+-----------+-----------+
| mysql-bin-changelog.000001 |       156 | No        | --> New sequence of Binlog files
+----------------------------+-----------+-----------+
1 row in set (0.00 sec)
```

Example 示例：关闭增强型二进制日志时执行的克隆或还原操作

源数据库集群：

```
mysql>show binary logs;

+----------------------------+-----------+-----------+
| Log_name                   | File_size | Encrypted |
+----------------------------+-----------+-----------+
| mysql-bin-changelog.000001 |       156 | No        |
| mysql-bin-changelog.000002 |       156 | No        | --> Enhanced Binlog enabled
| mysql-bin-changelog.000003 |       156 | No        | --> Enhanced Binlog enabled
| mysql-bin-changelog.000004 |       156 | No        |
| mysql-bin-changelog.000005 |       156 | No        |
| mysql-bin-changelog.000006 |       156 | No        |
+----------------------------+-----------+-----------+
```

```
6 rows in set (0.00 sec)
```

mysql-bin-changelog.000003 之后禁用了增强型二进制日志。在还原或克隆的数据库集群上，关闭增强型二进制日志后写入的二进制日志文件可用。

```
mysql>show binary logs;

+----------------------------+-----------+-----------+
| Log_name                   | File_size | Encrypted |
+----------------------------+-----------+-----------+
| mysql-bin-changelog.000004 |       156 | No        |
| mysql-bin-changelog.000005 |       156 | No        |
| mysql-bin-changelog.000006 |       156 | No        |
+----------------------------+-----------+-----------+
1 row in set (0.00 sec)
```

在 Amazon Aurora 全局数据库上

在 Amazon Aurora 全局数据库上，主数据库集群的二进制日志数据不会复制到辅助数据库集群。在执行跨区域故障转移过程后，二进制日志数据在新升级的主数据库集群中不可用。如果开启二进制日志，则新提升的数据库集群将开始其自己的二进制日志文件序列，从 1 开始（mysql-bin-changelog.000001）。

要在故障转移后开启增强型二进制日志，您必须在辅助数据库集群上设置所需的数据库集群参数。有关更多信息，请参阅 配置增强型二进制日志参数。

Example 示例：开启增强型二进制日志时执行全球数据库失效转移操作

旧的主数据库集群（故障转移前）：

```
mysql>show binary logs;

+----------------------------+-----------+-----------+
| Log_name                   | File_size | Encrypted |
+----------------------------+-----------+-----------+
| mysql-bin-changelog.000001 |       156 | No        |
| mysql-bin-changelog.000002 |       156 | No        |
| mysql-bin-changelog.000003 |       156 | No        |
| mysql-bin-changelog.000004 |       156 | No        | --> Enhanced Binlog enabled
```

```
| mysql-bin-changelog.000005 |       156 | No        | --> Enhanced Binlog enabled
| mysql-bin-changelog.000006 |       156 | No        | --> Enhanced Binlog enabled
+----------------------------+-----------+-----------+
6 rows in set (0.00 sec)
```

新的主数据库集群（故障转移后）：

开启增强型二进制日志后，二进制日志文件不会复制到辅助区域。为避免二进制日志数据出现不连续性，在开启增强型二进制日志之前写入的二进制日志文件不可用。

```
mysql>show binary logs;

+----------------------------+-----------+-----------+
| Log_name                   | File_size | Encrypted |
+----------------------------+-----------+-----------+
| mysql-bin-changelog.000001 |       156 | No        | --> Fresh sequence of Binlog
 files
+----------------------------+-----------+-----------+
1 row in set (0.00 sec)
```

Example 示例：关闭增强型二进制日志时执行全球数据库失效转移操作

源数据库集群：

```
mysql>show binary logs;

+----------------------------+-----------+-----------+
| Log_name                   | File_size | Encrypted |
+----------------------------+-----------+-----------+
| mysql-bin-changelog.000001 |       156 | No        |
| mysql-bin-changelog.000002 |       156 | No        | --> Enhanced Binlog enabled
| mysql-bin-changelog.000003 |       156 | No        | --> Enhanced Binlog enabled
| mysql-bin-changelog.000004 |       156 | No        |
| mysql-bin-changelog.000005 |       156 | No        |
| mysql-bin-changelog.000006 |       156 | No        |
+----------------------------+-----------+-----------+
6 rows in set (0.00 sec)
```

还原或克隆的数据库集群：

`mysql-bin-changelog.000003` 之后禁用了增强型二进制日志。关闭增强型二进制日志后写入的二进制日志文件会被复制，并在新提升的数据库集群中可用。

```
mysql>show binary logs;

+----------------------------+-----------+-----------+
| Log_name                   | File_size | Encrypted |
+----------------------------+-----------+-----------+
| mysql-bin-changelog.000004 |       156 | No        |
| mysql-bin-changelog.000005 |       156 | No        |
| mysql-bin-changelog.000006 |       156 | No        |
+----------------------------+-----------+-----------+
3 rows in set (0.00 sec)
```

增强型二进制日志的 Amazon CloudWatch 指标

以下 Amazon CloudWatch 指标仅在开启增强型二进制日志时发布。

CloudWatch 指标	描述	单位
ChangeLogBytesUsed	增强型二进制日志使用的存储空间量。	字节
ChangeLogReadIOPs	在 5 分钟的时间间隔内，在增强型二进制日志中执行的读取 I/O 操作的数量。	每 5 分钟计数
ChangeLogWriteIOPs	在 5 分钟的时间间隔内，在增强型二进制日志中执行的写入磁盘 I/O 操作的数量。	每 5 分钟计数

增强型二进制日志限制

开启增强型二进制日志时，以下限制适用于 Amazon Aurora 数据库集群。

- 仅在 Aurora MySQL 版本 3.03.1 及更高版本上支持增强型二进制日志。
- 在主数据库集群上写入的增强型二进制日志文件不会复制到克隆或还原的数据库集群。

- 与 Amazon Aurora 全局数据库一起使用时，主数据库集群的增强型二进制日志文件不会复制到辅助数据库集群。因此，在故障转移过程之后，历史二进制日志数据在新的主数据库集群中不可用。
- 将忽略以下二进制日志配置参数：
  - `binlog_group_commit_sync_delay`
  - `binlog_group_commit_sync_no_delay_count`
  - `binlog_max_flush_queue_time`
- 您无法删除或重命名数据库中损坏的表。要删除这些表，您可以联系 支持。
- 开启增强型二进制日志时，会禁用二进制日志 I/O 缓存。有关更多信息，请参阅 优化 Aurora MySQL 的二进制日志复制。

> **Note**
>
> 增强型二进制日志提供了与二进制日志 I/O 缓存类似的读取性能改进以及更好的写入性能改进。

- 不支持回溯特征。在以下情况下，无法在数据库集群中开启增强型二进制日志：
  - 当前已启用回溯特征的数据库集群。
  - 先前已启用但现在禁用回溯功能的数据库集群。
  - 从启用了回溯特征的源数据库集群或快照中还原的数据库集群。

使用基于 GTID 的复制

以下内容说明了如何在 Aurora MySQL 集群和外部源之间使用采用二进制日志（binlog）复制的全局事务标识符（GTID）。

> **Note**
>
> 对于 Aurora，只能通过对外部 MySQL 数据库使用二进制日志复制的 Aurora MySQL 集群使用此功能。另一个数据库可能是其他 AWS 区域的 Amazon RDS MySQL 实例、本地 MySQL 数据库或 Aurora 数据库集群。要了解如何配置此类复制，请参阅 Aurora 与 MySQL 之间或 Aurora 与其他 Aurora 数据库集群之间的复制（二进制日志复制）。

如果您使用的是二进制日志复制，不熟悉 MySQL 的基于 GTID 的复制，请参阅 MySQL 文档中的 Replication with global transaction identifiers。

Aurora MySQL 版本 2 和 3 支持基于 GTID 的复制。

主题

- 全局事务标识符 (GTID) 概述
- 基于 GTID 的复制的参数
- 为 Aurora MySQL 集群启用基于 GTID 的复制
- 为Aurora MySQL 数据库集群禁用基于 GTID 的复制

全局事务标识符 (GTID) 概述

全局事务标识符 (GTID) 是为提交的 MySQL 事务生成的唯一标识符。您可以使用 GTID 让二进制日志复制的故障排除更加简单便捷。

> **Note**
>
> 当 Aurora 在集群中的数据库实例之间同步数据时，该复制机制不会涉及二进制日志 (binlog)。对于 Aurora MySQL，基于 GTID 的复制仅在您还使用二进制日志复制从外部的 MySQL 兼容数据向 Aurora MySQL 数据库集群复制或从中复制时应用。

MySQL 使用两种不同类型的事务进行二进制日志复制：

- GTID 事务 – 由 GTID 标识的事务。
- 匿名事务 – 未分配 GTID 的事务。

在复制配置中，GTID 在所有数据库实例中是唯一的。GTID 简化了复制配置，因为在使用它们时，您不必引用日志文件位置。通过使用 GTID，还可以更轻松地跟踪复制的事务并确定源实例和副本是否一致。

在从外部的 MySQL 兼容数据向 Aurora 集群复制时，通常使用 Aurora 的基于 GTID 的复制。您可以将此复制配置设置为从本地数据库或 Amazon RDS 数据库迁移到 Aurora MySQL 的一部分。如果外部数据库已使用 GTID，为 Aurora 集群启用基于 GTID 的复制可以简化复制过程。

为 Aurora MySQL 集群配置基于 GTID 的复制的方法是先在数据库集群参数组中设置相关配置参数。然后将该参数组与集群关联。

基于 GTID 的复制的参数

可以使用以下参数配置基于 GTID 的复制。

参数	有效值	描述
gtid_mode	OFF, OFF_PERMISSIVE , ON_PERMISSIVE , ON	OFF 指定新事务是匿名事务（即，没有 GTID），并且事务必须是匿名事务才能复制。 OFF_PERMISSIVE 指定新事务是匿名事务，但可以复制所有事务。 ON_PERMISSIVE 指定新事务是 GTID 事务，但可以复制所有事务。 ON 指定新事务是 GTID 事务，并且事务必须是 GTID 事务才能复制。
enforce_gtid_consistency	OFF, ON, WARN	OFF 允许事务违反 GTID 一致性。 ON 禁止事务违反 GTID 一致性。 WARN 允许事务违反 GTID 一致性，但在违反一致性时生成警告。

> **Note**
>
> 在 AWS Management Console中，gtid_mode 参数显示为 gtid-mode。

对于基于 GTID 的复制，请为 Aurora MySQL 数据库集群的数据库集群参数组使用这些设置：

- ON 和 ON_PERMISSIVE 仅适用于从 Aurora MySQL 集群的传出复制。这两个值都可以让 Aurora 数据库集群为复制到外部数据库的事务使用 GTID。ON 要求外部数据库也使用基于 GTID 的复制。ON_PERMISSIVE 让基于 GTID 的复制成为外部数据库上的可选项。
- OFF_PERMISSIVE（如果设置）表明您的 Aurora 数据库集群可以接受来自外部数据库的传入复制。不论外部数据库是否使用基于 GTID 的复制，它都可以接受。

- OFF（如果设置）表明您的 Aurora 数据库集群只接受来自不使用基于 GTID 的复制的外部数据库的传入复制。

> **Tip**
>
> 传入复制是 Aurora MySQL 集群最常见的二进制日志复制场景。对于传入复制，建议您将 GTID 模式设置为 OFF_PERMISSIVE。该设置允许来自外部数据库的传入复制，不论复制源的 GTID 设置如何。

有关参数组的更多信息，请参阅 Amazon Aurora 的参数组。

为 Aurora MySQL 集群启用基于 GTID 的复制

在为 Aurora MySQL 数据库集群启用了基于 GTID 的复制后，GTID 设置将应用于入站和出站两种二进制日志复制。

为 Aurora MySQL 集群启用基于 GTID 的复制

1. 使用以下参数设置创建或编辑数据库集群参数组：
   - gtid_mode – ON 或 ON_PERMISSIVE
   - enforce_gtid_consistency – ON
2. 将数据库集群参数组与 Aurora MySQL 集群关联。为此，请按照 Amazon Aurora 的参数组 中的过程进行操作。
3. （可选）指定如何将 GTID 分配给不包括它们的事务。为实现此目的，请在 mysql.rds_assign_gtids_to_anonymous_transactions（Aurora MySQL 版本 3） 中调用已存储的过程。

为Aurora MySQL 数据库集群禁用基于 GTID 的复制

您可以为 Aurora MySQL 数据库集群禁用基于 GTID 的复制。这样做意味着 Aurora 集群不能对使用基于 GTID 的复制的外部数据库执行入站或出站二进制日志复制。

> **Note**
>
> 在以下过程中，只读副本 表示采用对外部数据库执行二进制日志复制的 Aurora 配置中的复制目标。它不表示只读 Aurora 副本数据库实例。例如，当 Aurora 集群接受来自外部源的传入复制时，Aurora 主实例充当二进制日志复制的只读副本。

有关此部分提到的存储过程的更多详细信息，请参阅 Aurora MySQL 存储过程参考。

为 Aurora MySQL 数据库集群禁用基于 GTID 的复制

1. 在 Aurora 副本上，运行以下过程：

   对于版本 3

   ```
   CALL mysql.rds_set_source_auto_position(0);
   ```

   对于版本 2

   ```
   CALL mysql.rds_set_master_auto_position(0);
   ```

2. 将 `gtid_mode` 重置为 `ON_PERMISSIVE`。

   a. 确保与 Aurora MySQL 集群关联的数据库集群参数组将 `gtid_mode` 设置为 `ON_PERMISSIVE`。

      有关使用参数组设置配置参数的更多信息，请参阅 Amazon Aurora 的参数组。

   b. 重启 Aurora MySQL 数据库集群。

3. 将 `gtid_mode` 重置为 `OFF_PERMISSIVE`。

   a. 确保与 Aurora MySQL 集群关联的数据库集群参数组将 `gtid_mode` 设置为 `OFF_PERMISSIVE`。

   b. 重启 Aurora MySQL 数据库集群。

4. 等待在 Aurora 主实例上应用所有 GTID 事务。要检查是否应用了这些事务，请按以下步骤操作：

   a. 在 Aurora 主实例上，运行 `SHOW MASTER STATUS` 命令。

      您的输出应类似于以下输出。

```
File                          Position
-------------------------------------
mysql-bin-changelog.000031      107
-------------------------------------
```

记下输出中的文件和位置。

b. 在每个只读副本上，使用上一步中的源实例上的文件和位置信息运行以下查询：

对于版本 3

```
SELECT SOURCE_POS_WAIT('file', position);
```

对于版本 2

```
SELECT MASTER_POS_WAIT('file', position);
```

例如，如果文件名是 mysql-bin-changelog.000031 并且位置是 107，请运行以下语句：

对于版本 3

```
SELECT SOURCE_POS_WAIT('mysql-bin-changelog.000031', 107);
```

对于版本 2

```
SELECT MASTER_POS_WAIT('mysql-bin-changelog.000031', 107);
```

5. 重置 GTID 参数以禁用基于 GTID 的复制。

a. 确保与 Aurora MySQL 集群关联的数据库集群参数组具有以下参数设置：

- gtid_mode – OFF
- enforce_gtid_consistency – OFF

b. 重启 Aurora MySQL 数据库集群。

在 Amazon Aurora MySQL 数据库集群中使用本地写入转发

本地（集群内）写入转发支持您的应用程序直接在 Aurora 副本上发出读/写事务。然后，这些事务转发到写入器数据库实例进行提交。当您的应用程序要求先写后读一致性（即能够读取事务中的最新写入内容）时，您可以使用本地写入转发。

只读副本从写入器异步接收更新。如果不进行写入转发，则必须在写入器数据库实例上处理任何要求先写后读一致性的读取。或者，您必须开发复杂的自定义应用程序逻辑，以利用多个只读副本来实现可扩展性。您的应用程序必须完全拆分所有读取和写入流量，保持两组数据库连接才能将流量发送到正确的端点。当查询是应用程序内单个逻辑会话或事务的一部分时，这种开发开销会使应用程序设计复杂化。此外，由于不同只读副本的复制滞后可能不同，因此很难在数据库中的所有实例之间实现全局读取一致性。

写入转发避免了拆分这些事务或将它们专门发送给写入器的必要性，从而简化了应用程序开发。对于需要读取事务中最新写入内容且对写入延迟不敏感的工作负载，通过这项新功能可以轻松实现读取扩展。

本地写入转发与全局写入转发不同，后者将写入从辅助数据库集群转发到 Aurora 全局数据库中的主数据库集群。您可以在属于 Aurora 全局数据库的数据库集群中使用本地写入转发。有关更多信息，请参阅 在 Amazon Aurora Global Database 中使用写入转发。

本地写入转发需要 Aurora MySQL 版本 3.04 或更高版本。

主题

- 启用本地写入转发
- 检查数据库集群是否启用了写入转发
- 应用程序和 SQL 与写入转发的兼容性
- 写入转发的隔离级别
- 写入转发的读取一致性
- 运行带写入转发的多部分语句
- 使用写入转发的事务
- 写入转发的配置参数
- 用于写入转发的 Amazon CloudWatch 指标和 Aurora MySQL 状态变量
- 识别转发的事务和查询

启用本地写入转发

原定设置情况下，Aurora MySQL 数据库集群不启用本地写入转发。您在集群级别而不是在实例级别启用本地写入转发。

> **Important**
>
> 您还可以为使用二进制日志记录的跨区域只读副本启用本地写入转发，但写入操作不会转发到源 AWS 区域。它们会被转发到二进制日志只读副本集群的写入器数据库实例。
> 仅当您有写入辅助 AWS 区域中的二进制日志只读副本的应用场景时，才使用此方法。否则，最后可能会出现“分裂大脑”的情况，即复制的数据集彼此不一致。
> 除非绝对必要，否则我们建议您对全局数据库使用全局写入转发，而不是对跨区域只读副本使用本地写入转发。有关更多信息，请参阅 在 Amazon Aurora Global Database 中使用写入转发。

控制台

使用 AWS Management Console，在创建或修改数据库集群时，选中只读副本写入转发下的开启本地写入转发复选框。

AWS CLI

要使用 AWS CLI 启用写入转发，请使用 `--enable-local-write-forwarding` 选项。当您使用 `create-db-cluster` 命令创建新的数据库集群时，此选项将起作用。当您使用 `modify-db-cluster` 命令修改现有数据库集群时，它也起作用。您可以结合这些相同的 CLI 命令使用 `--no-enable-local-write-forwarding` 选项来禁用写入转发。

以下示例创建了一个启用写入转发的 Aurora MySQL 数据库集群。

```
aws rds create-db-cluster \
  --db-cluster-identifier write-forwarding-test-cluster \
  --enable-local-write-forwarding \
  --engine aurora-mysql \
  --engine-version 8.0.mysql_aurora.3.04.0 \
  --master-username myuser \
  --master-user-password mypassword \
  --backup-retention 1
```

然后，您可以创建写入器和读取器数据库实例，以便您可以使用写入转发。有关更多信息，请参阅 创建 Amazon Aurora 数据库集群。

RDS API

要使用 Amazon RDS API 启用写入转发，请将 EnableLocalWriteForwarding 参数设置为 true。当您使用 CreateDBCluster 操作创建新的数据库集群时，此参数起作用。当您使用 ModifyDBCluster 操作修改现有数据库集群时，它也起作用。您可以通过将 EnableLocalWriteForwarding 参数设置为 false 来禁用写入转发。

为数据库会话启用写入转发

aurora_replica_read_consistency 参数是启用写入转发的数据库参数和数据库集群参数。您可以为读取一致性级别指定 EVENTUAL、SESSION 或 GLOBAL。要了解有关一致性级别的更多信息，请参阅写入转发的读取一致性。

以下规则适用于此参数：

- 原定设置值为 ''（null）。
- 仅当您将 aurora_replica_read_consistency 设置为 EVENTUAL、SESSION 或 GLOBAL 时，写入转发才可用。此参数仅适用于启用了写入转发的数据库集群的读取器实例。
- 您不能在多语句事务内设置此参数（如果为空）或取消其设置（如果已设置）。在此类事务中，您可以将其从一个有效值更改为另一个有效值，但我们不建议执行此操作。

检查数据库集群是否启用了写入转发

要确定是否可以在数据库集群中使用写入转发，请确认该集群的属性 LocalWriteForwardingStatus 是否设置为 enabled。

在 AWS Management Console中，在集群详细信息页面的配置选项卡上，您可以看到本地只读副本写入转发的状态为已启用。

要查看所有集群的写入转发设置的状态，请运行以下 AWS CLI 命令。

Example

```
aws rds describe-db-clusters \
--query '*[].
{DBClusterIdentifier:DBClusterIdentifier,LocalWriteForwardingStatus:LocalWriteForwardingStatus}

[
```

```
    {
        "LocalWriteForwardingStatus": "enabled",
        "DBClusterIdentifier": "write-forwarding-test-cluster-1"
    },
    {
        "LocalWriteForwardingStatus": "disabled",
        "DBClusterIdentifier": "write-forwarding-test-cluster-2"
    },
    {
        "LocalWriteForwardingStatus": "requested",
        "DBClusterIdentifier": "test-global-cluster-2"
    },
    {
        "LocalWriteForwardingStatus": "null",
        "DBClusterIdentifier": "aurora-mysql-v2-cluster"
    }
]
```

对于 LocalWriteForwardingStatus，数据库集群可以具有以下值：

- disabled – 写入转发已禁用。
- disabling – 正在禁用写入转发。
- enabled – 写入转发已启用。
- enabling – 正在启用写入转发。
- null – 写入转发不适用于此数据库集群。
- requested – 已请求写入转发，但尚未激活。

应用程序和 SQL 与写入转发的兼容性

您可以将以下类型的 SQL 语句与写入转发一起使用：

- 数据操作语言 (DML) 语句，如 INSERT、DELETE 和 UPDATE。对于您可以与写入转发一起使用的这些语句的属性存在一些限制，如下所述。
- SELECT ... LOCK IN SHARE MODE 和 SELECT FOR UPDATE 语句。
- PREPARE 和 EXECUTE 语句。

在具有写入转发功能的数据库集群中使用某些语句时，不允许这些语句，否则它们可能产生过时的结果。此外，不支持用户定义的函数和用户定义的过程。因此，原定设置情况下，数据库集群的

EnableLocalWriteForwarding 设置处于禁用状态。在启用该设置之前，请检查以确保您的应用程序代码不受任何这些限制的影响。

以下限制适用于与写入转发一起使用的 SQL 语句。在某些情况下，您可以在启用了写入转发的数据库集群上使用这些语句。如果 aurora_replica_read_consistency 配置参数未在会话中启用写入转发，则此方法起作用。如果您尝试在由于写入转发而不允许使用某个语句的情况下使用该语句，则会看到一条类似于以下内容的错误消息：

```
ERROR 1235 (42000): This version of MySQL doesn't yet support 'operation with write
 forwarding'.
```

数据定义语言 (DDL)

连接到写入器数据库实例以运行 DDL 语句。您不能从读取器数据库实例运行它们。

使用临时表中的数据更新永久表

您可以在启用写入转发的数据库集群上使用临时表。但是，如果语句引用临时表，则不能使用 DML 语句来修改永久表。例如，您不能使用从临时表中获取数据的 INSERT ... SELECT 语句。

XA 事务

在会话内启用写入转发时，不能在数据库集群上使用以下语句。您可以在未启用写入转发的数据库集群上使用这些语句，或者在 aurora_replica_read_consistency 设置为空的会话中使用这些语句。在会话内启用写入转发之前，请检查您的代码是否使用这些语句。

```
XA {START|BEGIN} xid [JOIN|RESUME]
XA END xid [SUSPEND [FOR MIGRATE]]
XA PREPARE xid
XA COMMIT xid [ONE PHASE]
XA ROLLBACK xid
XA RECOVER [CONVERT XID]
```

永久表的 LOAD 语句

您不能在启用写入转发的数据库集群上使用以下语句。

```
LOAD DATA INFILE 'data.txt' INTO TABLE t1;
LOAD XML LOCAL INFILE 'test.xml' INTO TABLE t1;
```

插件语句

您不能在启用写入转发的数据库集群上使用以下语句。

```
INSTALL PLUGIN example SONAME 'ha_example.so';
UNINSTALL PLUGIN example;
```

SAVEPOINT 语句

在会话内启用写入转发时，不能在数据库集群上使用以下语句。您可以在未启用写入转发的数据库集群上使用这些语句，或者在 aurora_replica_read_consistency 设置为空白的会话中使用这些语句。在会话内启用写入转发之前，请检查您的代码是否使用这些语句。

```
SAVEPOINT t1_save;
ROLLBACK TO SAVEPOINT t1_save;
RELEASE SAVEPOINT t1_save;
```

写入转发的隔离级别

在使用写入转发的会话中，您只能使用 REPEATABLE READ 隔离级别。尽管您也可以将 READ COMMITTED 隔离级别用于 Aurora 副本，但该隔离级别不适用于写入转发。有关 REPEATABLE READ 和 READ COMMITTED 隔离级别的信息，请参阅 Aurora MySQL 隔离级别。

写入转发的读取一致性

您可以控制数据库集群上的读取一致性程度。读取一致性级别确定数据库集群在每次读取操作之前要等待多少时间，以确保从主集群复制某些或所有更改。您可以调整读取一致性级别，以确保会话中所有转发的写入操作在数据库集群中都可见，然后再进行任何后续查询。您还可以使用此设置来确保数据库集群上的查询始终看到来自写入器的最新更新。此设置也适用于其他会话或其他集群提交的查询。要为应用程序指定此类行为，请为 aurora_replica_read_consistency 数据库参数或数据库集群参数选择一个值。

> **Important**
>
> 当您想要转发写入时，请始终设置 aurora_replica_read_consistency 数据库参数或数据库集群参数。否则，Aurora 不会转发写入。默认情况下，此参数的值为空，因此在使用此参数时请选择一个特定值。aurora_replica_read_consistency 参数仅影响启用了写入转发的数据库集群或实例。

如果您提高一致性级别，您的应用程序会花更多时间等待在数据库实例之间传播更改。您可以在快速响应时间与确保在运行查询之前在其他数据库实例中进行的更改完全可用之间选择平衡。

您可以为 aurora_replica_read_consistency 参数指定以下值：

- EVENTUAL – 在对写入器数据库实例执行写入操作之前，同一会话中写入操作的结果是不可见的。查询不会等待更新的结果可用。因此，它可能会检索较旧的数据或更新的数据，具体取决于语句的时间和复制滞后量。这与不使用写入转发的 Aurora MySQL 数据库集群的一致性相同。
- SESSION – 所有使用写入转发的查询都会看到在该会话中进行的所有更改的结果。无论事务是否已提交，这些更改都是可见的。如有必要，查询将等待要复制的转发写入操作的结果。
- GLOBAL – 会话会看到数据库集群中所有会话和实例上的所有已提交更改。每个查询可能会等待一段时间，该时间取决于会话滞后量。从查询开始时，如果数据库集群处于最新状态（具有来自写入器的所有已提交数据），查询将继续进行。

有关写入转发涉及的配置参数的信息，请参阅写入转发的配置参数。

> **Note**
>
> 您也可以使用 aurora_replica_read_consistency 作为会话变量，例如：
>
> ```
> mysql> set aurora_replica_read_consistency = 'session';
> ```

使用写入转发的示例

以下示例显示了 aurora_replica_read_consistency 参数对运行 INSERT 语句后跟 SELECT 语句的影响。根据 aurora_replica_read_consistency 的值和语句的时间，结果可能会有所不同。

为了实现更高的一致性，您可以在发出 SELECT 语句之前稍等一会。或者，Aurora 可以自动等到结果复制完成后再继续进行 SELECT。

有关设置数据库参数的信息，请参阅Amazon Aurora 的参数组。

Example **aurora_replica_read_consistency** 设置为 **EVENTUAL**

运行一个 INSERT 语句，紧接着运行一个 SELECT 语句，将返回一个 COUNT(*) 值，其中包含插入新行之前的行数。稍后再次运行 SELECT 将返回更新的行计数。这些 SELECT 语句不会等待。

```
mysql> select count(*) from t1;
```

```
+----------+
| count(*) |
+----------+
|        5 |
+----------+
1 row in set (0.00 sec)

mysql> insert into t1 values (6); select count(*) from t1;
+----------+
| count(*) |
+----------+
|        5 |
+----------+
1 row in set (0.00 sec)

mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        6 |
+----------+
1 row in set (0.00 sec)
```

Example **aurora_replica_read_consistency** 设置为 **SESSION**

紧随 INSERT 后的一条 SELECT 语句会等到 INSERT 语句中的更改变为可见。后续 SELECT 语句不会等待。

```
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        6 |
+----------+
1 row in set (0.01 sec)

mysql> insert into t1 values (6); select count(*) from t1; select count(*) from t1;
Query OK, 1 row affected (0.08 sec)
+----------+
| count(*) |
+----------+
|        7 |
+----------+
```

```
1 row in set (0.37 sec)
+----------+
| count(*) |
+----------+
|        7 |
+----------+
1 row in set (0.00 sec)
```

当读取一致性设置仍设置为 SESSION，如果在执行一条 INSERT 语句后引入简短的等待，则会使更新的行计数在下一条 SELECT 语句运行时可用。

```
mysql> insert into t1 values (6); select sleep(2); select count(*) from t1;
Query OK, 1 row affected (0.07 sec)
+----------+
| sleep(2) |
+----------+
|        0 |
+----------+
1 row in set (2.01 sec)
+----------+
| count(*) |
+----------+
|        8 |
+----------+
1 row in set (0.00 sec)
```

Example **aurora_replica_read_consistency** 设置为 **GLOBAL**

在执行查询之前，每条 SELECT 语句都会等待，截至该语句开始时的所有数据更改均可见。每条 SELECT 语句的等待时间各不相同，具体取决于复制滞后。

```
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        8 |
+----------+
1 row in set (0.75 sec)

mysql> select count(*) from t1;
+----------+
| count(*) |
```

```
+----------+
|        8 |
+----------+
1 row in set (0.37 sec)

mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        8 |
+----------+
1 row in set (0.66 sec)
```

运行带写入转发的多部分语句

DML 语句可能由多个部分组成，如 INSERT ... SELECT 语句或 DELETE ... WHERE 语句。在这种情况下，整个语句将转发到写入器数据库实例并在此处运行。

使用写入转发的事务

如果事务访问模式设置为只读，则不使用写入转发。您可以使用 SET TRANSACTION 语句或 START TRANSACTION 语句指定事务的访问模式。您还可以通过更改 transaction_read_only 会话变量的值来指定事务访问模式。只有在连接到启用了写入转发的数据库集群时，才能更改此会话值。

如果长时间运行的事务在很长一段时间内没有发出任何语句，则可能会超过空闲超时期限。此时段的默认值为一分钟。您可以设置 aurora_fwd_writer_idle_timeout 参数以将其增加到最多一天。超过空闲超时的事务将被写入器实例取消。您提交的下一个后续语句会收到超时错误。然后 Aurora 回滚事务。

在写入转发变得不可用的其他情况下，可能会发生此类错误。例如，如果您重启数据库集群或禁用写入转发，则 Aurora 取消使用写入转发的任何事务。

当使用本地写入转发的集群中的写入器实例重启时，使用本地写入转发的读取器实例上任何活跃的转发事务和查询都将自动关闭。在写入器实例再次可用后，您可以重试这些事务。

写入转发的配置参数

Aurora 数据库参数组包含写入转发功能的设置。下表汇总了有关这些参数的详细信息，表后附有使用说明。

参数	范围	类型	默认值	有效值
aurora_fwd_writer_idle_timeout	集群	无符号整数	60	1–86,400
aurora_fwd_writer_max_connections_pct	集群	无符号长整数	10	0–90
aurora_replica_read_consistency	集群或实例	枚举	" (null)	EVENTUAL, SESSION, GLOBAL

要控制传入的写入请求，请使用以下设置：

- aurora_fwd_writer_idle_timeout – 写入器数据库实例在关闭从读取器实例转发的连接之前等待此连接上的活动的秒数。如果会话在此期间之后仍处于空闲状态，则 Aurora 取消会话。
- aurora_fwd_writer_max_connections_pct – 可以在写入器数据库实例上用于处理从读取器实例转发的查询的数据库连接的上限。它表示为写入器的 max_connections 设置的百分比。例如，如果 max_connections 是 800，而 aurora_fwd_master_max_connections_pct 或 aurora_fwd_writer_max_connections_pct 是 10，则写入器最多允许 80 个同时转发会话。这些连接来自由 max_connections 设置管理的同一连接池。

  此设置仅适用于启用了写入转发的写入器。如果减小该值，不会影响现有连接。Aurora 尝试从数据库集群创建新连接时，会考虑账户设置的新值。默认值为 10，表示 max_connections 值的 10%。

> **Note**
>
> 由于 aurora_fwd_writer_idle_timeout 和 aurora_fwd_writer_max_connections_pct 是数据库集群参数，因此每个集群中的所有数据库实例对于这些参数都具有相同的值。

有关 aurora_replica_read_consistency 的更多信息，请参阅 写入转发的读取一致性。

有关数据库参数组的更多信息，请参阅 Amazon Aurora 的参数组。

用于写入转发的 Amazon CloudWatch 指标和 Aurora MySQL 状态变量

当您对 Aurora MySQL 使用写入转发时，以下 Amazon CloudWatch 指标和 Aurora MySQL 状态变量适用。有关 Aurora MySQL 写入器和读取器数据库实例的指标的更多信息，请参阅以下主题。

主题

- Aurora MySQL 写入器数据库实例的写入转发指标
- Aurora MySQL 读取器数据库实例的写入转发指标

Aurora MySQL 写入器数据库实例的写入转发指标

当您在一个或多个数据库集群上使用写入转发时，以下 Amazon CloudWatch 指标适用。这些指标均是在写入器数据库实例上测量的。

CloudWatch 指标	单位	描述
`ForwardingWriterDMLLatency`	毫秒	在写入器数据库实例上处理每个转发的 DML 语句的平均时间。 它不包括数据库集群转发写入请求的时间，也不包括将更改复制回写入器的时间。
`ForwardingWriterDMLThroughput`	每秒计数	此写入器数据库实例每秒处理的转发 DML 语句数。
`ForwardingWriterOpenSessions`	计数	写入器数据库实例上的转发会话数。

当您在一个或多个数据库集群上使用写入转发时，以下 Aurora MySQL 状态变量适用。这些状态变量均是在写入器数据库实例上测量的。

Aurora MySQL 状态变量	单位	描述
`Aurora_fwd_writer_dml_stmt_count`	计数	转发到此写入器数据库实例的 DML 语句总数。

Aurora MySQL 状态变量	单位	描述
Aurora_fwd_writer_dml_stmt_duration	微秒	转发到此写入器数据库实例的 DML 语句的总持续时间。
Aurora_fwd_writer_open_sessions	计数	写入器数据库实例上的转发会话数。
Aurora_fwd_writer_select_stmt_count	计数	转发到此写入器数据库实例的 SELECT 语句总数。
Aurora_fwd_writer_select_stmt_duration	微秒	转发到此写入器数据库实例的 SELECT 语句的总持续时间。

Aurora MySQL 读取器数据库实例的写入转发指标

以下 CloudWatch 指标在启用了写入转发的数据库集群中的每个读取器数据库实例上进行测量。

CloudWatch 指标	单位	描述
ForwardingReplicaDMLLatency	毫秒	副本上转发 DML 的平均响应时间。
ForwardingReplicaDMLThroughput	每秒计数	每秒处理的转发 DML 语句数。
ForwardingReplicaOpenSessions	计数	在读取器数据库实例上使用写入转发的会话数。
ForwardingReplicaReadWaitLatency	毫秒	读取器数据库实例上的 SELECT 语句等待赶上写入器的平均等待时间。 读取器数据库实例在处理查询之前等待的程度取决于 aurora_replica_read_consistency 设置。

CloudWatch 指标	单位	描述
ForwardingReplicaReadWaitThroughput	每秒计数	转发写入的所有会话中每秒处理的 SELECT 语句总数。
ForwardingReplicaSelectLatency	毫秒	已转发的 SELECT 延迟，监控期间内所有转发 SELECT 语句的平均值。
ForwardingReplicaSelectThroughput	每秒计数	已转发 SELECT 吞吐量，监控期内每秒平均值。

以下 Aurora MySQL 状态变量在启用了写入转发的数据库集群中的每个读取器数据库实例上进行测量。

Aurora MySQL 状态变量	单位	描述
Aurora_fwd_replica_dml_stmt_count	计数	从此读取器数据库实例转发的 DML 语句总数。
Aurora_fwd_replica_dml_stmt_duration	微秒	从此读取器数据库实例转发的所有 DML 语句的总持续时间。
Aurora_fwd_replica_errors_session_limit	计数	由于以下错误条件之一而被主集群拒绝的会话数。 • 写入器已满 • 正在转发的语句过多。
Aurora_fwd_replica_open_sessions	计数	在读取器数据库实例上使用写入转发的会话数。
Aurora_fwd_replica_read_wait_count	计数	此读取器数据库实例上的写入后读取等待的总数。
Aurora_fwd_replica_read_wait_duration	微秒	由于此读取器数据库实例上的读取一致性设置而导致的总等待持续时间。

Aurora MySQL 状态变量	单位	描述
Aurora_fwd_replica_select_stmt_count	计数	从此读取器数据库实例转发的 SELECT 语句总数。
Aurora_fwd_replica_select_stmt_duration	微秒	从此读取器数据库实例转发的 SELECT 语句的总持续时间。

识别转发的事务和查询

可以使用 information_schema.aurora_forwarding_processlist 表来识别转发的事务和查询。有关此表的更多信息，请参阅 information_schema.aurora_forwarding_processlist。

以下示例显示了写入器数据库实例上的所有转发连接。

```
mysql> select * from information_schema.AURORA_FORWARDING_PROCESSLIST where
 IS_FORWARDED=1 order by REPLICA_SESSION_ID;

+-----+----------+--------------------+----------+---------+------+--------------
+-------------------------------------------+--------------+--------------------
+----------------------------------+----------------------+----------------+
| ID  | USER     | HOST               | DB       | COMMAND | TIME | STATE        |
 INFO                                     | IS_FORWARDED | REPLICA_SESSION_ID |
 REPLICA_INSTANCE_IDENTIFIER     | REPLICA_CLUSTER_NAME | REPLICA_REGION |
+-----+----------+--------------------+----------+---------+------+--------------
+-------------------------------------------+--------------+--------------------
+----------------------------------+----------------------+----------------+
| 648 | myuser   | IP_address:port1   | sysbench | Query   |    0 | async commit |
 UPDATE sbtest58 SET k=k+1 WHERE id=4802579 |            1 |                637 | my-
db-cluster-instance-2        | my-db-cluster        | us-west-2      |
| 650 | myuser   | IP_address:port2   | sysbench | Query   |    0 | async commit |
 UPDATE sbtest54 SET k=k+1 WHERE id=2503953 |            1 |                639 | my-
db-cluster-instance-2        | my-db-cluster        | us-west-2      |
+-----+----------+--------------------+----------+---------+------+--------------
+-------------------------------------------+--------------+--------------------
+----------------------------------+----------------------+----------------+
```

在转发读取器数据库实例上，您可以通过运行 SHOW PROCESSLIST 来查看与这些写入器数据库连接关联的线程。写入器上的 REPLICA_SESSION_ID 值 637 和 639 与读取器上的 Id 值相同。

```
mysql> select @@aurora_server_id;

+--------------------------------+
| @@aurora_server_id             |
+--------------------------------+
| my-db-cluster-instance-2       |
+--------------------------------+
1 row in set (0.00 sec)

mysql> show processlist;

+-----+----------+--------------------+----------+---------+------+--------------
+--------------------------------------------+
| Id  | User     | Host               | db       | Command | Time | State        | Info
                                       |
+-----+----------+--------------------+----------+---------+------+--------------
+--------------------------------------------+
| 637 | myuser   | IP_address:port1   | sysbench | Query   |    0 | async commit |
 UPDATE sbtest12 SET k=k+1 WHERE id=4802579  |
| 639 | myuser   | IP_address:port2   | sysbench | Query   |    0 | async commit |
 UPDATE sbtest61 SET k=k+1 WHERE id=2503953  |
+-----+----------+--------------------+----------+---------+------+--------------
+--------------------------------------------+
12 rows in set (0.00 sec)
```

将 Amazon Aurora MySQL 与其他AWS服务集成

Amazon Aurora MySQL 与其他AWS服务集成在一起，因此，您可以扩展 Aurora MySQL 数据库集群以在AWS云中使用其他功能。您的 Aurora MySQL 数据库集群可以使用AWS服务来执行以下操作：

- 使用 AWS Lambda 或 `lambda_sync` 本机函数同步或异步地调用 `lambda_async` 函数。有关更多信息，请参阅 从 Amazon Aurora MySQL 数据库集群中调用 Lambda 函数。
- 通过使用 `LOAD DATA FROM S3` 或 `LOAD XML FROM S3` 命令，将数据从 Amazon Simple Storage Service (Amazon S3) 存储桶中存储的文本或 XML 文件加载到您的数据库集群中。有关更多信息，请参阅“将数据从 Amazon S3 存储桶中的文本文件加载到 Amazon Aurora MySQL 数据库集群”。
- 通过使用 `SELECT INTO OUTFILE S3` 命令，将数据从您的数据库集群保存到 Amazon S3 存储桶中存储的文本文件。有关更多信息，请参阅“将数据从 Amazon Aurora MySQL 数据库集群保存到 Amazon S3 存储桶中的文本文件”。
- 使用 Application Auto Scaling 自动添加或删除 Aurora 副本。有关更多信息，请参阅 Amazon Aurora Auto Scaling 与 Aurora 副本结合使用。
- 使用 Amazon Comprehend 或者 SageMaker AI 提供的各种机器学习算法执行情感分析。有关更多信息，请参阅 使用 Amazon Aurora 机器学习。

Aurora 通过使用 AWS（IAM）确保对其他AWS Identity and Access Management服务的访问能力。您可以创建具有必要权限的 IAM 角色，然后将该角色与您的数据库集群关联，来授予访问其他AWS服务的权限。有关如何允许 Aurora MySQL 数据库集群代表您访问其他AWS服务的详细信息和说明，请参阅授权 Amazon Aurora MySQL 代表您访问其他AWS服务。

授权 Amazon Aurora MySQL 代表您访问其他AWS服务

要使您的 Aurora MySQL 数据库集群能够代表您访问其他服务，请创建并配置一个 AWS Identity and Access Management（IAM）角色。该角色授权您的数据库集群中的数据库用户访问其他 AWS 服务。有关更多信息，请参阅“设置 IAM 角色以访问AWS服务”。

您还必须将 Aurora 数据库集群配置为允许与目标AWS服务的出站连接。有关更多信息，请参阅“启用从 Amazon Aurora 到其它 AWS 服务的网络通信”。

如果您这样做，您的数据库用户可以使用其他 AWS 服务执行以下操作：

- 使用 AWS Lambda 或 `lambda_sync` 本机函数同步或异步地调用 `lambda_async` 函数。或者，使用 AWS Lambda 过程异步地调用 `mysql.lambda_async` 函数。有关更多信息，请参阅“使用 Aurora MySQL 本机函数调用 Lambda 函数”。

- 通过使用 LOAD DATA FROM S3 或 LOAD XML FROM S3 语句，将数据从 Amazon S3 存储桶中存储的文本或 XML 文件加载到您的数据库集群中。有关更多信息，请参阅“将数据从 Amazon S3 存储桶中的文本文件加载到 Amazon Aurora MySQL 数据库集群”。
- 通过使用 SELECT INTO OUTFILE S3 语句，将数据从您的数据库集群保存到 Amazon S3 存储桶中存储的文本文件。有关更多信息，请参阅“将数据从 Amazon Aurora MySQL 数据库集群保存到 Amazon S3 存储桶中的文本文件”。
- 将日志数据导出到 Amazon CloudWatch Logs MySQL。有关更多信息，请参阅“将 Amazon Aurora MySQL 日志发布到 Amazon CloudWatch Logs”。
- 使用 Application Auto Scaling 自动添加或删除 Aurora 副本。有关更多信息，请参阅 Amazon Aurora Auto Scaling 与 Aurora 副本结合使用。

设置 IAM 角色以访问AWS服务

要允许您的 Aurora 数据库集群访问其他AWS服务，请执行以下操作：

1. 创建一个 IAM 策略来授予对 AWS 服务的权限。有关更多信息，请参阅以下主题。
   - 创建 IAM 策略以访问 Amazon S3 资源
   - 创建 IAM 策略以访问 AWS Lambda 资源
   - 创建 IAM 策略以访问 CloudWatch Logs 资源
   - 创建 IAM 策略以访问 AWS KMS 资源
2. 创建 IAM 角色并附加您创建的策略。有关更多信息，请参阅“创建 IAM 角色以允许 Amazon Aurora 访问AWS服务”。
3. 将该 IAM 角色与您的 Aurora 数据库集群关联。有关更多信息，请参阅 将 IAM 角色与 Amazon Aurora MySQL 数据库集群关联。

创建 IAM 策略以访问 Amazon S3 资源

Aurora 可以访问 Amazon S3 资源以加载数据，或者保存 Aurora 数据库集群中的数据。不过，您必须先创建 IAM 策略，提供允许 Aurora 访问 Amazon S3 的存储桶和对象权限。

下表列出了可代表您访问 Amazon S3 存储桶的 Aurora 功能，以及各个功能所需的最低存储桶和对象权限。

功能	存储桶权限	对象权限
LOAD DATA FROM S3	ListBucket	GetObject

功能	存储桶权限	对象权限
		GetObjectVersion
LOAD XML FROM S3	ListBucket	GetObject GetObjectVersion
SELECT INTO OUTFILE S3	ListBucket	AbortMultipartUpload DeleteObject GetObject ListMultipartUploadParts PutObject

以下策略可用于添加 Aurora 代表您访问 Amazon S3 存储桶时可能需要的权限。

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowAuroraToExampleBucket",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject",
                "s3:AbortMultipartUpload",
                "s3:ListBucket",
                "s3:DeleteObject",
                "s3:GetObjectVersion",
                "s3:ListMultipartUploadParts"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket/*",
                "arn:aws:s3:::amzn-s3-demo-bucket"
```

```
            ]
        }
    ]
}
```

> **Note**
>
> 确保添加 `Resource` 值对应的两个条目。Aurora 需要存储桶本身和存储桶内所有对象的权限。
> 根据您的使用案例，您可能不需要在示例策略中添加所有权限。也可能需要其他权限。例如，如果 Amazon S3 存储桶已加密，则您需要添加 `kms:Decrypt` 权限。

您可以按照以下步骤创建一个 IAM 策略，以便为 Aurora 提供所需的最小权限以代表您访问 Amazon S3 存储桶。要允许 Aurora 访问所有 Amazon S3 存储桶，您可以跳过这些步骤并使用 `AmazonS3ReadOnlyAccess` 或 `AmazonS3FullAccess` 预定义 IAM 策略，而不是创建自己的策略。

创建 IAM 策略来授予对您的 Amazon S3 资源的访问权限

1. 打开 IAM 管理控制台。
2. 在导航窗格中，选择策略。
3. 选择 Create policy (创建策略)。
4. 在 Visual editor (可视化编辑器) 选项卡上，选择 Choose a service (选择服务)，然后选择 S3。
5. 在 Actions (操作) 下面选择 Expand all (全部展开)，然后选择 IAM 策略所需的存储桶权限和对象权限。

   对象权限是 Amazon S3 中的对象操作的权限，需要为存储桶中的对象而不是存储桶本身授予这些权限。有关 Amazon S3 中的对象操作权限的更多信息，请参阅对象操作的权限。
6. 选择资源，然后为存储桶选择添加 ARN。
7. 在添加 ARN对话框中，提供有关资源的详细信息，然后选择添加。

   指定要允许访问的 Amazon S3 存储桶。例如，如果您希望支持 Aurora 访问名为 *`amzn-s3-demo-bucket`* 的 Amazon S3 存储桶，则将 Amazon 资源名称（ARN）的值设置为 `arn:aws:s3:::`*`amzn-s3-demo-bucket`*。
8. 如果列出了对象资源，请为对象选择Add ARN (添加 ARN)。

9. 在添加 ARN对话框中，提供有关资源的详细信息。

   对于 Amazon S3 存储桶，请指定要允许访问的 Amazon S3 存储桶。对于对象，您可以选择 Any (任意)，以便为存储桶中的任何对象授予权限。

   > **Note**
   >
   > 您可以将 Amazon 资源名称 (ARN) 设置为更具体的 ARN 值，以允许 Aurora 仅访问 Amazon S3 存储桶中的特定文件或文件夹。有关如何为 Amazon S3 定义访问策略的更多信息，请参阅管理您的 Amazon S3 资源的访问权限。

10. （可选）为 bucket (存储桶) 选择 Add ARN (添加 ARN) 以将另一个 Amazon S3 存储桶添加到策略中，然后为该存储桶重复前面的步骤。

    > **Note**
    >
    > 您可以重复该操作，以便在希望 Aurora 访问的每个 Amazon S3 存储桶的策略中添加相应的存储桶权限语句。您也可以根据需要授予访问 Amazon S3 中所有存储桶和对象的权限。

11. 选择 Review policy (查看策略)。
12. 对于 Name (名称)，请为您的 IAM 策略输入名称，例如 `AllowAuroraToExampleBucket`。在创建 IAM 角色与 Aurora 数据库集群关联时，需要使用此名称。您也可以添加可选的描述值。
13. 选择创建策略。
14. 完成 创建 IAM 角色以允许 Amazon Aurora 访问AWS服务 中的步骤。

创建 IAM 策略以访问 AWS Lambda 资源

您可以创建一个 IAM 策略，以便为 Aurora 提供所需的最小权限以代表您调用 AWS Lambda 函数。

以下策略添加 Aurora 代表您调用 AWS Lambda 函数所需的权限。

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
```

```
            "Sid": "AllowAuroraToExampleFunction",
            "Effect": "Allow",
            "Action": "lambda:InvokeFunction",
            "Resource":
 "arn:aws:lambda:<region>:<123456789012>:function:<example_function>"
        }
    ]
}
```

您可以按照以下步骤创建一个 IAM 策略，以便为 Aurora 提供所需的最小权限以代表您调用 AWS Lambda 函数。要允许 Aurora 调用所有 AWS Lambda 函数，您可以跳过这些步骤并使用预定义 `AWSLambdaRole` 策略，而不是创建自己的策略。

创建 IAM 策略以授予调用 AWS Lambda 函数的权限

1. 打开 IAM 控制台。
2. 在导航窗格中，选择策略。
3. 选择 Create policy (创建策略)。
4. 在可视化编辑器选项卡上，选择选择服务，然后选择 Lambda。
5. 在 Actions (操作) 下面选择 Expand all (全部展开)，然后选择 IAM 策略所需的 AWS Lambda 权限。

   确保选择了 `InvokeFunction`。这是允许 Amazon Aurora 调用 AWS Lambda 函数所需的最小权限。
6. 选择资源，然后为函数选择添加 ARN。
7. 在添加 ARN 对话框中，提供有关资源的详细信息。

   指定要允许访问的 Lambda 函数。例如，如果您希望允许 Aurora 访问名为 `example_function` 的 Lambda 函数，请将 ARN 值设置为 `arn:aws:lambda:::function:example_function`。

   有关如何为 AWS Lambda 定义访问策略的更多信息，请参阅 AWS Lambda 的身份验证和访问控制。
8. 或者，选择添加额外的权限以将另一个 AWS Lambda 函数添加到策略中，然后为该函数重复前面的步骤。

> **Note**
>
> 您可以重复该操作，以便在希望 Aurora 访问的每个 AWS Lambda 函数的策略中添加相应的函数权限语句。

9. 选择 Review policy (查看策略)。
10. 将名称设置为适合您的 IAM 策略的名称，例如 `AllowAuroraToExampleFunction`。在创建 IAM 角色与 Aurora 数据库集群关联时，需要使用此名称。您也可以添加可选的描述值。
11. 选择创建策略。
12. 完成 创建 IAM 角色以允许 Amazon Aurora 访问AWS服务 中的步骤。

创建 IAM 策略以访问 CloudWatch Logs 资源

Aurora 可以访问 CloudWatch Logs 以从 Aurora 数据库集群中导出审核日志数据。不过，您必须先创建一个 IAM 策略，以提供允许 Aurora 访问 CloudWatch Logs 的日志组和日志流权限。

以下策略添加 Aurora 代表您访问 Amazon CloudWatch Logs 所需的权限以及创建日志组和导出数据所需的最小权限。

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "EnableCreationAndManagementOfRDSCloudwatchLogEvents",
            "Effect": "Allow",
            "Action": [
                "logs:GetLogEvents",
                "logs:PutLogEvents"
            ],
            "Resource": "arn:aws:logs:*:*:log-group:/aws/rds/*:log-stream:*"
        },
        {
            "Sid":
 "EnableCreationAndManagementOfRDSCloudwatchLogGroupsAndStreams",
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
```

```
                "logs:DescribeLogStreams",
                "logs:PutRetentionPolicy",
                "logs:CreateLogGroup"
            ],
            "Resource": "arn:aws:logs:*:*:log-group:/aws/rds/*"
        }
    ]
}
```

您可以修改策略中的 ARN 以限制对特定 AWS 区域和账户的访问。

您可以执行以下步骤以创建一个 IAM 策略，以便提供 Aurora 代表您访问 CloudWatch Logs 所需的最小权限。要允许 Aurora 对 CloudWatch Logs 进行完全访问，您可以跳过这些步骤并使用 `CloudWatchLogsFullAccess` 预定义 IAM 策略，而不是创建自己的策略。有关更多信息，请参阅 Amazon CloudWatch 用户指南 中的为 CloudWatch Logs 使用基于身份的策略（IAM 策略）。

创建 IAM 策略来授予对您的 CloudWatch Logs 资源的访问权限

1. 打开 IAM 控制台。
2. 在导航窗格中，选择策略。
3. 选择 Create policy (创建策略)。
4. 在可视化编辑器选项卡上，选择选择服务，然后选择 CloudWatch Logs。
5. 在 Actions (操作) 下面选择 Expand all (全部展开)（位于右侧），然后选择 IAM 策略所需的 Amazon CloudWatch Logs 权限。

   确保选择了以下权限：

   - `CreateLogGroup`
   - `CreateLogStream`
   - `DescribeLogStreams`
   - `GetLogEvents`
   - `PutLogEvents`
   - `PutRetentionPolicy`
6. 选择资源，然后为 log-group 选择添加 ARN。
7. 在 Add ARN(s) (添加 ARN) 对话框中，输入以下值：

   - 区域 – 一个 AWS 区域或 *

- 账户 – 账号或 *
- 日志组名称 – /aws/rds/*

8. 在 Add ARN(s) (添加 ARN) 对话框中，选择 Add (添加)。
9. 为 log-stream 选择添加 ARN。
10. 在 Add ARN(s) (添加 ARN) 对话框中，输入以下值：

- 区域 – 一个 AWS 区域或 *
- 账户 – 账号或 *
- 日志组名称 – /aws/rds/*
- 日志流名称 – *

11. 在 Add ARN(s) (添加 ARN) 对话框中，选择 Add (添加)。
12. 选择 Review policy (查看策略)。
13. 将名称设置为适合您的 IAM 策略的名称，例如 AmazonRDSCloudWatchLogs。在创建 IAM 角色与 Aurora 数据库集群关联时，需要使用此名称。您也可以添加可选的描述值。
14. 选择创建策略。
15. 完成 创建 IAM 角色以允许 Amazon Aurora 访问AWS服务 中的步骤。

创建 IAM 策略以访问 AWS KMS 资源

Aurora 可以访问用于加密其数据库备份的 AWS KMS keys 密钥。不过，您必须先创建 IAM 策略来提供允许 Aurora 访问 KMS 密钥的权限。

以下策略可用于添加 Aurora 代表您访问 KMS 密钥所需的权限。

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": "arn:aws:kms:<region>:<123456789012>:key/<key-ID>"
        }
```

```
    ]
}
```

您可以执行以下步骤创建一个 IAM 策略，以便提供 Aurora 代表您访问 KMS 密钥所需的最小权限。

创建 IAM 策略来授予对您的 KMS 密钥的访问权限

1. 打开 IAM 控制台。
2. 在导航窗格中，选择策略。
3. 选择 Create policy (创建策略)。
4. 在可视化编辑器选项卡上，选择选择服务，然后选择 KMS。
5. 在 Actions (操作) 中，选择 Write (写入)，然后选择 Decrypt (解密)。
6. 依次选择资源和添加 ARN。
7. 在 Add ARN(s) (添加 ARN) 对话框中，输入以下值：
   - 区域 – 键入AWS区域，如 `us-west-2`。
   - 账户 – 键入用户账号。
   - 日志流名称 – 键入 KMS 密钥标识符。
8. 在 Add ARN(s) (添加 ARN) 对话框中，选择 Add (添加)。
9. 选择 Review policy (查看策略)。
10. 将名称设置为适合您的 IAM 策略的名称，例如 `AmazonRDSKMSKey`。在创建 IAM 角色与 Aurora 数据库集群关联时，需要使用此名称。您也可以添加可选的描述值。
11. 选择创建策略。
12. 完成 创建 IAM 角色以允许 Amazon Aurora 访问AWS服务 中的步骤。

创建 IAM 角色以允许 Amazon Aurora 访问AWS服务

在创建 IAM 策略以允许 Aurora 访问AWS资源后，您必须创建一个 IAM 角色并将 IAM 策略与新 IAM 角色关联起来。

要创建 IAM 角色以允许您的 Amazon RDS 集群代表您与其他AWS服务通信，请执行以下步骤。

创建 IAM 角色以允许 Amazon RDS 访问AWS服务

1. 打开 IAM 控制台。

2. 在导航窗格中，选择角色。
3. 选择 Create role (创建角色)。
4. 在 AWS service 下，选择 RDS。
5. 在 Select your use case (选择您的用例) 下，选择 RDS – Add Role to Database (RDS - 将角色添加到数据库)。
6. 选择下一步。
7. 在 Permissions policies（权限策略）页上的 Search（搜索）字段中输入策略的名称。
8. 当其显示在列表中时，选择您之前使用以下一个部分中的说明所定义的策略：
   - 创建 IAM 策略以访问 Amazon S3 资源
   - 创建 IAM 策略以访问 AWS Lambda 资源
   - 创建 IAM 策略以访问 CloudWatch Logs 资源
   - 创建 IAM 策略以访问 AWS KMS 资源
9. 选择下一步。
10. 在 Role name (角色名称) 中，输入 IAM 角色的名称，例如 `RDSLoadFromS3`。您也可以添加可选的描述值。
11. 选择 Create role (创建角色)。
12. 完成 将 IAM 角色与 Amazon Aurora MySQL 数据库集群关联 中的步骤。

将 IAM 角色与 Amazon Aurora MySQL 数据库集群关联

要允许 Amazon Aurora 数据库集群中的数据库用户访问其他 AWS 服务，您可以将在创建 IAM 角色以允许 Amazon Aurora 访问AWS服务中创建的 IAM 角色与该数据库集群关联。您还可以通过直接关联此服务让 AWS 创建新的 IAM 角色。

> **Note**
>
> 您无法将 IAM 角色与 Aurora Serverless v1 数据库集群关联。有关更多信息，请参阅 使用 Amazon Aurora Serverless v1。
> 您可以将 IAM 角色与 Aurora Serverless v2 数据库集群关联。

要将 IAM 角色与数据库集群关联，您可以执行两个操作：

1. 通过使用 RDS 控制台、add-role-to-db-cluster AWS CLI 命令或 AddRoleToDBCluster RDS API 操作，将角色添加到数据库集群的关联角色列表中。

   每个 Aurora 数据库集群最多可以添加 5 个 IAM 角色。

2. 将相关的AWS服务的集群级参数设置为关联的 IAM 角色的 ARN。

   下表介绍了用于访问其他 AWS 服务的 IAM 角色的集群级参数名称。

集群级参数	描述	
aws_default_lambda_role	在从您的数据库集群调用 Lambda 函数时使用。	
aws_default_logs_role	将您的日志数据从数据库集群导出至 Amazon CloudWatch Logs 时，不再需要此参数。Aurora MySQL 现在使用服务相关角色提供所需权限。有关服务相关角色的更多信息，请参阅将服务相关角色用于 Amazon Aurora。	
aws_default_s3_role	从数据库集群调用 LOAD DATA FROM S3、LOAD XML FROM S3 或 SELECT INTO OUTFILE S3 语句时使用。 在 Aurora MySQL 版本 2 中，如果没有为相应语句的 aurora_load_from_s3_role 或 aurora_select_into_s3_role 指定 IAM 角色，则使用在该参数中指定的 IAM 角色。 在 Aurora MySQL 版本 3 中，始终使用为该参数指定的 IAM 角色。	
aurora_load_from_s3_role	从数据库集群中调用 LOAD DATA FROM S3 或 LOAD XML FROM S3 语句时使用。如果没有为该参数指定 IAM 角色，则使用在 aws_default_s3_role 中指定的 IAM 角色。	

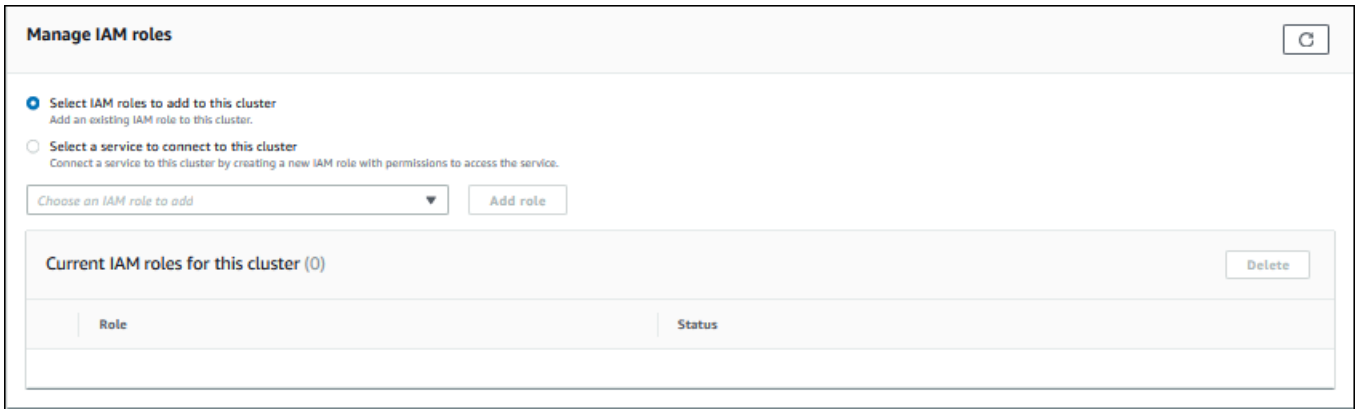
集群级参数	描述	
	在 Aurora MySQL 版本 3 中，此参数不可用。	
aurora_select_into_s3_role	从数据库集群中调用 SELECT INTO OUTFILE S3 语句时使用。如果没有为该参数指定 IAM 角色，则使用在 aws_default_s3_role 中指定的 IAM 角色。 在 Aurora MySQL 版本 3 中，此参数不可用。	

要关联 IAM 角色以允许您的 Amazon RDS 集群代表您与其他AWS服务通信，请执行以下步骤。

控制台

使用控制台将 IAM 角色与 Aurora 数据库集群关联

1. 打开 RDS 控制台：https://console.aws.amazon.com/rds/。
2. 选择数据库。
3. 选择要与 IAM 角色关联以显示其详细信息的 Aurora 数据库集群的名称。
4. 在 Connectivity & security（连接和安全）选项卡的 Manage IAM roles（管理 IAM 角色）部分中，执行以下操作之一：
   - Select IAM roles to add to this cluster（选择要添加到此集群的 IAM 角色）（原定设置）
   - Select a service to connect to this cluster（选择一项服务连接到此集群）



5. 要使用现有 IAM 角色，请从菜单中选择该角色，然后选择 Add role（添加角色）。

如果成功添加了角色，则其状态显示为 Pending，然后变为 Available。

6. 要直接连接服务，请执行以下操作：

   a. 选择 Select a service to connect to this cluster（选择一项服务连接到此集群）。

   b. 从菜单中选择服务，然后选择 Connect service（连接服务）。

   c. 在 Connect cluster to ***Service Name***（将集群连接到 Service Name）中，输入用于连接服务的 Amazon 资源名称（ARN），然后选择 Connect service（连接服务）。

   AWS 创建用于连接服务的新 IAM 角色。其状态显示为 Pending，然后变为 Available。

7. （可选）要停止将 IAM 角色与数据库集群关联并删除相关的权限，请选择该角色并选择 Delete（删除）。

为关联的 IAM 角色设置集群级参数

1. 在 RDS 控制台中，选择导航窗格中的参数组。
2. 如果您已在使用自定义数据库参数组，您可以选择使用该组而不是创建新的数据库集群参数组。如果您正在使用默认数据库集群参数组，请创建一个新的数据库集群参数组，如下列步骤中所述：

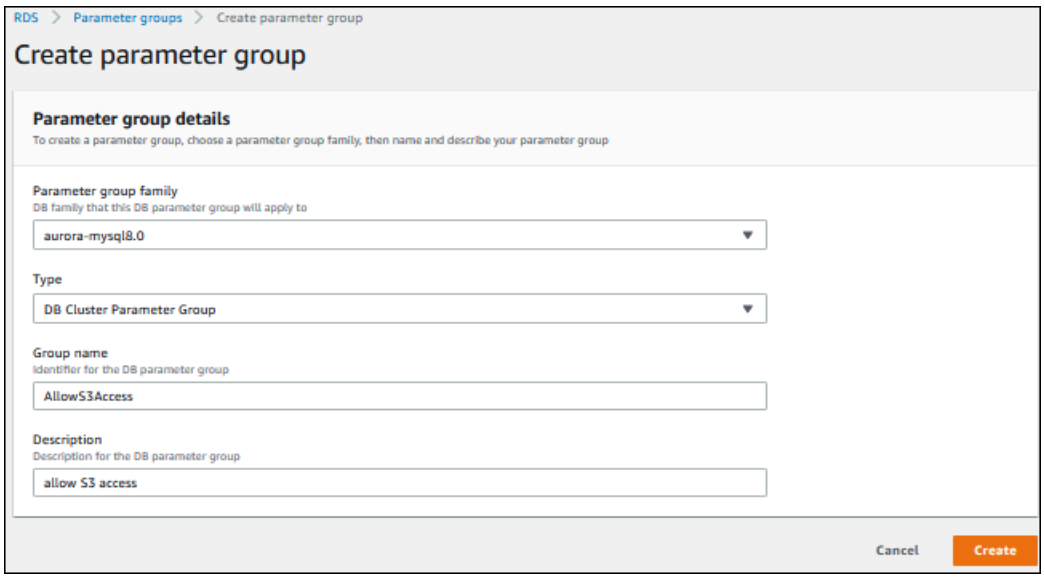
   a. 选择创建参数组。

   b. 对于参数组系列，请为 Aurora MySQL 8.0 兼容数据库集群选择 aurora-mysql8.0，或者为 Aurora MySQL 5.7 兼容数据库集群选择 aurora-mysql5.7。

   c. 对于类型，请选择数据库集群参数组。

   d. 对于组名，键入您的新数据库集群参数组的名称。

   e. 对于 Description，键入您的新数据库集群参数组的描述。



f. 选择创建。

3. 在 Parameter groups (参数组) 页面上，选择您的数据库集群参数组，并为 Parameter group actions (参数组操作) 选择 Edit (编辑)。
4. 将适当的集群级参数设置为相关的 IAM 角色 ARN 值。

   例如，可以只将 `aws_default_s3_role` 参数设置为 `arn:aws:iam::123456789012:role/AllowS3Access`。
5. 选择保存更改。
6. 要更改您的数据库集群的数据库集群参数组，请完成以下步骤：

   a. 选择 Databases (数据库)，然后选择 Aurora 数据库集群。

   b. 选择 Modify (修改)。

   c. 滚动到 Database options (数据库选项)，然后将 DB cluster parameter group (数据库集群参数组) 设置为数据库集群参数组。

   d. 选择 Continue (继续)。

   e. 验证您的更改，然后选择立即应用。

   f. 选择修改集群。

   g. 选择 Databases (数据库)，然后为您的数据库集群选择主实例。

   h. 对于操作，选择重启。

   在重新启动实例后，您的 IAM 角色将与数据库集群关联。

   有关集群参数组的更多信息，请参阅 Aurora MySQL 配置参数。

CLI

使用 AWS CLI 将 IAM 角色与数据库集群关联

1. 从 `add-role-to-db-cluster` 中调用 AWS CLI 命令，以将 IAM 角色的 ARN 添加到数据库集群中，如下所示。

```
PROMPT> aws rds add-role-to-db-cluster --db-cluster-identifier my-cluster --role-
arn arn:aws:iam::123456789012:role/AllowAuroraS3Role
PROMPT> aws rds add-role-to-db-cluster --db-cluster-identifier my-cluster --role-
arn arn:aws:iam::123456789012:role/AllowAuroraLambdaRole
```

2. 如果您正在使用默认数据库集群参数组，请创建一个新的数据库集群参数组。如果您已在使用自定义数据库参数组，您可以使用该组而不是创建新的数据库集群参数组。

   要创建新的数据库集群参数组，请从 `create-db-cluster-parameter-group` 中调用 AWS CLI 命令，如下所示。

```
PROMPT> aws rds create-db-cluster-parameter-group  --db-cluster-parameter-group-
name AllowAWSAccess \
     --db-parameter-group-family aurora5.7 --description "Allow access to Amazon S3
 and AWS Lambda"
```

   对于 Aurora MySQL 5.7 兼容数据库集群，请为 `aurora-mysql5.7` 指定 `--db-parameter-group-family`。对于兼容 Aurora MySQL 8.0 的数据库集群，请为 `--db-parameter-group-family` 指定 `aurora-mysql8.0`。

3. 在数据库集群参数组中设置相应的集群级参数以及相关的 IAM 角色 ARN 值，如下所示。

```
PROMPT> aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-name
 AllowAWSAccess \
    --parameters
 "ParameterName=aws_default_s3_role,ParameterValue=arn:aws:iam::123456789012:role/
AllowAuroraS3Role,method=pending-reboot" \
    --parameters
 "ParameterName=aws_default_lambda_role,ParameterValue=arn:aws:iam::123456789012:role/
AllowAuroraLambdaRole,method=pending-reboot"
```

4. 修改数据库集群以使用新的数据库集群参数组，然后重新启动集群，如下所示。

```
PROMPT> aws rds modify-db-cluster --db-cluster-identifier my-cluster --db-cluster-
parameter-group-name AllowAWSAccess
```

```
PROMPT> aws rds reboot-db-instance --db-instance-identifier my-cluster-primary
```

在重新启动实例后，您的 IAM 角色将与数据库集群关联。

有关集群参数组的更多信息，请参阅 Aurora MySQL 配置参数。

启用从 Amazon Aurora 到其它 AWS 服务的网络通信

要将某些其他AWS服务与 Amazon Aurora 一起使用，您的 Aurora 数据库集群的网络配置必须允许与这些服务的终端节点的出站连接。以下操作需要此网络配置。

- 调用 AWS Lambda 函数。要了解此功能，请参阅使用 Aurora MySQL 本机函数调用 Lambda 函数。
- 访问来自 Amazon S3 的文件。要了解此功能，请参阅将数据从 Amazon S3 存储桶中的文本文件加载到 Amazon Aurora MySQL 数据库集群和将数据从 Amazon Aurora MySQL 数据库集群保存到 Amazon S3 存储桶中的文本文件。
- 访问 AWS KMS 终端节点。将数据库活动流与 Aurora MySQL 结合使用需要 AWS KMS 访问权限。要了解此特征，请参阅使用数据库活动流监控 Amazon Aurora。
- 访问 SageMaker AI 端点。将 SageMaker AI 机器学习与 Aurora MySQL 结合使用时需要访问 SageMaker AI。要了解此特征，请参阅将 Amazon Aurora 机器学习与 Aurora MySQL 结合使用。

如果 Aurora 无法连接到服务终端节点，它将返回以下错误消息。

```
ERROR 1871 (HY000): S3 API returned error: Network Connection
```

```
ERROR 1873 (HY000): Lambda API returned error: Network Connection. Unable to connect to
 endpoint
```

```
ERROR 1815 (HY000): Internal error: Unable to initialize S3Stream
```

对于使用 Aurora MySQL 的数据库活动流，如果数据库集群无法访问 AWS KMS 终端节点，活动流将停止运行。Aurora 会使用 RDS 事件通知您此问题。

如果您在使用相应的AWS服务时遇到这些消息，请检查您的 Aurora 数据库集群是公有的还是私有的。如果您的 Aurora 数据库集群是私有的，您必须将其配置为允许连接。

对于要设为公有的 Aurora 数据库集群，必须将其标记为可公开访问。在这种情况下，如果在 AWS Management Console 中查看数据库集群的详细信息，就会发现公开访问设置为是。数据库集群还必须在 Amazon VPC 公有子网中。有关可公开访问的数据库实例的更多信息，请参阅在 VPC 中使用数据库集群。有关公有 Amazon VPC 子网的更多信息，请参阅您的 VPC 和子网。

如果您的 Aurora 数据库集群不可公开访问且位于 VPC 公有子网中，则它是私有的。您可能拥有一个私有的数据库集群，并希望使用需要此网络配置的功能之一。如果是这样，请配置集群，以便它可以通过网络地址转换 (NAT) 连接到 Internet 地址。作为 Amazon S3、Amazon SageMaker AI 和 AWS Lambda 的替代方案，您可以改为配置 VPC，让一个 VPC 端点用于与数据库集群的路由表关联的其他服务，请参阅在 VPC 中使用数据库集群。有关在 VPC 中配置 NAT 的更多信息，请参阅 NAT 网关。有关配置 VPC 终端节点的更多信息，请参阅 VPC 终端节点。您还可以创建一个 S3 网关端点来访问您的 S3 桶。有关更多信息，请参阅用于 Amazon S3 的网关端点。

您可能还必须在 VPC 安全组的出站规则中为网络访问控制列表 (ACL) 打开临时端口。有关网络 ACL 的临时端口的更多信息，请参阅《Amazon Virtual Private Cloud 用户指南》中的临时端口。

相关 主题

- 将 Aurora 与其他AWS服务集成
- 管理 Amazon Aurora 数据库集群

将数据从 Amazon S3 存储桶中的文本文件加载到 Amazon Aurora MySQL 数据库集群

您可以使用 `LOAD DATA FROM S3` 或 `LOAD XML FROM S3` 语句从 Amazon S3 存储桶上存储的文件中加载数据。在 Aurora MySQL 中，文件首先存储在本地磁盘上，然后导入到数据库。完成向数据库导入后，将删除本地文件。

> **Note**
>
> 对于 Aurora Serverless v1，不支持将数据从文本文件加载到表中。Aurora Serverless v2 支持此功能。

目录

- 为 Aurora 授予 Amazon S3 的访问权限
- 授予权限以在 Amazon Aurora MySQL 中加载数据

- 指定 Amazon S3 桶的路径（URI）
- LOAD DATA FROM S3
  - 语法
  - 参数
  - 使用清单指定要加载的数据文件
    - 使用 aurora_s3_load_history 表验证已加载的文件
  - 示例
- LOAD XML FROM S3
  - 语法
  - 参数

为 Aurora 授予 Amazon S3 的访问权限

您必须先为 Aurora MySQL 数据库集群授予 Amazon S3 的访问权限，然后才能从 Amazon S3 存储桶中加载数据。

为 Amazon S3 授予 Aurora MySQL 的访问权限

1. 创建一个 AWS Identity and Access Management（IAM）策略，以提供允许 Aurora MySQL 数据库集群访问 Amazon S3 存储桶和对象的权限。有关说明，请参阅创建 IAM 策略以访问 Amazon S3 资源。

   > **Note**
   >
   > 在 Aurora MySQL 版本 3.05 及更高版本中，您可以加载使用客户管理的 AWS KMS keys 进行加密的对象。为此，请在您的 IAM policy 中包含 `kms:Decrypt` 权限。有关更多信息，请参阅 创建 IAM 策略以访问 AWS KMS 资源。
   > 您不需要此权限即可使用 AWS 托管式密钥或 Amazon S3 管理密钥（SSE-S3）来加载对象。

2. 创建一个 IAM 角色，并将您在 创建 IAM 策略以访问 Amazon S3 资源 中创建的 IAM 策略附加到新的 IAM 角色。有关说明，请参阅创建 IAM 角色以允许 Amazon Aurora 访问AWS服务。
3. 确保数据库集群使用的是自定义数据库集群参数组。

   有关创建自定义数据库集群参数组的更多信息，请参阅在 Amazon Aurora 中创建数据库集群参数组。

4. 对于 Aurora MySQL 版本 2，将 `aurora_load_from_s3_role` 或 `aws_default_s3_role` 数据库集群参数设置为新 IAM 角色的 Amazon 资源名称（ARN）。如果没有为 `aurora_load_from_s3_role` 指定 IAM 角色，则 Aurora 使用在 `aws_default_s3_role` 中指定的 IAM 角色。

   对于 Aurora MySQL 版本 3，使用 `aws_default_s3_role`。

   如果集群是 Aurora 全局数据库的一部分，则为该全局数据库中的每个 Aurora 集群设置此参数。虽然只有 Aurora 全局数据库中的主集群可以加载数据，但故障转移机制可能会提升另一个集群，使其成为主集群。

   有关数据库集群参数的更多信息，请参阅Amazon Aurora 数据库集群和数据库实例参数。

5. 要允许 Aurora MySQL 数据库集群中的数据库用户访问 Amazon S3，请将您在创建 IAM 角色以允许 Amazon Aurora 访问AWS服务中创建的角色与该数据库集群关联。对于 Aurora 全局数据库，将此角色与该全局数据库中的每个 Aurora 集群关联。有关将 IAM 角色与数据库集群关联的信息，请参阅将 IAM 角色与 Amazon Aurora MySQL 数据库集群关联。

6. 配置 Aurora MySQL 数据库集群以允许建立到 Amazon S3 的出站连接。有关说明，请参阅启用从 Amazon Aurora 到其它 AWS 服务的网络通信。

   如果您的 数据库集群不可公开访问且位于 VPC 公有子网中，则它是私有的。您可以创建一个 S3 网关端点来访问您的 S3 桶。有关更多信息，请参阅用于 Amazon S3 的网关端点。

   对于 Aurora 全局数据库，为该全局数据库中的每个 Aurora 集群启用出站连接。

授予权限以在 Amazon Aurora MySQL 中加载数据

发出 `LOAD DATA FROM S3` 或 `LOAD XML FROM S3` 语句的数据库用户必须具有特定角色或权限才能发出任一语句。在 Aurora MySQL 版本 3 中，您可以授予 `AWS_LOAD_S3_ACCESS` 角色。在 Aurora MySQL 版本 2 中，您可以授予 `LOAD FROM S3` 权限。预设情况下，将为数据库集群的管理用户授予适当的角色或权限。您可以使用以下语句之一向另一个用户授予权限。

对 Aurora MySQL 版本 3 使用以下语句：

```
GRANT AWS_LOAD_S3_ACCESS TO 'user'@'domain-or-ip-address'
```

> **Tip**
>
> 当您使用 Aurora MySQL 版本 3 中的角色方法时，还可以通过使用 `SET ROLE role_name` 或 `SET ROLE ALL` 语句来激活角色。如果您不熟悉 MySQL 8.0 角色系统，可以在基于角色的权限模型中了解详情。有关更多详细信息，请参阅《MySQL 参考手册》中的 Using roles。这仅适用于当前的活动会话。当您重新连接时，必须再次运行 `SET ROLE` 语句来授予权限。有关更多信息，请参阅《MySQL 参考手册》中的 SET ROLE 语句。
> 可以使用 `activate_all_roles_on_login` 数据库集群参数，在用户连接到数据库实例时自动激活所有角色。设置此参数后，您通常不必显式调用 `SET ROLE` 语句，即可激活角色。有关更多信息，请参阅《MySQL 参考手册》中的 activate_all_roles_on_login。
> 但是，当存储过程由其他用户调用时，您必须在该存储过程的开头显式调用 `SET ROLE ALL` 才能激活该角色。

对 Aurora MySQL 版本 2 使用以下语句：

```
GRANT LOAD FROM S3 ON *.* TO 'user'@'domain-or-ip-address'
```

`AWS_LOAD_S3_ACCESS` 角色和 `LOAD FROM S3` 权限特定于 Amazon Aurora，而不适用于外部 MySQL 数据库或 RDS for MySQL 数据库实例。如果您在作为复制源的 Aurora 数据库集群和作为复制客户端的 MySQL 数据库之间设置了复制，则角色或权限的 `GRANT` 语句将导致复制停止并出现错误。您可以安全地跳过该错误，继续复制。要跳过 RDS for MySQL 实例上的错误，请使用 mysql_rds_skip_repl_error 过程。要跳过外部 MySQL 数据库上的错误，请使用 slave_skip_errors 系统变量（Aurora MySQL 版本 2）或 replica_skip_errors 系统变量（Aurora MySQL 版本 3）。

> **Note**
>
> 数据库用户必须对正在向其加载数据的数据库具有 `INSERT` 权限。

指定 Amazon S3 桶的路径（URI）

用于指定 Amazon S3 桶中文件路径（URI）的语法如下所示。

```
s3-region://amzn-s3-demo-bucket/file-name-or-prefix
```

路径包括以下值：

- `region`（可选）– 包含作为加载来源的 Amazon S3 存储桶的AWS区域。该值为可选项。如果您没有指定 `region` 值，则 Aurora 从与您的数据库集群位于相同区域的 Amazon S3 中加载您的文件。
- `bucket-name` – 包含要加载的数据的 Amazon S3 存储桶的名称。支持指定虚拟文件夹路径的对象前缀。
- `file-name-or-prefix` – Amazon S3 文本文件或 XML 文件的名称，或指定要加载的一个或多个文本或 XML 文件的前缀。您还可以指定一个清单文件以指定一个或多个要加载的文本文件。有关使用清单文件从 Amazon S3 中加载文本文件的更多信息，请参阅使用清单指定要加载的数据文件。

复制 S3 桶中文件的 URI

1. 登录到AWS Management Console，然后通过以下网址打开 Amazon S3 控制台：https://console.aws.amazon.com/s3/。
2. 在导航窗格中，选择桶，然后选择要复制其 URI 的桶。
3. 选择要从 S3 加载的前缀或文件。
4. 选择复制 S3 URI。

LOAD DATA FROM S3

您可以使用 `LOAD DATA FROM S3` 语句从 MySQL LOAD DATA INFILE 语句支持的任意文本文件格式加载数据，例如逗号分隔的文本数据。不支持压缩文件。

> **Note**
>
> 确保您的 Aurora MySQL 数据库集群允许与 S3 建立出站连接。有关更多信息，请参阅 启用从 Amazon Aurora 到其它 AWS 服务的网络通信。

语法

```
LOAD DATA [FROM] S3 [FILE | PREFIX | MANIFEST] 'S3-URI'
    [REPLACE | IGNORE]
    INTO TABLE tbl_name
    [PARTITION (partition_name,...)]
    [CHARACTER SET charset_name]
    [{FIELDS | COLUMNS}
        [TERMINATED BY 'string']
        [[OPTIONALLY] ENCLOSED BY 'char']
```

```
    [ESCAPED BY 'char']
]
[LINES
    [STARTING BY 'string']
    [TERMINATED BY 'string']
]
[IGNORE number {LINES | ROWS}]
[(col_name_or_user_var,...)]
[SET col_name = expr,...]
```

> **Note**
>
> 在 Aurora MySQL 版本 3.05 及更高版本中，关键字 FROM 是可选的。

参数

LOAD DATA FROM S3 语句使用以下必需和可选参数。在 MySQL 文档的 LOAD DATA 语句中可以找到有关其中一些参数的更多详细信息。

FILE | PREFIX | MANIFEST

指定是从单个文件、与给定前缀匹配的所有文件还是从指定清单上的所有文件中加载数据。默认值为 FILE。

S3-URI

指定要加载的文本或清单文件的 URI，或指定要使用的 Amazon S3 前缀。使用指定 Amazon S3 桶的路径（URI）中描述的语法指定 URI。

REPLACE | IGNORE

确定在输入行与数据库表中现有行具有相同唯一键值时采取什么操作。

- 如果您希望使用输入行替换表中的现有行，则指定 REPLACE。
- 如果您希望放弃输入行，则指定 IGNORE。

INTO TABLE

指定将输入行加载到的数据库表的名称。

PARTITION

要求将所有输入行插入到由指定分区名称列表（以逗号分隔）指定的分区中。如果输入行无法插入到指定分区之一，则语句失败并返回错误。

CHARACTER SET

指定输入文件中的数据的字符集。

FIELDS | COLUMNS

指定如何分隔输入文件中的字段或列。默认情况下使用制表符分隔字段。

LINES

指定如何分隔输入文件中的行。默认情况下，行由换行符 ('\n') 分隔。

IGNORE ***number*** LINES | ROWS

指定忽略输入文件开头的特定行数。例如，您可以使用 IGNORE 1 LINES 跳过包含列名的初始标题行，或者使用 IGNORE 2 ROWS 跳过输入文件的前两行数据。如果您还使用 PREFIX，则IGNORE 会跳过第一个输入文件开头的特定数量的行。

col_name_or_user_var, ...

指定一个或多个列名的逗号分隔列表，或者指定用户变量以按照名称指定要加载的列。用于此目的的用户变量的名称必须与文本文件中的元素名称匹配，前缀为 @。您可以采用用户变量来存储对应的字段值，以便在后面重复使用。

例如，以下语句将输入文件的第一列加载到 table1 的第一列，并将 table_column2 中的 table1 列值设置为第二列输入值除以 100。

```
LOAD DATA FROM S3 's3://amzn-s3-demo-bucket/data.txt'
    INTO TABLE table1
    (column1, @var1)
    SET table_column2 = @var1/100;
```

SET

指定以逗号分隔的分配操作列表，这些操作将表中各列的值设置为输入文件中未包含的值。

例如，以下语句将 table1 的前两列设置为输入文件中前两列的值，然后将 column3 中 table1 的值设置为当前时间戳。

```
LOAD DATA FROM S3  's3://amzn-s3-demo-bucket/data.txt'
    INTO TABLE table1
    (column1, column2)
    SET column3 = CURRENT_TIMESTAMP;
```

您可以在 SET 分配的右侧使用子查询。如果子查询返回要分配到列的值，则您只能使用标量子查询。此外，您还可以使用子查询来从所加载的表中选择。

如果从 Amazon S3 桶中加载数据，则无法使用 LOAD DATA FROM S3 语句的 LOCAL 关键字。

使用清单指定要加载的数据文件

可以将 LOAD DATA FROM S3 语句与 MANIFEST 关键字配合使用来指定 JSON 格式的清单文件，此文件列出了要加载到数据库集群中的表的文本文件。

以下 JSON 架构描述了清单文件的格式和内容。

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "additionalProperties": false,
    "definitions": {},
    "id": "Aurora_LoadFromS3_Manifest",
    "properties": {
        "entries": {
            "additionalItems": false,
            "id": "/properties/entries",
            "items": {
                "additionalProperties": false,
                "id": "/properties/entries/items",
                "properties": {
                    "mandatory": {
                        "default": "false",
                        "id": "/properties/entries/items/properties/mandatory",
                        "type": "boolean"
                    },
                    "url": {
                        "id": "/properties/entries/items/properties/url",
                        "maxLength": 1024,
                        "minLength": 1,
                        "type": "string"
                    }
                },
                "required": [
                    "url"
                ],
                "type": "object"
            },
            "type": "array",
```

```
            "uniqueItems": true
        }
    },
    "required": [
        "entries"
    ],
    "type": "object"
}
```

清单中的每个 url 必须指定带存储桶名称和文件的完整对象路径 (而不仅仅是前缀) 的 URL。您可以使用清单来加载来自不同存储桶或不同区域的文件，或加载未共享相同前缀的文件。如果 URL 中未指定区域，将使用目标 Aurora 数据库集群的区域。以下示例显示一个清单文件，此文件加载来自不同存储桶的四个文件。

```
{
  "entries": [
    {
      "url":"s3://aurora-bucket/2013-10-04-customerdata",
      "mandatory":true
    },
    {
      "url":"s3-us-west-2://aurora-bucket-usw2/2013-10-05-customerdata",
      "mandatory":true
    },
    {
      "url":"s3://aurora-bucket/2013-10-04-customerdata",
      "mandatory":false
    },
    {
      "url":"s3://aurora-bucket/2013-10-05-customerdata"
    }
  ]
}
```

可选的 mandatory 标志指定 LOAD DATA FROM S3 是否应在找不到文件时返回错误。mandatory 标志默认为 false。不管如何设置 mandatory，如果找不到文件，则 LOAD DATA FROM S3 将终止。

清单文件可具有任何扩展名。以下示例使用前一个示例中名为 LOAD DATA FROM S3 的清单以运行 **customer.manifest** 语句。

```
LOAD DATA FROM S3 MANIFEST 's3-us-west-2://aurora-bucket/customer.manifest'
```

```
    INTO TABLE CUSTOMER
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n'
    (ID, FIRSTNAME, LASTNAME, EMAIL);
```

该语句完成后，将为每个成功加载的文件向 aurora_s3_load_history 表中写入一个条目。

使用 aurora_s3_load_history 表验证已加载的文件

每个成功的 LOAD DATA FROM S3 语句都会使用一个条目为每个已加载的文件更新 aurora_s3_load_history 架构中的 mysql 表。

运行 LOAD DATA FROM S3 语句后，您可以通过查询 aurora_s3_load_history 表来确认已加载的文件。要查看通过迭代一次该语句而加载的文件，请使用 WHERE 子句为该语句中使用的清单文件筛选 Amazon S3 URI 上的记录。如果您之前已使用相同的清单文件，请使用 timestamp 字段筛选结果。

```
select * from mysql.aurora_s3_load_history where load_prefix = 'S3_URI';
```

下表介绍了 aurora_s3_load_history 表中的字段。

字段	描述
load_prefix	加载语句中指定的 URI。此 URI 可以映射到以下任一项： • LOAD DATA FROM S3 FILE 语句的单个数据文件 • 映射到 LOAD DATA FROM S3 PREFIX 语句的多个数据文件的 Amazon S3 前缀 • 包含要为 LOAD DATA FROM S3 MANIFEST 语句加载的文件的名称的单个清单文件
file_name	使用 load_prefix 字段中指定的 URI 从 Amazon S3 加载到 Aurora 的文件的名称。
version_number	file_name 字段指定的已加载文件的版本号（如果 Amazon S3 存储桶具有版本号）。
bytes_loaded	已加载文件的大小 (以字节为单位)。

字段	描述
load_timestamp	LOAD DATA FROM S3 语句的完成时间戳。

示例

以下语句从与 Aurora 数据库集群位于相同区域的 Amazon S3 存储桶中加载数据。该语句读取 *amzn-s3-demo-bucket* Amazon S3 存储桶中文件 customerdata.txt 的数据（以逗号分隔），然后将数据加载到表 store-schema.customer-table 中。

```
LOAD DATA FROM S3 's3://amzn-s3-demo-bucket/customerdata.csv'
    INTO TABLE store-schema.customer-table
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n'
    (ID, FIRSTNAME, LASTNAME, ADDRESS, EMAIL, PHONE);
```

以下语句从与 Aurora 数据库集群位于不同区域的 Amazon S3 存储桶中加载数据。该语句从 us-west-2 区域上的 *amzn-s3-demo-bucket* Amazon S3 存储桶中，读取与 employee-data 对象前缀匹配的所有文件中的数据（以逗号分隔），然后将数据加载到 employees 表中。

```
LOAD DATA FROM S3 PREFIX 's3-us-west-2://amzn-s3-demo-bucket/employee_data'
    INTO TABLE employees
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n'
    (ID, FIRSTNAME, LASTNAME, EMAIL, SALARY);
```

以下语句将数据从名为 q1_sales.json 的 JSON 清单文件中指定的文件加载到 sales 表中。

```
LOAD DATA FROM S3 MANIFEST 's3-us-west-2://amzn-s3-demo-bucket1/q1_sales.json'
    INTO TABLE sales
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n'
    (MONTH, STORE, GROSS, NET);
```

LOAD XML FROM S3

您可以使用 LOAD XML FROM S3 语句从 Amazon S3 存储桶上存储的三种不同 XML 格式之一的 XML 文件中加载数据：

- 列名作为 <row> 元素的属性。属性值指定表字段的内容。

```
<row column1="value1" column2="value2" .../>
```

- 列名作为 <row> 元素的子元素。子元素的值指定表字段的内容。

```
<row>
  <column1>value1</column1>
  <column2>value2</column2>
</row>
```

- 列名位于 name 元素的 <field> 元素的 <row> 属性中。<field> 元素的值指定表字段的内容。

```
<row>
  <field name='column1'>value1</field>
  <field name='column2'>value2</field>
</row>
```

语法

```
LOAD XML FROM S3 'S3-URI'
    [REPLACE | IGNORE]
    INTO TABLE tbl_name
    [PARTITION (partition_name,...)]
    [CHARACTER SET charset_name]
    [ROWS IDENTIFIED BY '<element-name>']
    [IGNORE number {LINES | ROWS}]
    [(field_name_or_user_var,...)]
    [SET col_name = expr,...]
```

参数

LOAD XML FROM S3 语句使用以下必需和可选参数。在 MySQL 文档的 LOAD XML 语句中可以找到有关其中一些参数的更多详细信息。

FILE | PREFIX

指定从单个文件还是从与指定前缀匹配的所有文件中加载数据。默认值为 FILE。

REPLACE | IGNORE

确定在输入行与数据库表中现有行具有相同唯一键值时采取什么操作。

- 如果您希望使用输入行替换表中的现有行，则指定 REPLACE。

- 如果要放弃输入行，请指定 IGNORE。默认值为 IGNORE。

INTO TABLE

指定将输入行加载到的数据库表的名称。

PARTITION

要求将所有输入行插入到由指定分区名称列表（以逗号分隔）指定的分区中。如果输入行无法插入到指定分区之一，则语句失败并返回错误。

CHARACTER SET

指定输入文件中的数据的字符集。

ROWS IDENTIFIED BY

指定用于标识输入文件中行的元素名称。默认为 <row>。

IGNORE ***number*** LINES | ROWS

指定忽略输入文件开头的特定行数。例如，您可以使用 IGNORE 1 LINES 跳过文本文件的第一行，或者使用 IGNORE 2 ROWS 跳过输入 XML 中的前两行数据。

field_name_or_user_var, ...

指定一个或多个 XML 元素的逗号分隔列表，或者指定用户变量以按照名称指定要加载的元素。用于此目的的用户变量的名称必须与 XML 文件中的元素名称匹配，前缀为 @。您可以采用用户变量来存储对应的字段值，以便在后面重复使用。

例如，以下语句将输入文件的第一列加载到 table1 的第一列，并将 table_column2 中的 table1 列值设置为第二列输入值除以 100。

```
LOAD XML FROM S3 's3://amzn-s3-demo-bucket/data.xml'
   INTO TABLE table1
   (column1, @var1)
   SET table_column2 = @var1/100;
```

SET

指定以逗号分隔的分配操作列表，这些操作将表中各列的值设置为输入文件中未包含的值。

例如，以下语句将 table1 的前两列设置为输入文件中前两列的值，然后将 column3 中 table1 的值设置为当前时间戳。

```
LOAD XML FROM S3 's3://amzn-s3-demo-bucket/data.xml'
```

```
  INTO TABLE table1
  (column1, column2)
  SET column3 = CURRENT_TIMESTAMP;
```

您可以在 SET 分配的右侧使用子查询。如果子查询返回要分配到列的值，则您只能使用标量子查询。此外，您无法使用子查询来从所加载的表中进行选择。

将数据从 Amazon Aurora MySQL 数据库集群保存到 Amazon S3 存储桶中的文本文件

您可以使用 SELECT INTO OUTFILE S3 语句从 Amazon Aurora MySQL 数据库集群中查询数据，并将数据保存到 Amazon S3 存储桶中存储的文本文件。在 Aurora MySQL 中，文件首先存储在本地磁盘上，然后导出到 S3。导出完成后，将删除本地文件。

您可以使用 Amazon S3 管理密钥（SSE-S3）或 AWS KMS key（SSE-KMS：AWS 托管式密钥或客户管理密钥) 对 Amazon S3 桶进行加密。

LOAD DATA FROM S3 语句可以使用 SELECT INTO OUTFILE S3 语句创建的文件将数据加载到 Aurora 数据库集群中。有关更多信息，请参阅 将数据从 Amazon S3 存储桶中的文本文件加载到 Amazon Aurora MySQL 数据库集群。

> **Note**
>
> Aurora Serverless v1 数据库集群不支持此功能。Aurora Serverless v2 数据库集群支持此功能。
>
> 也可以使用 AWS Management Console、AWS CLI 或 Amazon RDS API 将数据库集群数据和数据库集群快照数据保存到 Amazon S3。有关更多信息，请参阅将数据库集群数据导出到 Amazon S3 和将数据库集群快照数据导出到 Amazon S3。

目录

- 为 Aurora MySQL 授予 Amazon S3 的访问权限
- 授予权限以在 Aurora MySQL 中保存数据
- 指定 Amazon S3 存储桶的路径
- 创建清单以列出数据文件
- SELECT INTO OUTFILE S3
  - 语法

- 参数
- 注意事项
- 示例

为 Aurora MySQL 授予 Amazon S3 的访问权限

在将数据保存到 Amazon S3 存储桶之前，您必须先为 Aurora MySQL 数据库集群授予 Amazon S3 的访问权限。

为 Amazon S3 授予 Aurora MySQL 的访问权限

1. 创建一个 AWS Identity and Access Management（IAM）策略，以提供允许 Aurora MySQL 数据库集群访问 Amazon S3 存储桶和对象的权限。有关说明，请参阅创建 IAM 策略以访问 Amazon S3 资源。

   > **Note**
   >
   > 在 Aurora MySQL 版本 3.05 及更高版本中，您可以使用 AWS KMS 客户管理密钥对对象进行加密。为此，请在您的 IAM policy 中包含 `kms:GenerateDataKey` 权限。有关更多信息，请参阅 创建 IAM 策略以访问 AWS KMS 资源。
   > 您不需要此权限即可使用 AWS 托管式密钥或 Amazon S3 管理密钥（SSE-S3）来加密对象。

2. 创建一个 IAM 角色，并将您在 创建 IAM 策略以访问 Amazon S3 资源 中创建的 IAM 策略附加到新的 IAM 角色。有关说明，请参阅创建 IAM 角色以允许 Amazon Aurora 访问AWS服务。
3. 对于 Aurora MySQL 版本 2，将 `aurora_select_into_s3_role` 或 `aws_default_s3_role` 数据库集群参数设置为新 IAM 角色的 Amazon 资源名称（ARN）。如果没有为 `aurora_select_into_s3_role` 指定 IAM 角色，则 Aurora 使用在 `aws_default_s3_role` 中指定的 IAM 角色。

   对于 Aurora MySQL 版本 3，使用 `aws_default_s3_role`。

   如果集群是 Aurora 全局数据库的一部分，则为该全局数据库中的每个 Aurora 集群设置此参数。

   有关数据库集群参数的更多信息，请参阅Amazon Aurora 数据库集群和数据库实例参数。
4. 要允许 Aurora MySQL 数据库集群中的数据库用户访问 Amazon S3，请将您在创建 IAM 角色以允许 Amazon Aurora 访问AWS服务中创建的角色与该数据库集群关联。

对于 Aurora 全局数据库，将此角色与该全局数据库中的每个 Aurora 集群关联。

有关将 IAM 角色与数据库集群关联的信息，请参阅将 IAM 角色与 Amazon Aurora MySQL 数据库集群关联。

5. 配置 Aurora MySQL 数据库集群以允许建立到 Amazon S3 的出站连接。有关说明，请参阅启用从 Amazon Aurora 到其它 AWS 服务的网络通信。

   对于 Aurora 全局数据库，为该全局数据库中的每个 Aurora 集群启用出站连接。

授予权限以在 Aurora MySQL 中保存数据

发出 SELECT INTO OUTFILE S3 语句的数据库用户必须具有特定角色或权限。在 Aurora MySQL 版本 3 中，您可以授予 AWS_SELECT_S3_ACCESS 角色。在 Aurora MySQL 版本 2 中，您可以授予 SELECT INTO S3 权限。预设情况下，将为数据库集群的管理用户授予适当的角色或权限。您可以使用以下语句之一向另一个用户授予权限。

对 Aurora MySQL 版本 3 使用以下语句：

```
GRANT AWS_SELECT_S3_ACCESS TO 'user'@'domain-or-ip-address'
```

> **Tip**
>
> 当您使用 Aurora MySQL 版本 3 中的角色方法时，还可以通过使用 SET ROLE *role_name* 或 SET ROLE ALL 语句来激活角色。如果您不熟悉 MySQL 8.0 角色系统，可以在基于角色的权限模型中了解详情。有关更多详细信息，请参阅《MySQL 参考手册》中的 Using roles。
> 这仅适用于当前的活动会话。当您重新连接时，必须再次运行 SET ROLE 语句来授予权限。有关更多信息，请参阅《MySQL 参考手册》中的 SET ROLE 语句。
> 可以使用 activate_all_roles_on_login 数据库集群参数，在用户连接到数据库实例时自动激活所有角色。设置此参数后，您通常不必显式调用 SET ROLE 语句，即可激活角色。有关更多信息，请参阅《MySQL 参考手册》中的 activate_all_roles_on_login。
> 但是，当存储过程由其他用户调用时，您必须在该存储过程的开头显式调用 SET ROLE ALL 才能激活该角色。

对 Aurora MySQL 版本 2 使用以下语句：

```
GRANT SELECT INTO S3 ON *.* TO 'user'@'domain-or-ip-address'
```

AWS_SELECT_S3_ACCESS 角色和 SELECT INTO S3 权限特定于 Amazon Aurora MySQL，而不适用于 MySQL 数据库或 RDS for MySQL 数据库实例。如果您在作为复制源的 Aurora MySQL 数据库集群和作为复制客户端的 MySQL 数据库之间设置了复制，则角色或权限的 GRANT 语句将导致复制停止并出现错误。您可以安全地跳过该错误，继续复制。要跳过 RDS for MySQL 数据库实例上的错误，请使用 mysql_rds_skip_repl_error 过程。要跳过外部 MySQL 数据库上的错误，请使用 slave_skip_errors 系统变量（Aurora MySQL 版本 2）或 replica_skip_errors 系统变量（Aurora MySQL 版本 3）。

指定 Amazon S3 存储桶的路径

用于指定在 Amazon S3 存储桶上存储数据和清单文件的路径的语法与在 LOAD DATA FROM S3 PREFIX 语句中使用的语法类似，如下所示。

```
s3-region://bucket-name/file-prefix
```

路径包括以下值：

- region（可选）– 包含用来存储数据的 Amazon S3 存储桶的AWS区域。该值为可选项。如果您没有指定 region 值，则 Aurora 将文件保存到与您的数据库集群位于相同区域的 Amazon S3 中。
- bucket-name – 要将数据保存到的 Amazon S3 存储桶的名称。支持指定虚拟文件夹路径的对象前缀。
- file-prefix – 指定要保存到 Amazon S3 的文件的 Amazon S3 对象前缀。

SELECT INTO OUTFILE S3 语句创建的数据文件使用以下路径，其中 *00000* 表示从零开始的 5 位整数。

```
s3-region://bucket-name/file-prefix.part_00000
```

例如，假设 SELECT INTO OUTFILE S3 语句指定 s3-us-west-2://bucket/prefix 作为存储数据文件的路径并创建三个数据文件。指定的 Amazon S3 存储桶包含以下数据文件。

- s3-us-west-2://bucket/prefix.part_00000
- s3-us-west-2://bucket/prefix.part_00001
- s3-us-west-2://bucket/prefix.part_00002

创建清单以列出数据文件

您可以使用 SELECT INTO OUTFILE S3 语句和 MANIFEST ON 选项，以 JSON 格式创建列出由语句创建的文本文件的清单文件。LOAD DATA FROM S3 语句可以使用清单文件将数据文件加载回 Aurora MySQL 数据库集群中。有关使用清单将数据文件从 Amazon S3 加载到 Aurora MySQL 数据库集群的更多信息，请参阅使用清单指定要加载的数据文件。

清单中包含的由 SELECT INTO OUTFILE S3 语句创建的数据文件按照它们由语句创建的顺序列出。例如，假设 SELECT INTO OUTFILE S3 语句指定 s3-us-west-2://bucket/prefix 作为存储数据文件的路径并创建三个数据文件和一个清单文件。指定的 Amazon S3 存储桶包含一个名为 s3-us-west-2://bucket/prefix.manifest 的清单文件，其中包含以下信息。

```
{
  "entries": [
    {
      "url":"s3-us-west-2://bucket/prefix.part_00000"
    },
    {
      "url":"s3-us-west-2://bucket/prefix.part_00001"
    },
    {
      "url":"s3-us-west-2://bucket/prefix.part_00002"
    }
  ]
}
```

SELECT INTO OUTFILE S3

您可以使用 SELECT INTO OUTFILE S3 语句从数据库集群中查询数据，并将数据直接保存到 Amazon S3 存储桶中存储的带分隔符的文本文件。

不支持压缩文件。从 Aurora MySQL 版本 2.09.0 开始支持加密文件。

语法

```
SELECT
    [ALL | DISTINCT | DISTINCTROW ]
        [HIGH_PRIORITY]
        [STRAIGHT_JOIN]
        [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
        [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
    select_expr [, select_expr ...]
```

```
    [FROM table_references
        [PARTITION partition_list]
    [WHERE where_condition]
    [GROUP BY {col_name | expr | position}
        [ASC | DESC], ... [WITH ROLLUP]]
    [HAVING where_condition]
    [ORDER BY {col_name | expr | position}
         [ASC | DESC], ...]
    [LIMIT {[offset,] row_count | row_count OFFSET offset}]
INTO OUTFILE S3 's3_uri'
[CHARACTER SET charset_name]
    [export_options]
    [MANIFEST {ON | OFF}]
    [OVERWRITE {ON | OFF}]
    [ENCRYPTION {ON | OFF | SSE_S3 | SSE_KMS ['cmk_id']}]

export_options:
    [FORMAT {CSV|TEXT} [HEADER]]
    [{FIELDS | COLUMNS}
        [TERMINATED BY 'string']
        [[OPTIONALLY] ENCLOSED BY 'char']
        [ESCAPED BY 'char']
    ]
    [LINES
        [STARTING BY 'string']
        [TERMINATED BY 'string']
]
```

参数

SELECT INTO OUTFILE S3 语句使用以下特定于 Aurora 的必需和可选参数。

s3-uri

为要使用的 Amazon S3 前缀指定 URI。使用指定 Amazon S3 存储桶的路径中所述的语法。

FORMAT {CSV|TEXT} [HEADER]

（可选）以 CSV 格式保存数据。

TEXT 选项是默认选项，并生成现有的 MySQL 导出格式。

CSV 选项会生成逗号分隔的数据值。CSV 格式遵循 RFC-4180 规范。如果指定可选关键字 HEADER，则输出文件包含一个标题行。标题行中的标签与 SELECT 语句中的列名称相对应。您可

以将 CSV 文件用于训练数据模型，以便与 AWS ML 服务一起使用。有关将导出的 Aurora 数据与 AWS ML 服务结合使用的更多信息，请参阅将数据导出到 Amazon S3 以进行 SageMaker AI 模型训练（高级）。

MANIFEST {ON | OFF}

指示是否在 Amazon S3 中创建清单文件。清单文件是 JavaScript 对象表示法 (JSON) 文件，可用于通过 LOAD DATA FROM S3 MANIFEST 语句将数据加载到 Aurora 数据库集群中。有关 LOAD DATA FROM S3 MANIFEST 的更多信息，请参阅将数据从 Amazon S3 存储桶中的文本文件加载到 Amazon Aurora MySQL 数据库集群。

如果在查询中指定了 MANIFEST ON，则在创建并上传所有数据文件后在 Amazon S3 中创建清单文件。使用以下路径创建清单文件：

```
s3-region://bucket-name/file-prefix.manifest
```

有关清单文件的内容格式的更多信息，请参阅 创建清单以列出数据文件。

OVERWRITE {ON | OFF}

指示是否覆盖指定的 Amazon S3 桶中的现有文件。如果指定 OVERWRITE ON，则覆盖与在 s3-uri 中指定的 URI 内的文件前缀匹配的现有文件。否则将出错。

ENCRYPTION {ON | OFF | SSE_S3 | SSE_KMS ['*cmk_id*']}

指示是使用 Amazon S3 管理密钥（SSE-S3）还是使用 AWS KMS keys（SSE-KMS，包括 AWS 托管式密钥和客户管理密钥）进行服务器端加密。SSE_S3 和 SSE_KMS 设置适用于 Aurora MySQL 版本 3.05 及更高版本。

您也可以使用 aurora_select_into_s3_encryption_default 会话变量代替 ENCRYPTION 子句，如以下示例所示。使用 SQL 子句或会话变量，但不能同时使用这两者。

```
set session set session aurora_select_into_s3_encryption_default={ON | OFF | SSE_S3
 | SSE_KMS};
```

SSE_S3 和 SSE_KMS 设置适用于 Aurora MySQL 版本 3.05 及更高版本。

将 aurora_select_into_s3_encryption_default 设置为以下值时：

- OFF – 遵循 S3 桶的默认加密策略。aurora_select_into_s3_encryption_default 的默认值为 OFF。
- ON 或 SSE_S3 – 使用 Amazon S3 管理密钥（SSE-S3）对 S3 对象进行加密。
- SSE_KMS – 则使用 AWS KMS key加密 S3 对象。

在这种情况下，您还需要包含会话变量 aurora_s3_default_cmk_id，例如：

```
set session aurora_select_into_s3_encryption_default={SSE_KMS};
set session aurora_s3_default_cmk_id={NULL | 'cmk_id'};
```

- 如果aurora_s3_default_cmk_id是NULL，则使用加密 S3 对象AWS 托管式密钥。
- 如果 aurora_s3_default_cmk_id 为非空字符串 cmk_id，则使用客户管理密钥加密 S3 对象。

  cmk_id 值不能为空字符串。

当您使用 SELECT INTO OUTFILE S3 命令时，Aurora 按如下方式决定加密：

- 如果 ENCRYPTION 子句存在于 SQL 命令中，则 Aurora 仅依赖 ENCRYPTION 的值，而不使用会话变量。
- 如果 ENCRYPTION 子句不存在，则 Aurora 依赖于会话变量的值。

有关更多信息，请参阅《Amazon Simple Storage Service 用户指南》中的使用 Amazon S3 管理密钥（SSE-S3）进行服务器端加密和使用 AWS KMS 密钥（SSE-KMS）进行服务器端加密。

在 MySQL 文档的 SELECT 语句和 LOAD DATA 语句中，可以找到有关其他参数的更多详细信息。

注意事项

写入到 Amazon S3 存储桶的文件数取决于 SELECT INTO OUTFILE S3 语句选择的数据数量以及 Aurora MySQL 的文件大小阈值。默认文件大小阈值为 6GB。如果语句选择的数据小于文件大小阈值，则将创建单个文件；否则将创建多个文件。该语句所创建文件的其他注意事项包括下列内容：

- Aurora MySQL 确保数据文件中的行不会跨文件边界拆分。对于多个文件，除了最后一个文件之外，每个数据文件的大小通常接近文件大小阈值。不过，有时保持在文件大小阈值之下会导致在两个数据文件之间拆分某一行。在这种情况下，Aurora MySQL 创建一个保留该行而不拆分的数据文件，但可能会大于文件大小阈值。
- 由于 Aurora MySQL 中的每个 SELECT 语句作为原子事务运行，选择大数据集的 SELECT INTO OUTFILE S3 语句可能需要一段时间来运行。如果语句由于任何原因失败，您可能需要重新启动和发出该语句。不过，如果语句失败，已上传到 Amazon S3 的文件将保留在指定的 Amazon S3 存储桶中。您可以使用其他语句来上传剩余的数据而不用重新启动。
- 如果要选择的数据量很大（超过 25 GB），我们建议您使用多个 SELECT INTO OUTFILE S3 语句以将数据保存到 Amazon S3 中。每个语句应选择不同部分的数据来保存，并且在保存数据文件时指

定使用 file_prefix 参数中的不同 s3-uri。使用多个语句对要选择的数据进行分区可以更轻松地从一个语句中的错误中恢复。如果一条语句发生错误，则只需要重新选择部分数据并将其上传到 Amazon S3 即可。使用多个语句还有助于避免单个长时间运行的事务，这可改进性能。

- 如果多个 SELECT INTO OUTFILE S3 语句在 file_prefix 参数中使用相同的 s3-uri，并且并行运行这些语句以选择数据并上传到 Amazon S3 中，则行为是不确定的。
- Aurora MySQL 不会将元数据（例如，表架构或文件元数据）上传到 Amazon S3 中。
- 有时，您可以重新运行 SELECT INTO OUTFILE S3 查询，以便从故障中恢复。在这些情况下，您必须在 Amazon S3 存储桶中删除具有在 s3-uri 中指定的文件前缀的任何现有数据文件，或者在 OVERWRITE ON 查询中包括 SELECT INTO OUTFILE S3。

SELECT INTO OUTFILE S3 语句返回一个典型的 MySQL 错误编号以及有关成功或失败的响应。如果您无权访问 MySQL 错误编号和响应，则确定何时才完成的最简单方法是在语句中指定 MANIFEST ON。清单文件是该语句写入的最后一个文件。换而言之，如果您具有清单文件，则语句已完成。

目前，没有其他方法可以在运行期间直接监视 SELECT INTO OUTFILE S3 语句的进度。不过，假设您使用该语句将大量数据从 Aurora MySQL 写入到 Amazon S3 中，并且您知道语句选择的数据大小。在这种情况下，您可以通过监视 Amazon S3 中数据文件的创建操作来估算进度。

为此，您可以利用这样一个常识，也就是每次当语句选择大约 6GB 数据时，将在指定 Amazon S3 存储桶中创建一个数据文件。将选定数据的大小除以 6GB 可以估计要创建的数据文件数。您可以通过监视在语句运行期间上传到 Amazon S3 的文件数来估算语句的进度。

示例

以下语句选择 employees 表中的所有数据并将数据保存到与 Aurora MySQL 数据库集群位于不同区域的 Amazon S3 存储桶中。该语句创建数据文件，其中每个字段以逗号 (,) 字符终止，每行以换行 (\n) 字符终止。如果在指定的 Amazon S3 存储桶中存在与 sample_employee_data 文件前缀匹配的文件，该语句将返回错误。

```
SELECT * FROM employees INTO OUTFILE S3 's3-us-west-2://aurora-select-into-s3-pdx/
sample_employee_data'
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n';
```

以下语句选择 employees 表中的所有数据，并将数据保存到与 Aurora MySQL 数据库集群位于相同区域的 Amazon S3 存储桶中。该语句创建数据文件，其中每个字段以逗号 (,) 字符终止，每行以换行 (\n) 字符终止，还创建一个清单文件。如果在指定的 Amazon S3 存储桶中存在与 sample_employee_data 文件前缀匹配的文件，该语句将返回错误。

```
SELECT * FROM employees INTO OUTFILE S3 's3://aurora-select-into-s3-pdx/
sample_employee_data'
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n'
    MANIFEST ON;
```

以下语句选择 employees 表中的所有数据并将数据保存到与 Aurora 数据库集群位于不同区域的 Amazon S3 存储桶中。该语句创建数据文件，其中每个字段以逗号 (,) 字符终止，每行以换行 (\n) 字符终止。该语句在指定的 Amazon S3 存储桶中覆盖与 sample_employee_data 文件前缀匹配的任何现有文件。

```
SELECT * FROM employees INTO OUTFILE S3 's3-us-west-2://aurora-select-into-s3-pdx/
sample_employee_data'
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n'
    OVERWRITE ON;
```

以下语句选择 employees 表中的所有数据，并将数据保存到与 Aurora MySQL 数据库集群位于相同区域的 Amazon S3 存储桶中。该语句创建数据文件，其中每个字段以逗号 (,) 字符终止，每行以换行 (\n) 字符终止，还创建一个清单文件。该语句在指定的 Amazon S3 存储桶中覆盖与 sample_employee_data 文件前缀匹配的任何现有文件。

```
SELECT * FROM employees INTO OUTFILE S3 's3://aurora-select-into-s3-pdx/
sample_employee_data'
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n'
    MANIFEST ON
    OVERWRITE ON;
```

从 Amazon Aurora MySQL 数据库集群中调用 Lambda 函数

您可以使用本机函数 lambda_sync 或 lambda_async 从与 Amazon Aurora MySQL 兼容的数据库集群中调用 AWS Lambda 函数。从 Aurora MySQL 中调用 Lambda 函数之前，Aurora 数据库集群必须具有 Lambda 的访问权限。有关授予对 Aurora MySQL 的访问权限的详细信息，请参阅 为 Aurora 授予 Lambda 的访问权限。有关 lambda_sync 和 lambda_async 存储函数的信息，请参阅 使用 Aurora MySQL 本机函数调用 Lambda 函数。

您还可以通过使用存储过程调用 AWS Lambda 函数。然而，使用存储过程已弃用。如果您使用以下 Aurora MySQL 版本之一，强烈建议使用 Aurora MySQL 本机函数：

- Aurora MySQL 版本 2，适用于与 MySQL 5.7 兼容的集群。
- Aurora MySQL 版本 3.01 及更高版本，针对与 MySQL 8.0 兼容的集群。此存储过程在 Aurora MySQL 版本 3 中不可用。

有关向 Aurora 授予对 Lambda 的访问权限和调用 Lambda 函数的信息，请参阅以下主题。

主题

- 为 Aurora 授予 Lambda 的访问权限
- 使用 Aurora MySQL 本机函数调用 Lambda 函数
- 使用 Aurora MySQL 存储过程调用 Lambda 函数（已弃用）

为 Aurora 授予 Lambda 的访问权限

从 Aurora MySQL 数据库集群中调用 Lambda 函数之前，请确保先为您的集群授予访问 Lambda 的权限。

为 Lambda 授予 Aurora MySQL 的访问权限

1. 创建一个 AWS Identity and Access Management（IAM）策略，以提供允许 Aurora MySQL 数据库集群调用 Lambda 函数的权限。有关说明，请参阅创建 IAM 策略以访问 AWS Lambda 资源。
2. 创建一个 IAM 角色，并将您在 创建 IAM 策略以访问 AWS Lambda 资源 中创建的 IAM 策略附加到新的 IAM 角色。有关说明，请参阅创建 IAM 角色以允许 Amazon Aurora 访问AWS服务。
3. 将 `aws_default_lambda_role` 数据库集群参数设置为新 IAM 角色的 Amazon 资源名称 (ARN)。

   如果集群是 Aurora 全局数据库的一部分，则为该全局数据库中的每个 Aurora 集群应用相同的设置。

   有关数据库集群参数的更多信息，请参阅Amazon Aurora 数据库集群和数据库实例参数。
4. 要允许 Aurora MySQL 数据库集群中的数据库用户调用 Lambda 函数，请将您在创建 IAM 角色以允许 Amazon Aurora 访问AWS服务中创建的角色与该数据库集群关联。有关将 IAM 角色与数据库集群关联的信息，请参阅将 IAM 角色与 Amazon Aurora MySQL 数据库集群关联。

   如果集群是 Aurora 全局数据库的一部分，则将此角色与该全局数据库中的每个 Aurora 集群关联。

5. 配置 Aurora MySQL 数据库集群以允许建立到 Lambda 的出站连接。有关说明，请参阅启用从 Amazon Aurora 到其它 AWS 服务的网络通信。

   如果集群是 Aurora 全局数据库的一部分，则为该全局数据库中的每个 Aurora 集群启用出站连接。

使用 Aurora MySQL 本机函数调用 Lambda 函数

> **Note**
>
> 当您使用 Aurora MySQL 2 或 Aurora MySQL 版本 3.01 及更高版本时，您可以调用 `lambda_sync` 和 `lambda_async` 原生函数。有关 Aurora MySQL 版本的更多信息，请参阅Amazon Aurora MySQL 的数据库引擎更新。

您可以调用本机函数 `lambda_sync` 和 `lambda_async`，以从 Aurora MySQL 数据库集群中调用 AWS Lambda 函数。如果要将在 Aurora MySQL 上运行的数据库与其他AWS服务集成，这种方法可能是非常有用的。例如，每次在数据库的特定表中插入行时，您可能希望使用 Amazon Simple Notification Service (Amazon SNS) 发送通知。

目录

- 使用本机函数调用 Lambda 函数
  - 在 Aurora MySQL 版本 3 中给角色授权
  - 在 Aurora MySQL 版本 2 中授予权限
  - lambda_sync 函数的语法
  - lambda_sync 函数的参数
  - lambda_sync 函数示例
  - lambda_async 函数的语法
  - lambda_async 函数的参数
  - lambda_async 函数示例
  - 在触发器中调用 Lambda 函数

使用本机函数调用 Lambda 函数

lambda_sync 和 lambda_async 函数是内置的本机函数，它们同步或异步地调用 Lambda 函数。如果您必须知道 Lambda 函数的结果才能执行其他操作，请使用同步函数 lambda_sync。如果您不需要知道 Lambda 函数的结果即可执行其他操作，请使用异步函数 lambda_async。

在 Aurora MySQL 版本 3 中给角色授权

在 Aurora MySQL 版本 3 中，必须为调用本机函数的用户授予 AWS_LAMBDA_ACCESS 角色。要为用户授予此角色，请以管理用户身份连接到数据库实例，然后运行以下语句。

```
GRANT AWS_LAMBDA_ACCESS TO user@domain-or-ip-address
```

您可以运行以下语句以吊销该角色。

```
REVOKE AWS_LAMBDA_ACCESS FROM user@domain-or-ip-address
```

> **Tip**
>
> 当您使用 Aurora MySQL 版本 3 中的角色方法时，还可以通过使用 SET ROLE *role_name* 或 SET ROLE ALL 语句来激活角色。如果您不熟悉 MySQL 8.0 角色系统，可以在基于角色的权限模型中了解详情。有关更多详细信息，请参阅《MySQL 参考手册》中的 Using roles。这仅适用于当前的活动会话。当您重新连接时，必须再次运行 SET ROLE 语句来授予权限。有关更多信息，请参阅《MySQL 参考手册》中的 SET ROLE 语句。
> 可以使用 activate_all_roles_on_login 数据库集群参数，在用户连接到数据库实例时自动激活所有角色。设置此参数后，您通常不必显式调用 SET ROLE 语句，即可激活角色。有关更多信息，请参阅《MySQL 参考手册》中的 activate_all_roles_on_login。
> 但是，当存储过程由其他用户调用时，您必须在该存储过程的开头显式调用 SET ROLE ALL 才能激活该角色。

如果您在尝试调用 Lambda 函数时出现如下错误，请运行 SET ROLE 语句。

```
SQL Error [1227] [42000]: Access denied; you need (at least one of) the Invoke Lambda
 privilege(s) for this operation
```

确保将角色授予正确的用户，如 mysql.users 表格条目中所示。可能有多个同名但位于不同主机上的用户。根据调用 lambda_sync 函数的应用程序或主机，MySQL 会根据 host 列条目来选择最匹配的用户。

在 Aurora MySQL 版本 2 中授予权限

在 Aurora MySQL 版本 2 中，必须为调用原生函数的用户授予 INVOKE LAMBDA 权限。要为用户授予此权限，请以管理用户身份连接到数据库实例，然后运行以下语句。

```
GRANT INVOKE LAMBDA ON *.* TO user@domain-or-ip-address
```

您可以运行以下语句以吊销该权限。

```
REVOKE INVOKE LAMBDA ON *.* FROM user@domain-or-ip-address
```

lambda_sync 函数的语法

您可以使用 lambda_sync 调用类型同步地调用 RequestResponse 函数。该函数在 JSON 负载中返回 Lambda 调用的结果。该函数使用以下语法。

```
lambda_sync (
  lambda_function_ARN,
  JSON_payload
)
```

lambda_sync 函数的参数

lambda_sync 函数具有以下参数。

lambda_function_ARN

要调用的 Lambda 函数的 Amazon Resource Name (ARN)。

JSON_payload

调用的 Lambda 函数的负载，采用 JSON 格式。

> **Note**
>
> Aurora MySQL 版本 3 支持 MySQL 8.0 中的 JSON 解析函数。但是，Aurora MySQL 版本 2 不包含这些函数。在 Lambda 函数返回原子值 (如数字或字符串) 时，不需要进行 JSON 解析。

lambda_sync 函数示例

基于 lambda_sync 的以下查询使用函数 ARN 同步地调用 Lambda 函数 BasicTestLambda。该函数的负载是 {"operation": "ping"}。

```
SELECT lambda_sync(
    'arn:aws:lambda:us-east-1:123456789012:function:BasicTestLambda',
    '{"operation": "ping"}');
```

lambda_async 函数的语法

您可以使用 lambda_async 调用类型异步地调用 Event 函数。该函数在 JSON 负载中返回 Lambda 调用的结果。该函数使用以下语法。

```
lambda_async (
  lambda_function_ARN,
  JSON_payload
)
```

lambda_async 函数的参数

lambda_async 函数具有以下参数。

lambda_function_ARN

要调用的 Lambda 函数的 Amazon Resource Name (ARN)。

JSON_payload

调用的 Lambda 函数的负载，采用 JSON 格式。

> **Note**
>
> Aurora MySQL 版本 3 支持 MySQL 8.0 中的 JSON 解析函数。但是，Aurora MySQL 版本 2 不包含这些函数。在 Lambda 函数返回原子值 (如数字或字符串) 时，不需要进行 JSON 解析。

lambda_async 函数示例

基于 lambda_async 的以下查询使用函数 ARN 异步地调用 Lambda 函数 BasicTestLambda。该函数的负载是 {"operation": "ping"}。

```
SELECT lambda_async(
    'arn:aws:lambda:us-east-1:123456789012:function:BasicTestLambda',
    '{"operation": "ping"}');
```

在触发器中调用 Lambda 函数

您可以使用触发器对数据修改语句调用 Lambda。以下示例使用 `lambda_async` 原生函数并将结果存储在变量中。

```
mysql>SET @result=0;
mysql>DELIMITER //
mysql>CREATE TRIGGER myFirstTrigger
      AFTER INSERT
          ON Test_trigger FOR EACH ROW
      BEGIN
      SELECT lambda_async(
          'arn:aws:lambda:us-east-1:123456789012:function:BasicTestLambda',
          '{"operation": "ping"}')
          INTO @result;
      END; //
mysql>DELIMITER ;
```

> **Note**
>
> 触发器不是针对每个 SQL 语句运行一次，而是针对每个修改后的行运行一次（一次一行）。
> 当触发器运行时，进程是同步的。数据修改语句仅在触发器完成时返回。
> 对于高写入流量的表，请谨慎使用其中的触发器来调用 AWS Lambda 函数。`INSERT`、`UPDATE` 和 `DELETE` 触发器按行激活。在包含 `INSERT`、`UPDATE` 或 `DELETE` 触发器的表中，如果出现密集型写入工作负载，会导致大量调用 AWS Lambda 函数。

使用 Aurora MySQL 存储过程调用 Lambda 函数（已弃用）

您可以调用 `mysql.lambda_async` 过程以从 Aurora MySQL 数据库集群中调用 AWS Lambda 函数。如果要将在 Aurora MySQL 上运行的数据库与其他AWS服务集成，这种方法可能是非常有用的。例如，每次在数据库的特定表中插入行时，您可能希望使用 Amazon Simple Notification Service (Amazon SNS) 发送通知。

目录

- Aurora MySQL 版本注意事项
- 结合使用 mysql.lambda_async 过程来调用 Lambda 函数（已弃用）
  - 语法
  - 参数
  - 示例

Aurora MySQL 版本注意事项

从 Aurora MySQL 版本 2 开始，您可以使用原生函数方法调用 Lambda 函数，而不是使用这些存储过程。有关本机函数的更多信息，请参阅 使用本机函数调用 Lambda 函数。

在 Aurora MySQL 版本 2 中，不再支持存储过程 `mysql.lambda_async`。强烈建议您改用原生 Lambda 函数。

存储过程在 Aurora MySQL 版本 3 中不可用。

结合使用 mysql.lambda_async 过程来调用 Lambda 函数（已弃用）

`mysql.lambda_async` 过程是一个异步调用 Lambda 函数的内置存储过程。要使用该过程，您的数据库用户必须对 `EXECUTE` 存储过程拥有 `mysql.lambda_async` 权限。

语法

`mysql.lambda_async` 过程使用以下语法。

```
CALL mysql.lambda_async (
  lambda_function_ARN,
  lambda_function_input
)
```

参数

`mysql.lambda_async` 过程具有以下参数。

lambda_function_ARN

要调用的 Lambda 函数的 Amazon Resource Name (ARN)。

lambda_function_input

所调用 Lambda 函数的 JSON 格式的输入字符串。

示例

作为最佳实践，我们建议您将对 mysql.lambda_async 的调用包装在存储过程中，该存储过程可从不同的来源 (例如触发器或客户端代码) 调用。这有助于避免出现阻抗不一致问题，并且可更轻松地调用 Lambda 函数。

> **Note**
>
> 对于高写入流量的表，请谨慎使用其中的触发器来调用 AWS Lambda 函数。INSERT、UPDATE 和 DELETE 触发器按行激活。在包含 INSERT、UPDATE 或 DELETE 触发器的表中，如果出现密集型写入工作负载，会导致大量调用 AWS Lambda 函数。尽管对 mysql.lambda_async 过程的调用是异步操作，但触发器是同步的。产生大量触发器激活的语句不等待调用 AWS Lambda 函数完成，但是在返回对客户端的控制之前，它却需要等待触发器完成。

Example 示例：调用 AWS Lambda 函数来发送电子邮件

以下示例创建一个存储过程，您可以在您的数据库代码中调用该过程来使用 Lambda 函数发送电子邮件。

AWS Lambda 函数

```
import boto3

ses = boto3.client('ses')

def SES_send_email(event, context):

    return ses.send_email(
        Source=event['email_from'],
        Destination={
            'ToAddresses': [
            event['email_to'],
            ]
        },

        Message={
            'Subject': {
            'Data': event['email_subject']
            },
```

```
            'Body': {
                'Text': {
                    'Data': event['email_body']
                }
            }
        }
    )
```

存储过程

```
DROP PROCEDURE IF EXISTS SES_send_email;
DELIMITER ;;
  CREATE PROCEDURE SES_send_email(IN email_from VARCHAR(255),
                                  IN email_to VARCHAR(255),
                                  IN subject VARCHAR(255),
                                  IN body TEXT) LANGUAGE SQL
  BEGIN
    CALL mysql.lambda_async(
         'arn:aws:lambda:us-west-2:123456789012:function:SES_send_email',
         CONCAT('{"email_to" : "', email_to,
             '", "email_from" : "', email_from,
             '", "email_subject" : "', subject,
             '", "email_body" : "', body, '"}')
     );
  END
  ;;
DELIMITER ;
```

调用存储过程以调用 AWS Lambda 函数

```
mysql> call SES_send_email('example_from@amazon.com', 'example_to@amazon.com', 'Email
 subject', 'Email content');
```

Example 示例：调用 AWS Lambda 函数来从触发器发布事件

以下示例创建一个存储过程，该过程使用 Amazon SNS 发布事件。向表中添加行时，该代码会从触发器调用过程。

AWS Lambda 函数

```
import boto3
```

```
sns = boto3.client('sns')

def SNS_publish_message(event, context):

    return sns.publish(
        TopicArn='arn:aws:sns:us-west-2:123456789012:Sample_Topic',
        Message=event['message'],
        Subject=event['subject'],
        MessageStructure='string'
    )
```

存储过程

```
DROP PROCEDURE IF EXISTS SNS_Publish_Message;
DELIMITER ;;
CREATE PROCEDURE SNS_Publish_Message (IN subject VARCHAR(255),
                                      IN message TEXT) LANGUAGE SQL
BEGIN
  CALL mysql.lambda_async('arn:aws:lambda:us-
west-2:123456789012:function:SNS_publish_message',
     CONCAT('{ "subject" : "', subject,
            '", "message" : "', message, '" }')
     );
END
;;
DELIMITER ;
```

表

```
CREATE TABLE 'Customer_Feedback' (
  'id' int(11) NOT NULL AUTO_INCREMENT,
  'customer_name' varchar(255) NOT NULL,
  'customer_feedback' varchar(1024) NOT NULL,
  PRIMARY KEY ('id')
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

触发器

```
DELIMITER ;;
CREATE TRIGGER TR_Customer_Feedback_AI
  AFTER INSERT ON Customer_Feedback
  FOR EACH ROW
```

```
BEGIN
  SELECT CONCAT('New customer feedback from ', NEW.customer_name),
 NEW.customer_feedback INTO @subject, @feedback;
  CALL SNS_Publish_Message(@subject, @feedback);
END
;;
DELIMITER ;
```

将行插入表中触发通知

```
mysql> insert into Customer_Feedback (customer_name, customer_feedback) VALUES ('Sample
 Customer', 'Good job guys!');
```

将 Amazon Aurora MySQL 日志发布到 Amazon CloudWatch Logs

您可以配置 Aurora MySQL 数据库集群以将一般、慢速、审核和错误日志数据发布到 Amazon CloudWatch Logs 中的日志组。利用 CloudWatch Logs，可以对日志数据进行实时分析并使用 CloudWatch 创建警报和查看指标。您可以使用 CloudWatch Logs 在高持久性存储中存储日志记录。

要将日志发布到 CloudWatch Logs，必须启用相应日志。默认情况下启用错误日志，但您必须明确启用其他类型的日志。有关在 MySQL 中启用日志的信息，请参阅 MySQL 文档中的选择一般查询和慢速查询日志输出目标。有关启用 Aurora MySQL 审核日志的更多信息，请参阅 启用高级审核。

> **Note**
>
> - 如果禁用了导出日志数据，则 Aurora 不会删除现有日志组或日志流。如果禁用了导出日志数据，现有日志数据在 CloudWatch Logs 中保持可用 (具体取决于日志保留)，并且您仍将产生存储审核日志数据的费用。您可以使用 CloudWatch Logs 控制台、AWS CLI 或 CloudWatch Logs API 删除日志流和日志组。
> - 将审计日志发布到 CloudWatch Logs 的一种替代方法是启用高级审计，然后创建一个自定义数据库集群参数组并将 `server_audit_logs_upload` 参数设置为 `1`。`server_audit_logs_upload` 数据库集群参数的默认值为 `0`。有关启用高级审计的信息，请参阅在 Amazon Aurora MySQL 数据库集群中使用高级审计。
>
>   如果使用该替代方法，您必须具有 IAM 角色以访问 CloudWatch Logs，并将 `aws_default_logs_role` 集群级参数设置为该角色的 ARN。有关创建 角色的信息，请参阅设置 IAM 角色以访问AWS服务。不过，如果您具有 `AWSServiceRoleForRDS` 服务相关角色，它提供 CloudWatch Logs 的访问权限并覆盖所有自定义角色。有关 Amazon RDS 的服务相关角色的信息，请参阅 将服务相关角色用于 Amazon Aurora。

- 如果您不希望将审核日志导出到 CloudWatch Logs，请确保导出审核日志的所有方法都已禁用。这些方法包括 AWS Management Console、AWS CLI、RDS API 和 `server_audit_logs_upload` 参数。
- 对于 Aurora Serverless v1 数据库集群而言，该过程与包含预调配或 Aurora Serverless v2 数据库实例的数据库集群略有不同。Aurora Serverless v1 集群会自动上传您通过配置参数启用的所有日志。

  因此，您将通过在数据库集群参数组中打开和关闭不同的日志类型，来启用或禁用 Aurora Serverless v1 数据库集群的日志上传。您不能通过 AWS Management Console、AWS CLI 或 RDS API 修改集群本身的设置。有关打开和关闭 Aurora Serverless v1 集群的 MySQL 日志的信息，请参阅 Aurora Serverless v1 的参数组。

控制台

您可以使用控制台将预配置集群的 Aurora MySQL 日志发布到 CloudWatch Logs。

从控制台发布 Aurora MySQL 日志

1. 通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Databases (数据库)。
3. 选择要为其发布日志数据的 Aurora MySQL 数据库集群。
4. 选择 Modify (修改)。
5. 在 Log exports (日志导出) 部分中，选择要开始发布到 CloudWatch Logs 的日志。
6. 选择继续，然后选择摘要页面上的 修改数据库集群。

AWS CLI

您可以使用 AWS CLI 发布预置集群的 Aurora MySQL 日志。要这样做，您可以使用以下选项运行 modify-db-cluster AWS CLI 命令：

- `--db-cluster-identifier`—数据库集群标识符。
- `--cloudwatch-logs-export-configuration` — 为数据库集群导出到 CloudWatch Logs 而启用的日志类型的配置设置。

您还可以运行以下 AWS CLI 命令之一以发布 Aurora MySQL 日志：

- create-db-cluster
- restore-db-cluster-from-s3
- restore-db-cluster-from-snapshot
- restore-db-cluster-to-point-in-time

通过以下选项运行上述 AWS CLI 命令之一：

- `--db-cluster-identifier`—数据库集群标识符。
- `--engine` — 数据库引擎。
- `--enable-cloudwatch-logs-exports` — 为数据库集群导出到 CloudWatch Logs 而启用的日志类型的配置设置。

根据您运行的 AWS CLI 命令，可能需要其他选项。

Example

以下命令修改现有的 Aurora MySQL 数据库集群以将日志文件发布到 CloudWatch Logs。

对于 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --cloudwatch-logs-export-configuration '{"EnableLogTypes":
["error","general","slowquery","audit","instance"]}'
```

对于 Windows：

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --cloudwatch-logs-export-configuration '{"EnableLogTypes":
["error","general","slowquery","audit","instance"]}'
```

Example

以下命令创建一个 Aurora MySQL 数据库集群以将日志文件发布到 CloudWatch Logs。

对于 Linux、macOS 或 Unix：

```
aws rds create-db-cluster \
    --db-cluster-identifier mydbcluster \
    --engine aurora \
    --enable-cloudwatch-logs-exports
 '["error","general","slowquery","audit","instance"]'
```

对于 Windows：

```
aws rds create-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --engine aurora ^
    --enable-cloudwatch-logs-exports
 '["error","general","slowquery","audit","instance"]'
```

RDS API

您可以使用 RDS API 发布预配置集群的 Aurora MySQL 日志。要这样做，您可以使用以下选项运行 ModifyDBCluster 操作：

- `DBClusterIdentifier`—数据库集群标识符。
- `CloudwatchLogsExportConfiguration` — 为数据库集群导出到 CloudWatch Logs 而启用的日志类型的配置设置。

您还可以通过运行以下 RDS API 操作之一，使用 RDS API 发布 Aurora MySQL 日志：

- CreateDBCluster
- RestoreDBClusterFromS3
- RestoreDBClusterFromSnapshot
- RestoreDBClusterToPointInTime

运行带以下参数的 RDS API 操作：

- `DBClusterIdentifier`—数据库集群标识符。
- `Engine` — 数据库引擎。
- `EnableCloudwatchLogsExports` — 为数据库集群导出到 CloudWatch Logs 而启用的日志类型的配置设置。

根据您运行的 AWS CLI 命令，可能需要其他参数。

在 Amazon CloudWatch 中监控日志事件

在启用 Aurora MySQL 日志事件后，您可以在 Amazon CloudWatch Logs 中监控事件。将自动使用以下前缀为 Aurora 数据库集群创建新的日志组，其中 *cluster-name* 表示数据库集群名称，*log_type* 表示日志类型。

```
/aws/rds/cluster/cluster-name/log_type
```

例如，如果您将导出功能配置为包括名为 mydbcluster 的数据库集群的慢速查询日志，则慢速查询数据将存储在 /aws/rds/cluster/mydbcluster/slowquery 日志组中。

将使用不同的日志流将来自数据库集群中的所有数据库实例的事件推送到一个日志组。该行为取决于以下哪些条件为真：

- 存在包含指定名称的日志组。

  Aurora 使用现有日志组为集群导出日志数据。您可以使用自动化配置（例如 AWS CloudFormation）创建具有预定义日志保留期、指标筛选条件和客户访问权限的日志组。
- 具有指定名称的日志组不存在。

  当在实例的日志文件中检测到匹配的日志条目时，Aurora MySQL 会自动在 CloudWatch Logs 中创建一个新的日志组。日志组使用永不过期的默认日志保留期。

  要更改日志保留期，请使用 CloudWatch Logs 控制台、AWS CLI 或 CloudWatch Logs API。有关在 CloudWatch Logs 中更改日志保留期的更多信息，请参阅在 CloudWatch Logs 中更改日志数据保留期。

要在数据库集群的日志事件中搜索信息，请使用 CloudWatch Logs 控制台、AWS CLI 或 CloudWatch Logs API。有关搜索和筛选日志数据的更多信息，请参阅搜索和筛选日志数据。

Amazon Aurora MySQL 实验室模式

> **Important**
>
> Aurora MySQL 版本 2 引入了实验室模式来实现快速 DDL 优化，这提高了某些 DDL 操作的效率。在 Aurora MySQL 版本 3 中，实验室模式已被移除，快速 DDL 被名为 Instant DDL 的 MySQL 8.0 功能所取代。

Aurora 实验室模式用于启用在当前 Aurora 数据库版本中提供但没有默认启用的 Aurora 功能。虽然不建议在生产数据库集群中使用 Aurora 实验室模式功能，但您可以使用 Aurora 实验室模式在开发和测试环境中为数据库集群启用这些功能。有关在启用 Aurora 实验室模式时提供的 Aurora 功能的更多信息，请参阅Aurora 实验室模式功能。

`aurora_lab_mode` 参数是默认参数组中的实例级参数。在默认参数组中，该参数设置为 `0`（已禁用）。要启用 Aurora 实验室模式，请创建一个自定义参数组，在该自定义参数组中将 `aurora_lab_mode` 参数设置为 `1`（已启用），然后修改 Aurora 集群中的一个或多个数据库实例以使用该自定义参数组。然后连接到相应的实例终端节点以尝试实验室模式功能。有关修改数据库参数组的信息，请参阅在 Amazon Aurora 中修改数据库参数组中的参数。有关参数组和 Amazon Aurora 的信息，请参阅Aurora MySQL 配置参数。

Aurora 实验室模式功能

启用 Aurora 实验室模式时，以下 Aurora 功能当前可用。您必须先启用 Aurora 实验室模式，然后才能使用所有这些功能。

快速 DDL

此功能允许您几乎即时运行 `ALTER TABLE` *`tbl_name`* `ADD COLUMN` *`col_name column_definition`* 操作。完成该操作无需复制表，对其他 DML 语句也没有重大影响。由于该操作不会在表复制中使用临时存储，因此，即使对小型实例类中的大型表，也可以使用 DDL 语句。

目前，快速 DDL 仅支持将没有默认值且可为空的列添加到表的最后。有关使用此功能的更多信息，请参阅使用快速 DDL 在 Amazon Aurora 中修改表。

Amazon Aurora MySQL 的最佳实践

本主题提供使用 Amazon Aurora MySQL 数据库集群或向其迁移数据的最佳实践和选项信息。本主题中的信息总结并重申了一些准则和过程，您可以在 管理 Amazon Aurora 数据库集群 中找到这些信息。

目录

- 确定您连接到的数据库实例
- Aurora MySQL 性能和扩展的最佳实践
  - 使用 T 实例类进行开发和测试
  - 使用异步键预取优化 Aurora MySQL 索引的联接查询
    - 启用异步键预取
    - 优化异步键预取的查询
  - 使用哈希联接优化大型 Aurora MySQL 联接查询
    - 启用哈希联接
    - 优化哈希联接的查询
  - 使用 Amazon Aurora 为 MySQL 数据库扩展读取
  - 优化时间戳操作
  - 虚拟索引 ID 溢出错误
- Aurora MySQL 高可用性的最佳实践
  - 使用 Amazon Aurora 实现 MySQL 数据库的灾难恢复
  - 从 MySQL 迁移到 Amazon Aurora MySQL 并减少停机时间
  - 避免 Aurora MySQL 数据库实例性能降低、自动重启和故障转移
- Aurora MySQL 中 MySQL 功能的建议
  - 在 Aurora MySQL 中使用多线程复制
  - 使用本机 MySQL 函数调用 AWS Lambda 函数
  - 避免将 XA 事务与 Amazon Aurora MySQL 结合使用
  - 在 DML 语句执行期间保持打开外键
  - 配置刷新日志缓冲区的频率
  - 最大限度地减少 Aurora MySQL 死锁以及排查相关问题
    - 最大限度地减少 InnoDB 死锁
    - 监控 InnoDB 死锁

- 使用 Amazon CloudWatch 指标评估 Aurora MySQL 数据库实例使用情况

确定您连接到的数据库实例

要确定连接到 Aurora MySQL 数据库集群中的哪个数据库实例，请检查 `innodb_read_only` 全局变量，如以下示例中所示。

```
SHOW GLOBAL VARIABLES LIKE 'innodb_read_only';
```

`innodb_read_only` 变量设置为 `ON` (如果您已连接到读取器数据库实例)。此设置为 `OFF` (如果您已连接到写入器数据库实例，例如预置集群中的主实例)。

如果要在应用程序代码中添加逻辑以平衡工作负载或确保写入操作使用正确的连接，该方法可能是非常有用的。

Aurora MySQL 性能和扩展的最佳实践

您可以应用以下最佳实践来改进 Aurora MySQL 集群的性能和可扩展性。

主题

- 使用 T 实例类进行开发和测试
- 使用异步键预取优化 Aurora MySQL 索引的联接查询
- 使用哈希联接优化大型 Aurora MySQL 联接查询
- 使用 Amazon Aurora 为 MySQL 数据库扩展读取
- 优化时间戳操作
- 虚拟索引 ID 溢出错误

使用 T 实例类进行开发和测试

使用 `db.t2`、`db.t3` 或 `db.t4g` 数据库实例类的 Amazon Aurora MySQL 实例最适合不支持长时间运行较高工作负载的应用程序。T 实例旨在提供适度的基准性能，并能够根据您工作负载的需要实现性能的显著突增。它们旨在用于不经常或不持续使用完整 CPU、但偶尔需要突增性能的工作负载。建议仅将 T 数据库实例类用于开发和测试服务器，或其他非生产服务器。有关 T 实例类的更多详细信息，请参阅具爆发能力的实例。

如果您的 Aurora 集群大于 40 TB，请勿使用 T 实例类。当数据库带有大量数据时，管理架构对象的内存开销可能会超过 T 实例的容量。

不要在 Amazon Aurora MySQL T 实例上启用 MySQL 性能架构。如果启用了性能架构，T 实例可能会出现内存不足的情况。

> **Tip**
>
> 如果您的数据库有时处于空闲状态，但有时又有大量工作负载，则可以使用 Aurora Serverless v2 作为 T 实例的替代。使用 Aurora Serverless v2，您可以定义容量范围，Aurora 会根据当前的工作负载自动扩缩数据库。有关使用情况的详细信息，请参阅 使用 Aurora Serverless v2。有关可与 Aurora Serverless v2 一起使用的数据库引擎版本，请参阅 Aurora Serverless v2 的要求和限制。

在 Aurora MySQL 数据库集群中使用 T 实例作为数据库实例时，建议执行以下操作：

- 对数据库集群中的所有实例使用相同的数据库实例类。例如，如果您将 `db.t2.medium` 用于写入器实例，那么我们建议您也将 `db.t2.medium` 用于读取器实例。
- 不要调整任何与内存相关的配置设置，例如 `innodb_buffer_pool_size`。Aurora 对 T 实例上的内存缓冲区使用一组高度优化的默认值。Aurora 在内存受限的实例上运行时需要采用这些特殊的默认值。如果您在 T 实例上更改任何与内存相关的设置，则更有可能遇到内存不足的情况，即使您的更改旨在增加缓冲区大小。
- 监控 CPU 积分余额 (`CPUCreditBalance`) 以确保其处于可持续的水平。也就是说，CPU 积分将在使用 CPU 时按相同的费率累积。

  如果您用完实例的 CPU 积分，则会发现可用 CPU 立即下降，并且实例的读取和写入延迟将会增加。这种情况导致实例的总体性能大大降低。

  如果您的 CPU 积分余额未处于可持续的水平，建议您修改数据库实例以使用支持的 R 数据库实例类之一 (扩展计算)。

  有关监视指标的更多信息，请参阅 在 Amazon RDS 控制台中查看指标。
- 监控写入器实例与读取器实例之间的副本滞后（`AuroraReplicaLag`）。

  如果读取器实例在写入器实例之前耗尽 CPU 积分，则产生的滞后可能会导致读取器实例频繁重新启动。如果应用程序在读取器实例之间分配较高的读取操作负载，同时写入器实例具有非常低的写入操作负载，则通常会出现这种情况。

  如果您发现副本滞后持续增加，请确保数据库集群中的写入器实例的 CPU 积分余额未被用完。

如果您的 CPU 积分余额未处于可持续的水平，我们建议您修改数据库实例以使用支持的 R 数据库实例类之一 (扩展计算)。

- 对于已启用二进制日志记录的数据库集群，将每事务的插入次数保持在 100 万以下。

  如果数据库集群的数据库集群参数组将 `binlog_format` 参数设置为 `OFF` 以外的值，并且数据库集群收到的事务包含超过 100 万个要插入的行，数据库集群可能会出现内存不足的情况。您可以监控可释放内存 (`FreeableMemory`) 指标以确定数据库集群是否用完可用的内存。然后，您可以检查写入操作 (`VolumeWriteIOPS`) 指标以确定写入器实例是否收到较高的写入操作负载。如果出现这种情况，我们建议您更新应用程序以将事务中的插入数量限制为少于 100 万个。或者，您也可以修改实例以使用支持的 R 数据库实例类之一 (扩展计算)。

使用异步键预取优化 Aurora MySQL 索引的联接查询

Aurora MySQL 可以使用异步键预取（AKP）功能来提高跨索引联接表的查询的性能。该功能通过预测运行查询所需的行 (JOIN 查询需要使用批处理键访问 (BKA) 联接算法和多区间读 (MRR) 优化功能) 来提高性能。有关 BKA 和 MRR 的更多信息，请参阅 MySQL 文档中的块嵌套循环和批处理键访问联接和多区间读优化。

要利用 AKP 功能，查询必须使用 BKA 和 MRR。通常，当查询的 JOIN 子句使用二级索引并且还需要主索引中的一些列时，会出现此类查询。例如，如果 JOIN 子句表示小型外部表和大型内部表之间的索引值的 equijoin，并且索引在大型表中具有高选择性，则可以使用 AKP。AKP 与 BKA 和 MRR 协作，在计算 JOIN 子句期间执行二级索引到主索引的查找。AKP 标识计算 JOIN 子句期间运行查询所需的行。之后，在运行查询之前，它使用后台线程异步将包含这些行的页加载到内存中。

AKP 适用于 Aurora MySQL 版本 2.10 及更高版本和版本 3。有关 Aurora MySQL 版本的更多信息，请参阅Amazon Aurora MySQL 的数据库引擎更新。

启用异步键预取

您可以将 MySQL 服务器变量 `aurora_use_key_prefetch` 设置为 `on` 以启用 AKP 功能。默认情况下，该值设置为 `on`。不过，在您也启用 BKA 联接算法并禁用基于成本的 MRR 功能之前，无法启用 AKP。为此，您必须为 MySQL 服务器变量 `optimizer_switch` 设置以下值：

- 将 `batched_key_access` 设置为 `on`。该值控制对 BKA 联接算法的使用。默认情况下，该值设置为 `off`。
- 将 `mrr_cost_based` 设置为 `off`。该值控制对基于成本的 MRR 功能的使用。默认情况下，该值设置为 `on`。

目前，您只能在会话级别设置这些值。以下示例说明了如何设置这些值以通过执行 SET 语句来为当前会话启用 AKP。

```
mysql> set @@session.aurora_use_key_prefetch=on;
mysql> set @@session.optimizer_switch='batched_key_access=on,mrr_cost_based=off';
```

同样，您可以使用 SET 语句为当前会话禁用 AKP 和 BKA 联接算法并重新启用基于成本的 MRR 功能，如以下示例中所示。

```
mysql> set @@session.aurora_use_key_prefetch=off;
mysql> set @@session.optimizer_switch='batched_key_access=off,mrr_cost_based=on';
```

有关 batched_key_access 和 mrr_cost_based 优化程序开关的更多信息，请参阅 MySQL 文档中的 可切换的优化。

优化异步键预取的查询

您可以确认查询是否能利用 AKP 功能。为此，请使用 EXPLAIN 语句来分析查询，然后再运行查询。EXPLAIN 语句提供有关用于指定查询的执行计划的信息。

在 EXPLAIN 语句的输出中，Extra 列描述执行计划附带的其他信息。如果 AKP 功能应用于查询中使用的表，则此列包含下列值之一：

- `Using Key Prefetching`
- `Using join buffer (Batched Key Access with Key Prefetching)`

以下示例说明如何使用 EXPLAIN 来查看可利用 AKP 的查询的执行计划。

```
mysql> explain select sql_no_cache
    ->     ps_partkey,
    ->     sum(ps_supplycost * ps_availqty) as value
    -> from
    ->     partsupp,
    ->     supplier,
    ->     nation
    -> where
    ->     ps_suppkey = s_suppkey
    ->     and s_nationkey = n_nationkey
```

```
    ->     and n_name = 'ETHIOPIA'
    -> group by
    ->     ps_partkey having
    ->         sum(ps_supplycost * ps_availqty) > (
    ->             select
    ->                 sum(ps_supplycost * ps_availqty) * 0.0000003333
    ->             from
    ->                 partsupp,
    ->                 supplier,
    ->                 nation
    ->             where
    ->                 ps_suppkey = s_suppkey
    ->                 and s_nationkey = n_nationkey
    ->                 and n_name = 'ETHIOPIA'
    ->         )
    -> order by
    ->     value desc;
+----+-------------+----------+------+-----------------------+---------------
+---------+-----------------------------------+------+----------
+-------------------------------------------------------------+
| id | select_type | table    | type | possible_keys         | key           | key_len
 | ref                               | rows | filtered | Extra
                            |
+----+-------------+----------+------+-----------------------+---------------
+---------+-----------------------------------+------+----------
+-------------------------------------------------------------+
|  1 | PRIMARY     | nation   | ALL  | PRIMARY               | NULL          | NULL
 | NULL                              |   25 |   100.00 | Using where; Using temporary;
 Using filesort                |
|  1 | PRIMARY     | supplier | ref  | PRIMARY,i_s_nationkey | i_s_nationkey | 5
 | dbt3_scale_10.nation.n_nationkey | 2057 |   100.00 | Using index
                            |
|  1 | PRIMARY     | partsupp | ref  | i_ps_suppkey          | i_ps_suppkey  | 4
 | dbt3_scale_10.supplier.s_suppkey |   42 |   100.00 | Using join buffer (Batched Key
 Access with Key Prefetching) |
|  2 | SUBQUERY    | nation   | ALL  | PRIMARY               | NULL          | NULL
 | NULL                              |   25 |   100.00 | Using where
                            |
|  2 | SUBQUERY    | supplier | ref  | PRIMARY,i_s_nationkey | i_s_nationkey | 5
 | dbt3_scale_10.nation.n_nationkey | 2057 |   100.00 | Using index
                            |
|  2 | SUBQUERY    | partsupp | ref  | i_ps_suppkey          | i_ps_suppkey  | 4
 | dbt3_scale_10.supplier.s_suppkey |   42 |   100.00 | Using join buffer (Batched Key
 Access with Key Prefetching) |
```

```
+----+-------------+----------+------+----------------------+--------------
+---------+-----------------------------------+------+----------
+-------------------------------------------------------------+
6 rows in set, 1 warning (0.00 sec)
```

有关 EXPLAIN 输出格式的更多信息，请参阅 MySQL 文档中的扩展的 EXPLAIN 输出格式。

使用哈希联接优化大型 Aurora MySQL 联接查询

在需要使用 equijoin 联接大量数据时，哈希联接可以提高查询性能。您可以为 Aurora MySQL 启用哈希联接。

哈希联接列可以是任何复杂表达式。在哈希联接列中，您可以使用以下方法比较不同的数据类型：

- 您可以比较精确数值数据类型类别中的任意类型，例如，int、bigint、numeric 和 bit。
- 您可以比较近似数值数据类型类别中的任意类型，例如，float 和 double。
- 如果字符串类型具有相同的字符集和排序规则，则可以比较具有这些类型的项目。
- 如果日期和时间戳数据类型相同，则可以比较具有这些类型的项目。

> **Note**
>
> 无法比较不同类别的数据类型。

以下限制适用于 Aurora MySQL 的哈希联接：

- Aurora MySQL 版本 2 不支持左右外部联接，但版本 3 支持。
- 不支持半联接（如子查询），除非先具体化子查询。
- 不支持多表更新或删除。

> **Note**
>
> 支持单表更新或删除。

- BLOB 以及空间数据类型列不能是哈希联接中的联接列。

启用哈希联接

启用哈希联接：

- Aurora MySQL 版本 2 – 将数据库参数或数据库集群参数 `aurora_disable_hash_join` 设置为 `0`。关闭 `aurora_disable_hash_join` 会将 `optimizer_switch` 的值设置为 `hash_join=on`。
- Aurora MySQL 版本 3 – 将 MySQL 服务器参数 `optimizer_switch` 设置为 `block_nested_loop=on`。

哈希联接在 Aurora MySQL 版本 3 中原定设置情况下开启，而在 Aurora MySQL 版本 2 中原定设置情况下关闭。以下示例说明了如何为 Aurora MySQL 版本 3 启用哈希联接。您可以先发布语句 `select @@optimizer_switch`，以了解 SET 参数字符串中存在的其他设置。更新 `optimizer_switch` 参数中的一个设置不会删除或修改其他设置。

```
mysql> SET optimizer_switch='block_nested_loop=on';
```

> **Note**
>
> 对于 Aurora MySQL 版本 3，所有次要版本均支持哈希联接，并且原定设置开启哈希联接。
> 对于 Aurora MySQL 版本 2，所有次要版本均支持哈希联接。在 Aurora MySQL 版本 2 中，哈希联接功能始终由 `aurora_disable_hash_join` 值控制。

在使用该设置时，优化程序选择使用基于成本、查询特性和资源可用性的哈希联接。如果成本估算不正确，您可以强制优化程序选择一个哈希联接。为此，请将 MySQL 服务器变量 `hash_join_cost_based` 设置为 `off`。以下示例说明了如何强制优化程序选择哈希联接。

```
mysql> SET optimizer_switch='hash_join_cost_based=off';
```

> **Note**
>
> 此设置将覆盖基于成本的优化程序的决策。虽然该设置对于测试和开发很有用，但我们建议您不要在生产中使用它。

优化哈希联接的查询

要确定查询是否可以使用哈希联接，请先使用 EXPLAIN 语句分析查询。EXPLAIN 语句提供有关用于指定查询的执行计划的信息。

在 EXPLAIN 语句的输出中，Extra 列描述执行计划附带的其他信息。如果哈希联接应用于查询中使用的表，该列将包含类似下面的值：

- `Using where; Using join buffer (Hash Join Outer table table1_name)`
- `Using where; Using join buffer (Hash Join Inner table table2_name)`

以下示例说明了如何使用 EXPLAIN 查看哈希联接查询的执行计划。

```
mysql> explain SELECT sql_no_cache * FROM hj_small, hj_big, hj_big2
    ->     WHERE hj_small.col1 = hj_big.col1 and hj_big.col1=hj_big2.col1 ORDER BY 1;
+----+-------------+----------+------+---------------+------+---------+------+------
+----------------------------------------------------------------+
| id | select_type | table    | type | possible_keys | key  | key_len | ref  | rows |
 Extra                                                          |
+----+-------------+----------+------+---------------+------+---------+------+------
+----------------------------------------------------------------+
|  1 | SIMPLE      | hj_small | ALL  | NULL          | NULL | NULL    | NULL |    6 |
 Using temporary; Using filesort                                |
|  1 | SIMPLE      | hj_big   | ALL  | NULL          | NULL | NULL    | NULL |   10 |
 Using where; Using join buffer (Hash Join Outer table hj_big)  |
|  1 | SIMPLE      | hj_big2  | ALL  | NULL          | NULL | NULL    | NULL |   15 |
 Using where; Using join buffer (Hash Join Inner table hj_big2) |
+----+-------------+----------+------+---------------+------+---------+------+------
+----------------------------------------------------------------+
3 rows in set (0.04 sec)
```

在输出中，Hash Join Inner table 是用于构建哈希表的表，Hash Join Outer table 是用于探查哈希表的表。

有关扩展的 EXPLAIN 输出格式的更多信息，请参阅 MySQL 产品文档中的扩展的 EXPLAIN 输出格式。

在 Aurora MySQL 2.08 及更高版本中，您可以使用 SQL 提示来影响查询是否使用哈希联接，以及用于联接的构建和探查端的表。有关详细信息，请参阅 Aurora MySQL 提示。

使用 Amazon Aurora 为 MySQL 数据库扩展读取

您可以将 Amazon Aurora 用于 MySQL 数据库实例，以便利用 Amazon Aurora 的读取扩展功能并为 MySQL 数据库实例扩展读取工作负载。要使用 Aurora 对 MySQL 数据库实例进行读取扩展，请创建 Aurora MySQL 数据库集群并使它成为 MySQL 数据库实例的只读副本。然后连接到 Aurora MySQL 集群以处理读取查询。源数据库可以是 RDS for MySQL 数据库实例或是在 Amazon RDS 外部运行的 MySQL 数据库。有关更多信息，请参阅 使用 Amazon Aurora 扩展 MySQL 数据库的读取。

优化时间戳操作

当系统变量 `time_zone` 的值设置为 `SYSTEM` 时，每个需要时区计算的 MySQL 函数调用都会进行系统库调用。当您运行在高并发条件下返回或更改此类 `TIMESTAMP` 值的 SQL 语句时，可能会遇到延迟、锁定争用和 CPU 使用率增加的情况。有关更多信息，请参阅 MySQL 文档中的 time_zone。

为避免这种行为，我们建议您将 `time_zone` 数据库集群参数的值更改为 `UTC`。有关更多信息，请参阅 在 Amazon Aurora 中修改数据库集群参数组中的参数。

虽然 `time_zone` 参数是动态的（不需要重新启动数据库服务器），但新值仅用于新连接。为确保更新所有连接以使用新的 `time_zone` 值，我们建议您在更新数据库集群参数后回收应用程序连接。

虚拟索引 ID 溢出错误

Aurora MySQL 将虚拟索引 ID 的值限制为 8 位，以防出现由 MySQL 中的撤消格式引起的问题。如果索引超过虚拟索引 ID 限制，则集群可能不可用。当索引接近虚拟索引 ID 限制时，或者当您尝试创建高于虚拟索引 ID 限制的索引时，RDS 可能会引发错误代码 63955 或警告代码 63955。要解决虚拟索引 ID 限制错误，我们建议您使用逻辑转储和还原来重新创建数据库。

有关 Amazon Aurora MySQL 的逻辑转储和还原的更多信息，请参阅 Migrate very large databases to Amazon Aurora MySQL using MyDumper and MyLoader。有关在 Amazon Aurora 中访问错误日志的更多信息，请参阅监控 Amazon Aurora 日志文件。

Aurora MySQL 高可用性的最佳实践

您可以应用以下最佳实践来改进 Aurora MySQL 集群的可用性。

主题

- 使用 Amazon Aurora 实现 MySQL 数据库的灾难恢复
- 从 MySQL 迁移到 Amazon Aurora MySQL 并减少停机时间

- 避免 Aurora MySQL 数据库实例性能降低、自动重启和故障转移

使用 Amazon Aurora 实现 MySQL 数据库的灾难恢复

您可以在 MySQL 数据库实例中使用 Amazon Aurora 创建异地备份以进行灾难恢复。要使用 Aurora 实现 MySQL 数据库实例的灾难恢复，请创建 Amazon Aurora 数据库集群并使它成为 MySQL 数据库实例的只读副本。这适用于 RDS for MySQL 数据库实例或是在 Amazon RDS 外部运行的 MySQL 数据库。

> **Important**
>
> 在 MySQL 数据库实例和 Amazon Aurora MySQL 数据库集群之间设置复制时，应监控复制以确保它正常运行并在必要时进行修复。

有关如何创建 Amazon Aurora MySQL 数据库集群并使它成为 MySQL 数据库实例的只读副本的说明，请遵循 使用 Amazon Aurora 为 MySQL 数据库扩展读取 中的过程。

有关灾难恢复模型的更多信息，请参阅如何为 Amazon Aurora MySQL 集群选择最佳灾难恢复选项。

从 MySQL 迁移到 Amazon Aurora MySQL 并减少停机时间

将数据从支持活动应用程序的 MySQL 数据库导入到 Amazon Aurora MySQL 数据库集群时，您可能希望缩短在迁移时发生服务中断的时间。为此，您可以使用 Amazon Relational Database Service 用户指南的将数据导入到 MySQL 或 MariaDB 数据库实例并减少停机时间中介绍的过程。如果使用非常大的数据库，该步骤可能是非常有用的。您可以使用该步骤最大限度减少通过网络传送到 AWS 的数据量以降低导入成本。

该过程所列的步骤可将数据库数据的副本传输到 Amazon EC2 实例，并将数据导入到新的 RDS for MySQL 数据库实例。因为 Amazon Aurora 与 MySQL 兼容，所以您可以为目标 Amazon RDS MySQL 数据库实例改用 Amazon Aurora 数据库集群。

避免 Aurora MySQL 数据库实例性能降低、自动重启和故障转移

如果您正在运行繁重的工作负载，或峰值工作负载超出为数据库实例分配的资源，则可能会耗尽运行应用程序和 Aurora 数据库的资源。要获取有关数据库实例的指标，例如 CPU 利用率、内存使用率和使用的数据库连接数，您可以参考 Amazon CloudWatch、Performance Insights 和增强型监控所提供的指标。有关监控数据库实例的更多信息，请参阅 监控 Amazon Aurora 集群中的指标。

如果您的工作负载耗尽了您正在使用的资源，则数据库实例可能会变慢、重新启动，甚至故障转移到另一个数据库实例。为避免这种情况，请监控您的资源利用率，检查数据库实例上运行的工作负载，并在必要时进行优化。如果优化不能改善实例指标和缓解资源耗尽问题，请考虑在达到数据库实例的限制之前对其进行纵向扩展。有关可用数据库实例类及其规格的更多信息，请参阅 Amazon Aurora 数据库实例类。

Aurora MySQL 中 MySQL 功能的建议

以下功能在 Aurora MySQL 中可用于实现 MySQL 兼容性。但是，它们在 Aurora 环境中存在性能、可扩展性、稳定性或兼容性问题。因此，我们建议您在使用这些功能时遵循某些准则。例如，我们不建议使用某些功能进行生产 Aurora 部署。

主题

- 在 Aurora MySQL 中使用多线程复制
- 使用本机 MySQL 函数调用 AWS Lambda 函数
- 避免将 XA 事务与 Amazon Aurora MySQL 结合使用
- 在 DML 语句执行期间保持打开外键
- 配置刷新日志缓冲区的频率
- 最大限度地减少 Aurora MySQL 死锁以及排查相关问题

在 Aurora MySQL 中使用多线程复制

使用多线程二进制日志复制时，SQL 线程会从中继日志中读取事件并将其排队，以便 SQL 工作线程应用。SQL 工作线程由协调器线程管理。尽可能并行应用二进制日志事件。

Aurora MySQL 版本 3 和 Aurora MySQL 2.12.1 及更高版本支持多线程复制。

对于低于 3.04 的 Aurora MySQL 版本，当 Aurora MySQL 数据库集群用作二进制日志复制的只读副本时，Aurora 默认情况下使用单线程复制。

早期的 Aurora MySQL 版本 2 继承了 MySQL Community Edition 中多线程复制方面存在的一些问题。对于这些版本，建议您不要在生产环境中使用多线程复制。

如果您确实要使用多线程复制，建议您对其进行全面测试。

有关在 Amazon Aurora 中使用复制的更多信息，请参阅使用 Amazon Aurora 进行复制。有关 Aurora MySQL 中多线程复制的更多信息，请参阅多线程二进制日志复制。

使用本机 MySQL 函数调用 AWS Lambda 函数

我们建议使用原生 MySQL 函数 `lambda_sync` 和 `lambda_async` 以调用 Lambda 函数。

如果使用已弃用的 `mysql.lambda_async` 过程，我们建议您将对 `mysql.lambda_async` 过程的调用封装在一个存储过程中。您可以从不同的来源调用该存储过程，例如，触发器或客户端代码。这种方法可以帮助您避免出现阻抗不一致问题，并使数据库编程人员更轻松地调用 Lambda 函数。

有关从 Amazon Aurora 中调用 Lambda 函数的更多信息，请参阅 从 Amazon Aurora MySQL 数据库集群中调用 Lambda 函数。

避免将 XA 事务与 Amazon Aurora MySQL 结合使用

我们建议您不要在 Aurora MySQL 中使用扩展架构 (XA) 事务，因为在 XA 处于 PREPARED 状态时，它们可能会造成较长的恢复时间。如果必须在 Aurora MySQL 中使用 XA 事务，请遵循以下最佳实践：

- 不要在 PREPARED 状态下打开 XA 事务。
- 使 XA 事务尽可能小。

有关将 XA 事务与 MySQL 结合使用的更多信息，请参阅 MySQL 文档中的 XA 事务。

在 DML 语句执行期间保持打开外键

当 `foreign_key_checks` 变量设置为 `0` (off) 时，我们强烈建议您不要运行任何数据定义语言 (DDL) 语句。

如果您需要插入或更新暂时违反外键的行，请执行以下步骤：

1. 将 `foreign_key_checks` 设置为 `0`。
2. 执行数据操纵语言 (DML) 更改。
3. 确保您完成的更改不会违反任何外键约束。
4. 将 `foreign_key_checks` 设置为 `1` (on)。

此外，请遵循外键约束的这些其他最佳实践：

- 确保客户端应用程序未将 `foreign_key_checks` 变量作为 `0` 变量的一部分设置为 `init_connect`。

- 如果从诸如 mysqldump 这样的逻辑备份还原失败或还原不完整，请务必先将 foreign_key_checks 设置为 1，然后再在同一会话中开始任何其他操作。当逻辑备份开始时，它会将 foreign_key_checks 设置为 0。

配置刷新日志缓冲区的频率

在 MySQL 社区版中，为了使事务持久，必须将 InnoDB 日志缓冲区刷新到持久存储。您可以使用 innodb_flush_log_at_trx_commit 参数来配置将日志缓冲区刷新到磁盘的频率。

在将 innodb_flush_log_at_trx_commit 参数设置为默认值 1 时，每次事务提交时都会刷新日志缓冲区。此设置有助于保持数据库符合 ACID 要求。我们建议您保留原定设置 1。

将 innodb_flush_log_at_trx_commit 更改为非默认值有助于减少数据操作语言（DML）延迟，但会牺牲日志记录的持久性。这种缺乏持久性使数据库不符合 ACID 要求。我们建议您的数据库符合 ACID 要求，以避免在服务器重新启动时丢失数据的风险。有关此参数的更多信息，请参阅 MySQL 文档中的 innodb_flush_log_at_trx_commit。

在 Aurora MySQL 中，重做日志处理被卸载到存储层，因此数据库实例上不会发生刷新到日志文件的过程。发出写入操作时，重做日志将从写入器数据库实例直接发送到 Aurora 集群卷。跨网络执行的仅有的写入操作是写入重做日志记录。始终不会从数据库层写入任何页面。

默认情况下，每个提交事务的线程都要等待来自 Aurora 集群卷的确认。此确认表明该记录和所有先前的重做日志记录均已写入并已达到法定数目。无论是通过自动提交还是显式提交，保留日志记录并达到法定数目都会使事务变得持久。有关 Aurora 存储架构的更多信息，请参阅 Amazon Aurora 存储揭秘。

Aurora MySQL 不会像 MySQL 社区版那样将日志刷新到数据文件。但是，在将重做日志记录写入 Aurora 集群卷时，您可以使用 innodb_flush_log_at_trx_commit 参数放松持久性约束。

对于 Aurora MySQL 版本 2：

- innodb_flush_log_at_trx_commit = 0 或 2 – 数据库不会等待有关重做日志记录已写入 Aurora 集群卷的确认。
- innodb_flush_log_at_trx_commit = 1 – 数据库会等待有关重做日志记录已写入 Aurora 集群卷的确认。

对于 Aurora MySQL 版本 3：

- `innodb_flush_log_at_trx_commit` = 0 – 数据库不会等待有关重做日志记录已写入 Aurora 集群卷的确认。
- `innodb_flush_log_at_trx_commit` = 1 或 2 – 数据库会等待有关重做日志记录已写入 Aurora 集群卷的确认。

因此，要在 Aurora MySQL 版本 3 中获得与 Aurora MySQL 版本 2 中将该值设置为 0 或 2 相同的非默认行为，请将此参数设置为 0。

虽然这些设置可以降低客户端的 DML 延迟，但也可能导致在故障转移或重新启动时丢失数据。因此，我们建议您将 `innodb_flush_log_at_trx_commit` 参数保持设置为默认值 1。

虽然在 MySQL 社区版和 Aurora MySQL 中都可能发生数据丢失，但由于每个数据库的架构不同，对于它们的行为也有所不同。这些架构差异可能导致不同程度的数据丢失。要确保您的数据库符合 ACID 要求，请始终将 `innodb_flush_log_at_trx_commit` 设置为 1。

> **Note**
>
> 在 Aurora MySQL 版本 3 中，在将 `innodb_flush_log_at_trx_commit` 更改为 1 以外的值之前，必须先将 `innodb_trx_commit_allow_data_loss` 的值更改为 1。这样做即表示您承认数据丢失的风险。

最大限度地减少 Aurora MySQL 死锁以及排查相关问题

运行工作负载的用户在同时修改同一数据页上的记录时，如果经常遇到对唯一的二级索引或外键违反约束的情形，则可能会遇到更多的死锁和锁等待超时。这些死锁和超时是由于 MySQL 社区版错误修复造成的。

此修复包含在 MySQL 社区版版本 5.7.26 及更高版本中，并已向后移植到 Aurora MySQL 版本 2.10.3 及更高版本中。该修复对于实施可序列性是必要的，其具体方法为：对于这些类型的数据操作语言（DML）操作，针对 InnoDB 表中的记录更改实施额外的锁定。此问题是在调查先前的 MySQL 社区版错误修复所引入的死锁问题时发现的。

该修复更改了对于 InnoDB 存储引擎中元组（行）更新的部分回滚的内部处理。在外键或唯一二级索引上生成违反约束的操作会导致部分回滚。这包括但不限于并发 `INSERT...ON DUPLICATE KEY UPDATE`、`REPLACE INTO`，和 `INSERT IGNORE` 语句（upsert）。

在这种情况下，部分回滚不是指应用程序级事务的回滚，而是当遇到违反约束的情形时，InnoDB 对聚集索引更改进行内部回滚。例如，在 upsert 操作期间发现了重复的键值。

在正常的插入操作中，InnoDB 会自动为每个索引创建聚集和二级索引条目。如果 InnoDB 在 upsert 操作期间在唯一二级索引上检测到重复值，则必须还原聚集索引中插入的条目（部分回滚），然后必须将更新应用于现有的重复行。在此内部部分回滚步骤中，InnoDB 必须锁定被视为操作的一部分的每条记录。该修复通过在部分回滚后引入额外的锁定来确保事务的可序列性。

最大限度地减少 InnoDB 死锁

您可以采用以下方法来减少数据库实例中出现死锁的频率。可以在 MySQL 文档中找到更多示例。

1. 为了减少出现死锁的几率，请在进行一组相关更改后立即提交事务。可以通过将大型事务（两次提交之间的多行更新）分解为较小的事务来做到这一点。如果您要批量插入行，则请尝试减少批量插入大小，尤其是在使用前面提到的 upsert 操作时。

   要减少可能的部分回滚次数，可以尝试以下一些方法：

   a. 将批量插入操作替换为一次插入一行。这可以减少可能存在冲突的事务持有锁的时间量。

   b. 不使用 `REPLACE INTO`，而是将 SQL 语句重写为多语句事务，如以下所示：

   ```
   BEGIN;
   DELETE conflicting rows;
   INSERT new rows;
   COMMIT;
   ```

   c. 不使用 `INSERT...ON DUPLICATE KEY UPDATE`，而是将 SQL 语句重写为多语句事务，如以下所示：

   ```
   BEGIN;
   SELECT rows that conflict on secondary indexes;
   UPDATE conflicting rows;
   INSERT new rows;
   COMMIT;
   ```

2. 避免长时间运行的事务，无论是活跃的还是空闲的，因为它们可能会被锁定。这包括交互式 MySQL 客户端会话，这些会话可能会在未提交的事务中打开很长一段时间。在优化事务大小或批次大小时，影响可能会有所不同，具体取决于许多因素，例如并发性、重复项数和表结构。任何更改都应根据您的工作负载进行实施和测试。
3. 在某些情况下，当两个事务尝试以不同的顺序访问一个或多个表中的相同数据集时，可能会出现死锁。为防止这种情况，您可以修改事务以按相同顺序访问数据，从而使访问序列化。例如，创建要完成的事务队列。当多个事务同时发生时，这种方法可以帮助避免死锁。
4. 向表中添加精心选择的索引可以提高选择性并减少访问行的需求，从而减少锁定。

5. 如果您遇到间隙锁定，可以将会话或事务的事务隔离级别修改为 READ COMMITTED，以防止出现这种情况。有关 InnoDB 隔离级别及其行为的更多信息，请参阅 MySQL 文档中的事务隔离级别。

> **Note**
>
> 虽然您可以采取预防措施来降低死锁发生的可能性，但死锁是数据库的预期行为，仍然可能发生。应用程序应具有必要的逻辑，以便在遇到死锁时进行处理。例如，在应用程序中实现重试和退避逻辑。最好解决问题的根本原因，但如果确实出现死锁，应用程序可以选择等待并重试。

监控 InnoDB 死锁

当应用程序事务尝试以导致循环等待的方式获取表级和行级锁定时，MySQL 中可能会发生死锁。偶尔的 InnoDB 死锁不一定是问题，因为 InnoDB 存储引擎会立即检测到这一情况并自动回滚其中一个事务。如果您经常遇到死锁，我们建议您检查和修改您的应用程序，以缓解性能问题并避免死锁。开启死锁检测（原定设置）时，InnoDB 会自动检测事务死锁并回滚一个或多个事务以打破死锁。InnoDB 尝试挑选要回滚的小事务，其中事务的大小由插入、更新或删除的行数决定。

- SHOW ENGINE 语句 – SHOW ENGINE INNODB STATUS \G 语句包含自上次重新启动以来在数据库上遇到的最新死锁的详细信息。
- MySQL 错误日志 – 如果您经常遇到死锁，其中 SHOW ENGINE 语句的输出不充足，则可以开启 innodb_print_all_deadlocks 数据库集群参数。

  开启此参数后，有关 InnoDB 用户事务中所有死锁的信息将记录在 Aurora MySQL 错误日志中。
- Amazon CloudWatch 指标 – 我们还建议您使用 CloudWatch 指标 Deadlocks 主动监控死锁。有关更多信息，请参阅 Amazon Aurora 的实例级指标。
- Amazon CloudWatch Logs – 利用 CloudWatch Logs，您可以查看指标、分析日志数据并创建实时警报。有关更多信息，请参阅使用 Amazon CloudWatch 监控 Amazon Aurora MySQL 和 Amazon RDS for MySQL 中的错误，并使用 Amazon SNS 发送通知。

  使用已开启 innodb_print_all_deadlocks 的 CloudWatch Logs，您可以配置警报，以便在死锁数量超过给定阈值时通知您。要定义阈值，我们建议您观察趋势并使用基于正常工作负载的值。
- Performance Insights – 当您使用 Performance Insights 时，您可以监控 innodb_deadlocks 和 innodb_lock_wait_timeout 指标。有关这些指标的更多信息，请参阅 Aurora MySQL 的非本机计数器。

使用 Amazon CloudWatch 指标评估 Aurora MySQL 数据库实例使用情况

您可以使用 CloudWatch 指标来监控数据库实例吞吐量，并确定您的数据库实例类是否为应用程序提供了足够的资源。有关数据库实例类的信息，请参阅适用于 Aurora 的数据库实例类的硬件规格。查找数据库实例类的规格以了解网络性能。

如果您的数据库实例使用情况接近实例类限制，则性能可能会开始变慢。CloudWatch 指标可以证实这种情况，因此您可以计划手动纵向扩展到更大的实例类。

结合使用以下 CloudWatch 指标值，看看您是否接近实例类限制：

- NetworkThroughput –Aurora 数据库集群中每个实例的客户端接收和发送的网络吞吐量。此吞吐量值不包括数据库集群中的实例与集群卷之间的网络流量。
- StorageNetworkThroughput – Aurora 数据库集群中每个实例从 Aurora 存储子系统接收以及发送到此子系统的网络吞吐量。

将 NetworkThroughput 与 StorageNetworkThroughput 相加，以找出 Aurora 数据库集群中每个实例从 Aurora 存储子系统接收和发送到此系统的网络吞吐量。您的实例的实例类限制应大于这两个组合指标的总和。

在发送和接收时，您可以使用以下指标来查看来自客户端应用程序的网络流量的更多详细信息：

- NetworkReceiveThroughput – Aurora MySQL 数据库集群中的每个数据库实例从客户端接收的网络吞吐量。此吞吐量不包括数据库集群中的实例与集群卷之间的网络流量。
- NetworkTransmitThroughput – Aurora 数据库集群中每个实例发送到客户端的网络吞吐量。此吞吐量不包括数据库集群中的实例与集群卷之间的网络流量。
- StorageNetworkReceiveThroughput – 数据库集群中每个实例从 Aurora 存储子系统接收的网络吞吐量。
- StorageNetworkTransmitThroughput – Aurora MySQL 数据库集群中每个实例发送到 Aurora 存储子系统的网络吞吐量。

将所有这些指标相加，评估您的网络使用量与数据库实例类限制的对比情况。实例类限制应大于这些组合指标的总和。

网络限制和存储的 CPU 利用率是直接相关的。当网络吞吐量增加时，CPU 利用率也会增加。监控 CPU 和网络使用情况可提供有关资源耗尽的方式和原因的信息。

为帮助显著减少网络使用量，您可以考虑以下措施：

- 使用更大的数据库实例类。
- 批量划分写入请求以减少总体事务量。
- 将只读工作负载定向到只读实例。
- 删除任何未使用的索引。

排除 Amazon Aurora MySQL 数据库性能故障

本主题重点介绍一些常见的 Aurora MySQL 数据库性能问题，以及如何进行故障排除或收集信息来快速修复这些问题。我们将数据库性能分为两类：

- 服务器性能 - 整个数据库服务器的运行速度较慢。
- 查询性能 - 一个或多个查询的运行时间较长。

AWS 监控选项

我们建议您使用以下 AWS 监控选项来协助进行故障排除：

- Amazon CloudWatch – Amazon CloudWatch 实时监控 AWS 资源以及在 AWS 上运行的应用程序。您可以使用 CloudWatch 收集和跟踪指标，这些指标是您可以针对资源和应用程序衡量的变量。有关更多信息，请参阅 What is Amazon CloudWatch?

  您可以在 AWS Management Console上查看数据库实例的所有系统指标和过程信息。您可以配置 Aurora MySQL 数据库集群以将一般、慢速、审核和错误日志数据发布到 Amazon CloudWatch Logs 中的日志组。这样，您就可以查看趋势，在主机受到影响时维护日志，并创建“正常”性能的基准来轻松地识别异常或变化。有关更多信息，请参阅 将 Amazon Aurora MySQL 日志发布到 Amazon CloudWatch Logs。
- 增强监控 – 要为 Aurora MySQL 数据库启用其它 Amazon CloudWatch 指标，请开启增强监控。创建或修改 Aurora 数据库集群时，请选择启用增强监控。这样，Aurora 就可以向 CloudWatch 发布性能指标。可用的一些关键指标包括 CPU 使用率、数据库连接、存储使用情况和查询延迟。这些指标有助于确定性能瓶颈。

  为数据库实例传输的信息量与为增强监控功能定义的粒度成正比。监控间隔越短，操作系统指标报告频率越高，监控成本也就越高。要管理成本，请为您的 AWS 账户中的不同实例设置不同的粒度。创建实例时的默认粒度为 60 秒。有关更多信息，请参阅 增强监测的成本。
- Performance Insights – 您可以查看所有数据库调用指标。这包括数据库锁定、等待和已处理的行数，所有这些都可用于进行故障排除。创建或修改 Aurora 数据库集群时，请选择开启 Performance Insights。默认情况下，Performance Insights 的数据留存期为 7 天，但可以对其进行自定义来分析长期性能趋势。如果留存期超过 7 天，则需要升级到付费套餐。有关更多信息，请参阅 Performance Insights 定价。您可以分别为每个 Aurora 数据库实例设置数据留存期。有关更多信息，请参阅 在 Amazon Aurora 上使用性能详情监控数据库负载。

出现 Aurora MySQL 数据库性能问题的最常见原因

您可以使用以下步骤来解决 Aurora MySQL 数据库中的性能问题。我们按调查的逻辑顺序列出了这些步骤，但它们并不是线性的。一项发现可能跨越多个步骤，从而形成一系列调查路径。

1. 工作负载 – 了解您的数据库工作负载。
2. 日志记录 – 查看所有数据库日志。
3. 数据库连接 – 确保应用程序与数据库之间的连接可靠。
4. 查询性能 – 检查您的查询执行计划，查看它们是否已更改。代码更改可能会导致计划发生变化。

排查 Aurora MySQL 数据库的工作负载问题

数据库工作负载可以看作是读取和写入。了解“正常”数据库工作负载后，您可以调整查询和数据库服务器，来满足不断变化的需求。性能可能发生变化的原因有很多，因此第一步是了解发生了什么变化。

- 是否进行了主要版本或次要版本升级？

  主要版本升级包括对引擎代码的更改，尤其是优化器中的更改，这些更改可能会更改查询执行计划。升级数据库版本，尤其是主要版本时，分析数据库工作负载并相应进行调整非常重要。调整可能包括优化和重写查询，或者添加和更新参数设置，具体取决于测试的结果。了解造成影响的原因将使您能够开始专注于该特定区域。

  有关更多信息，请参阅 MySQL 文档中的 What is new in MySQL 8.0 和 Server and status variables and options added, deprecated, or removed in MySQL 8.0，以及比较 Aurora MySQL 版本 2 和 Aurora MySQL 版本 3。
- 正在处理的数据（行计数）是否有所增加？
- 是否有更多查询并行运行？
- 模式或数据库是否发生变化？
- 是否存在代码缺陷或修复？

目录

- 实例主机指标
  - CPU 使用率
  - 内存使用量
  - 网络吞吐量
- 数据库指标
- 排查 Aurora MySQL 数据库的内存使用问题
  - 示例 1：持续的高内存使用量
  - 示例 2：短暂内存峰值
  - 示例 3：可释放内存持续下降且无法回收
- 排查 Aurora MySQL 数据库内存不足问题

实例主机指标

监控 CPU、内存和网络活动等实例主机指标，来协助了解工作负载是否发生了变化。了解工作负载变化有两个主要概念：

- 利用率 - 设备（例如 CPU 或磁盘）的使用情况。它可以是基于时间的，也可以是基于容量的。
  - 基于时间 - 资源在特定观察期内忙碌的时间量。
  - 基于容量 – 系统或组件可以提供的吞吐量，以其容量的百分比表示。
- 饱和度 - 资源需要的工作量超过其处理能力的程度。当基于容量的使用率达到 100% 时，将无法处理额外的工作，必须排队。

CPU 使用率

您可以使用以下工具来确定 CPU 使用率和饱和度：

- CloudWatch 提供了 `CPUUtilization` 指标。如果该指标达到 100%，则实例饱和。但是，CloudWatch 指标按 1 分钟取平均值，因而粒度不足。

  有关 CloudWatch 指标的更多信息，请参阅Amazon Aurora 的实例级指标。
- 增强监控提供由操作系统 `top` 命令返回的指标。它以 1 秒的粒度显示平均负载和以下 CPU 状态：
  - `Idle (%)` = 空闲时间
  - `IRQ (%)` = 软件中断
  - `Nice (%)` = 对于优先级为 niced 的进程，状态为 Nice 的时间。
  - `Steal (%)` = 为其他租户提供服务所花费的时间（与虚拟化相关）
  - `System (%)` = 系统时间
  - `User (%)` = 用户时间
  - `Wait (%)` = I/O 等待

  有关增强监控指标的更多信息，请参阅 Aurora 的操作系统指标。

内存使用量

如果系统面临内存压力，并且资源消耗达到饱和，则应观察到高程度的页面扫描、分页、交换和内存不足错误。

您可以使用以下工具来确定内存使用量和饱和度：

CloudWatch 提供了 FreeableMemory 指标，该指标显示通过刷新部分操作系统缓存和当前可用内存可以回收多少内存。

有关 CloudWatch 指标的更多信息，请参阅Amazon Aurora 的实例级指标。

增强监控提供以下指标，有助于您识别内存使用量问题：

- Buffers（KB）– 在写入存储设备前用于缓冲 I/O 请求的内存量（以 KB 为单位）。
- Cached（KB）– 用于缓存基于文件系统的 I/O 的内存量。
- Free（KB）– 未分配的内存量（以 KB 为单位）。
- Swap –“已缓存”、“可用”和“总计”。

例如，如果您看到您的数据库实例使用 Swap 内存，则您的工作负载的内存总量将大于您的实例当前可用的内存量。我们建议增加数据库实例的大小或调整工作负载来使用更少的内存。

有关增强监控指标的更多信息，请参阅 Aurora 的操作系统指标。

有关使用性能架构和 sys 架构来确定哪些连接和组件正在使用内存的更多详细信息，请参阅 排查 Aurora MySQL 数据库的内存使用问题。

网络吞吐量

CloudWatch 提供以下网络总吞吐量指标，所有指标均按 1 分钟取平均值：

- NetworkReceiveThroughput – Aurora 数据库集群中每个实例从客户端接收的网络吞吐量。
- NetworkTransmitThroughput – Aurora 数据库集群中每个实例发送到客户端的网络吞吐量。
- NetworkThroughput – Aurora 数据库集群中每个实例从客户端接收和发送到客户端的网络吞吐量。
- StorageNetworkReceiveThroughput – 数据库集群中每个实例从 Aurora 存储子系统接收的网络吞吐量。
- StorageNetworkTransmitThroughput – Aurora 数据库集群中每个实例发送到 Aurora 存储子系统的网络吞吐量。
- StorageNetworkThroughput – Aurora 数据库集群中每个实例从 Aurora 存储子系统接收与发送到该子系统的网络吞吐量。

有关 CloudWatch 指标的更多信息，请参阅Amazon Aurora 的实例级指标。

增强监控提供 network 已接收（RX）和已发送（TX）图表，粒度高达 1 秒。

有关增强监控指标的更多信息，请参阅 Aurora 的操作系统指标。

数据库指标

检查以下 CloudWatch 指标，来了解工作负载变化：

- BlockedTransactions – 每秒内数据库中被阻止的事务的平均数。
- BufferCacheHitRatio – 缓冲区缓存提供服务的请求的百分比。
- CommitThroughput – 每秒平均提交操作数量。
- DatabaseConnections – 连接至数据库实例的客户端网络连接数。
- Deadlocks – 每秒内数据库中死锁的平均数。
- DMLThroughput – 每秒平均插入、更新和删除数。
- ResultSetCacheHitRatio – 查询缓存提供服务的请求的百分比。
- RollbackSegmentHistoryListLength – 记录已提交事务（带有删除标记的记录）的撤销日志。
- RowLockTime – 为 InnoDB 表获取行锁定所花的总时间。
- SelectThroughput – 每秒平均选择查询数。

有关 CloudWatch 指标的更多信息，请参阅Amazon Aurora 的实例级指标。

检查工作负载时，请考虑以下问题：

1. 数据库实例类最近是否发生了变化，例如，将实例大小从 8xlarge 缩小到 4xlarge，或者从 db.r5 更改为 db.r6？
2. 您能否创建一个克隆并重现此问题，或者该问题只发生在那一个实例上？
3. 是否存在服务器资源耗尽、CPU 过高或内存耗尽的问题？ 如果是，则这可能意味着需要额外的硬件。
4. 一个或多个查询是否需要更长的时间？
5. 这些更改是否由升级（尤其是主要版本升级）引起？ 如果是，则比较升级前和升级后的指标。
6. 读取器数据库实例的数量是否发生了变化？
7. 您是否启用了常规、审计或二进制日志记录？ 有关更多信息，请参阅 Aurora MySQL 数据库日志记录。
8. 您是否启用、禁用或更改了对二进制日志（binlog）复制的使用？

9. 是否存在任何持有大量行锁的长期运行的事务？ 检查 InnoDB 历史记录列表长度（HLL），来获取长时间运行的事务的指示。

   有关更多信息，请参阅InnoDB 历史记录列表长度显著增加和博客文章 Why is my SELECT query running slowly on my Amazon Aurora MySQL DB cluster?

   a. 如果写入事务导致 HLL 较大，则表示 UNDO 日志正在累积（未定期清理）。在大型写入事务中，这种累积可能会迅速增长。在 MySQL 中，UNDO 存储在 SYSTEM 表空间中。SYSTEM 表空间不可收缩。UNDO 日志可能会导致 SYSTEM 表空间增长到若干 GB，甚至 TB。清除后，通过对数据进行逻辑备份（转储）来释放分配的空间，然后将转储导入到新的数据库实例。

   b. 如果较大的 HLL 是由读取事务（长时间运行的查询）引起的，则可能意味着该查询使用了大量的临时空间。通过重启来释放临时空间。检查 Performance Insights 数据库指标，来了解 Temp 部分（例如 `created_tmp_tables`）是否有任何变化。有关更多信息，请参阅 在 Amazon Aurora 上使用性能详情监控数据库负载。

10. 能否将长时间运行的事务拆分为修改较少行的较小事务？

11. 被阻止的事务是否有任何变化或死锁是否增加？ 检查 Performance Insights 数据库指标，来了解 `Locks` 部分中的状态变量（例如 `innodb_row_lock_time`、 `innodb_row_lock_waits` 和 `innodb_dead_locks`）是否有任何变化。使用 1 分钟或 5 分钟间隔。

12. 等待事件是否增加？ 以 1 分钟或 5 分钟间隔检查 Performance Insights 等待事件和等待类型。分析排名靠前的等待事件，看看它们是否与工作负载变化或数据库争用相关。例如，`buf_pool mutex` 表示缓冲池争用。有关更多信息，请参阅 使用等待事件优化 Aurora MySQL。

排查 Aurora MySQL 数据库的内存使用问题

虽然 CloudWatch、增强监控和性能详情可以很好地概述操作系统级别的内存使用情况，例如数据库进程使用了多少内存，但它们不允许您细分引擎中的哪些连接或组件可能导致这种内存使用。

要对此进行故障排除，您可以使用 Performance Schema 和 `sys` 架构。在 Aurora MySQL 版本 3 中，当启用 Performance Schema 时，默认情况下会启用内存检测。在 Aurora MySQL 版本 2 中，默认情况下，仅对 Performance Schema 内存使用情况启用内存检测。有关 Performance Schema 中可用于跟踪内存使用情况和启用 Performance Schema 内存检测的表的信息，请参阅 MySQL 文档中的 Memory summary tables。有关将 Performance Schema 与性能详情结合使用的更多信息，请参阅Aurora MySQL 上性能详情的性能架构概述。

虽然 Performance Schema 中提供了用于跟踪当前内存使用情况的详细信息，但 MySQL sys schema 在 Performance Schema 表之上具有视图，您可以使用这些视图来快速查明使用内存的位置。

在 `sys` 架构中，可以使用以下视图，按连接、组件和查询来跟踪内存使用情况。

视图	描述
memory_by_host_by_current_bytes	按主机提供有关引擎内存使用情况的信息。这对于识别哪些应用程序服务器或客户端主机正在消耗内存很有用。
memory_by_thread_by_current_bytes	按线程 ID 提供有关引擎内存使用情况的信息。MySQL 中的线程 ID 可以是客户端连接或后台线程。您可以使用 sys.processlist 视图或 performance_schema.threads 表将线程 ID 映射到 MySQL 连接 ID。
memory_by_user_by_current_bytes	按用户提供有关引擎内存使用情况的信息。这对于识别哪些用户账户或客户端正在消耗内存很有用。
memory_global_by_current_bytes	按引擎组件提供有关引擎内存使用情况的信息。这对于按引擎缓冲区或组件全局识别内存使用情况很有用。例如，您可能会看到 InnoDB 缓冲池的 `memory/innodb/buf_buf_pool` 事件，或者预处理语句的 `memory/sql/Prepared_statement::main_mem_root` 事件。
memory_global_total	概述数据库引擎中跟踪的内存使用总量。

在 Aurora MySQL 版本 3.05 及更高版本中，您还可以在 Performance Schema statement summary tables 中按语句摘要跟踪最大内存使用量。语句摘要表包含标准化语句摘要及有关其执行情况的汇总统计数据。`MAX_TOTAL_MEMORY` 列可以帮助您确定自上次重置统计数据以来或自重启数据库实例以来，查询摘要所使用的最大内存。这对于识别可能消耗大量内存的特定查询很有用。

> **Note**
>
> Performance Schema 和 `sys` 架构显示服务器上的当前内存使用情况，以及每个连接和引擎组件消耗的内存的历史最高水平。由于 Performance Schema 在内存中维护，因此数据库实例重

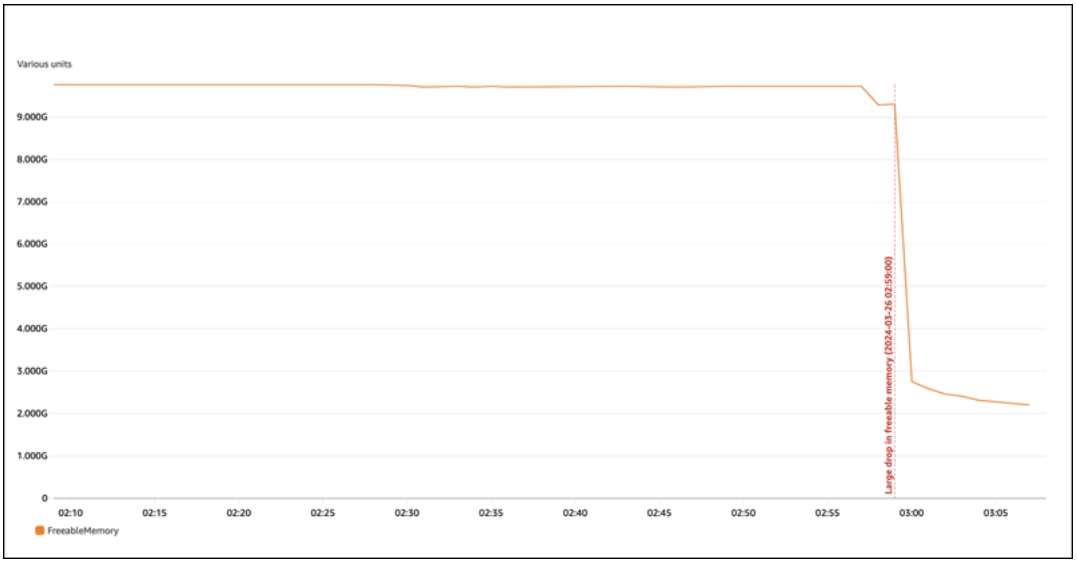
启时会重置信息。为了保持一段时间内的历史记录，建议您在 Performance Schema 之外配置此数据的检索和存储。

主题

- 示例 1：持续的高内存使用量
- 示例 2：短暂内存峰值
- 示例 3：可释放内存持续下降且无法回收

示例 1：持续的高内存使用量

在 CloudWatch 中全局观察 `FreeableMemory`，我们可以看到，在 2024 年 3 月 26 日凌晨 2:59（UTC 时间），内存使用量大幅增加。



这并不能告诉我们全貌。要确定哪个组件使用的内存最多，您可以登录数据库并查看 `sys.memory_global_by_current_bytes`。此表包含 MySQL 跟踪的内存事件列表，以及有关每个事件的内存分配的信息。每个内存跟踪事件都以 `memory/%` 开头，后跟与该事件关联的引擎组件/功能的其它信息。

例如，`memory/performance_schema/%` 用于与 Performance Schema 相关的内存事件，`memory/innodb/%` 用于 InnoDB，等等。有关事件命名约定的更多信息，请参阅 MySQL 文档中的 Performance Schema instrument naming conventions。

从以下查询中，我们可以根据 `current_alloc` 找到可能的罪魁祸首，但我们也可能看到许多 `memory/performance_schema/%` 事件。

```
mysql> SELECT * FROM sys.memory_global_by_current_bytes LIMIT 10;

+-------------------------------------------------------------------------------
+---------------+---------------+-------------------+------------+------------
+----------------+
| event_name                                                                    |
 current_count | current_alloc | current_avg_alloc | high_count | high_alloc |
 high_avg_alloc |
+-------------------------------------------------------------------------------
+---------------+---------------+-------------------+------------+------------
+----------------+
| memory/sql/Prepared_statement::main_mem_root                                  |
 512817 | 4.91 GiB      | 10.04 KiB         |     512823 | 4.91 GiB   | 10.04 KiB
 |
| memory/performance_schema/prepared_statements_instances                       |
   252 | 488.25 MiB    | 1.94 MiB          |        252 | 488.25 MiB | 1.94 MiB       |
| memory/innodb/hash0hash                                                       |
     4 | 79.07 MiB     | 19.77 MiB         |          4 | 79.07 MiB  | 19.77 MiB      |
| memory/performance_schema/events_errors_summary_by_thread_by_error            |
  1028 | 52.27 MiB     | 52.06 KiB         |       1028 | 52.27 MiB  | 52.06 KiB      |
| memory/performance_schema/events_statements_summary_by_thread_by_event_name |
     4 | 47.25 MiB     | 11.81 MiB         |          4 | 47.25 MiB  | 11.81 MiB      |
| memory/performance_schema/events_statements_summary_by_digest                |
     1 | 40.28 MiB     | 40.28 MiB         |          1 | 40.28 MiB  | 40.28 MiB      |
| memory/performance_schema/memory_summary_by_thread_by_event_name              |
     4 | 31.64 MiB     | 7.91 MiB          |          4 | 31.64 MiB  | 7.91 MiB       |
| memory/innodb/memory                                                          |
 15227 | 27.44 MiB     | 1.85 KiB          |      20619 | 33.33 MiB  | 1.66 KiB       |
| memory/sql/String::value                                                      |
 74411 | 21.85 MiB     |  307 bytes        |      76867 | 25.54 MiB  |  348 bytes     |
| memory/sql/TABLE                                                              |
  8381 | 21.03 MiB     | 2.57 KiB          |       8381 | 21.03 MiB  | 2.57 KiB       |
+-------------------------------------------------------------------------------
+---------------+---------------+-------------------+------------+------------
+----------------+
10 rows in set (0.02 sec)
```

我们之前提到过， Performance Schema 存储在内存中，这意味着在 `performance_schema` 内存检测中也会对它进行跟踪。

> **Note**
>
> 如果您发现 Performance Schema 使用了大量内存，并且想要限制其内存使用量，则可以根据需要调整数据库参数。有关更多信息，请参阅 MySQL 文档中的 The Performance Schema memory-allocation model。

为了便于阅读，您可以重新运行相同的查询，但排除 Performance Schema 事件。输出显示以下内容：

- 主要的内存消耗者是 memory/sql/Prepared_statement::main_mem_root。
- current_alloc 列告诉我们，MySQL 当前为此事件分配了 4.91 GiB。
- high_alloc column 告诉我们，4.91 GiB 是自上次重置统计数据或服务器重启以来 current_alloc 的历史最高水平。这意味着 memory/sql/Prepared_statement::main_mem_root 已达到其最高值。

```
mysql> SELECT * FROM sys.memory_global_by_current_bytes WHERE event_name NOT LIKE
 'memory/performance_schema/%' LIMIT 10;

+-----------------------------------------------+---------------+---------------
+------------------+-----------+-----------+----------------+
| event_name                                    | current_count | current_alloc |
 current_avg_alloc | high_count | high_alloc | high_avg_alloc |
+-----------------------------------------------+---------------+---------------
+------------------+-----------+-----------+----------------+
| memory/sql/Prepared_statement::main_mem_root  |        512817 | 4.91 GiB      | 10.04
 KiB         |     512823 | 4.91 GiB   | 10.04 KiB      |
| memory/innodb/hash0hash                       |             4 | 79.07 MiB     | 19.77
 MiB         |          4 | 79.07 MiB  | 19.77 MiB      |
| memory/innodb/memory                          |         17096 | 31.68 MiB     | 1.90
 KiB         |      22498 | 37.60 MiB  | 1.71 KiB       |
| memory/sql/String::value                      |        122277 | 27.94 MiB     |  239
 bytes       |     124699 | 29.47 MiB  |  247 bytes     |
| memory/sql/TABLE                              |          9927 | 24.67 MiB     | 2.55
 KiB         |       9929 | 24.68 MiB  | 2.55 KiB       |
| memory/innodb/lock0lock                       |          8888 | 19.71 MiB     | 2.27
 KiB         |       8888 | 19.71 MiB  | 2.27 KiB       |
| memory/sql/Prepared_statement::infrastructure |        257623 | 16.24 MiB     |   66
 bytes       |     257631 | 16.24 MiB  |   66 bytes     |
```

```
| memory/mysys/KEY_CACHE                          |             3 | 16.00 MiB     | 5.33
 MiB          |          3 | 16.00 MiB  | 5.33 MiB       |
| memory/innodb/sync0arr                          |             3 | 7.03 MiB      | 2.34
 MiB          |          3 | 7.03 MiB   | 2.34 MiB       |
| memory/sql/THD::main_mem_root                   |           815 | 6.56 MiB      | 8.24
 KiB          |        849 | 7.19 MiB   | 8.67 KiB       |
+-------------------------------------------------+---------------+---------------
+------------------+-----------+-----------+---------------+
10 rows in set (0.06 sec)
```

从事件的名称中，我们可以看出此内存正用于预处理语句。如果您想查看哪些连接正在使用此内存，您可以检查 memory_by_thread_by_current_bytes。

在以下示例中，为每个连接分配了大约 7 MiB，历史最高水平约为 6.29 MiB (current_max_alloc)。这是有道理的，因为该示例正在将 sysbench 用于 80 个表和 800 个带有预处理语句的连接。如果您要在这种情况下减少内存使用量，可以优化应用程序对预处理语句的使用，来减少内存消耗。

```
mysql> SELECT * FROM sys.memory_by_thread_by_current_bytes;

+-----------+-----------------------------------------+--------------------
+------------------+------------------+------------------+-----------------+
| thread_id | user                                    | current_count_used |
 current_allocated | current_avg_alloc | current_max_alloc | total_allocated |
+-----------+-----------------------------------------+--------------------
+------------------+------------------+------------------+-----------------+
|        46 | rdsadmin@localhost                      |                405 | 8.47 MiB
          | 21.42 KiB         | 8.00 MiB          | 155.86 MiB      |
|        61 | reinvent@10.0.4.4                       |               1749 | 6.72 MiB
          | 3.93 KiB          | 6.29 MiB          | 14.24 MiB       |
|       101 | reinvent@10.0.4.4                       |               1845 | 6.71 MiB
          | 3.72 KiB          | 6.29 MiB          | 14.50 MiB       |
|        55 | reinvent@10.0.4.4                       |               1674 | 6.68 MiB
          | 4.09 KiB          | 6.29 MiB          | 14.13 MiB       |
|        57 | reinvent@10.0.4.4                       |               1416 | 6.66 MiB
          | 4.82 KiB          | 6.29 MiB          | 13.52 MiB       |
|       112 | reinvent@10.0.4.4                       |               1759 | 6.66 MiB
          | 3.88 KiB          | 6.29 MiB          | 14.17 MiB       |
|        66 | reinvent@10.0.4.4                       |               1428 | 6.64 MiB
          | 4.76 KiB          | 6.29 MiB          | 13.47 MiB       |
|        75 | reinvent@10.0.4.4                       |               1389 | 6.62 MiB
          | 4.88 KiB          | 6.29 MiB          | 13.40 MiB       |
```

```
|       116 | reinvent@10.0.4.4                       |              1333 | 6.61 MiB
          | 5.08 KiB          | 6.29 MiB          | 13.21 MiB       |
|        90 | reinvent@10.0.4.4                       |              1448 | 6.59 MiB
          | 4.66 KiB          | 6.29 MiB          | 13.58 MiB       |
|        98 | reinvent@10.0.4.4                       |              1440 | 6.57 MiB
          | 4.67 KiB          | 6.29 MiB          | 13.52 MiB       |
|        94 | reinvent@10.0.4.4                       |              1433 | 6.57 MiB
          | 4.69 KiB          | 6.29 MiB          | 13.49 MiB       |
|        62 | reinvent@10.0.4.4                       |              1323 | 6.55 MiB
          | 5.07 KiB          | 6.29 MiB          | 13.48 MiB       |
|        87 | reinvent@10.0.4.4                       |              1323 | 6.55 MiB
          | 5.07 KiB          | 6.29 MiB          | 13.25 MiB       |
|        99 | reinvent@10.0.4.4                       |              1346 | 6.54 MiB
          | 4.98 KiB          | 6.29 MiB          | 13.24 MiB       |
|       105 | reinvent@10.0.4.4                       |              1347 | 6.54 MiB
          | 4.97 KiB          | 6.29 MiB          | 13.34 MiB       |
|        73 | reinvent@10.0.4.4                       |              1335 | 6.54 MiB
          | 5.02 KiB          | 6.29 MiB          | 13.23 MiB       |
|        54 | reinvent@10.0.4.4                       |              1510 | 6.53 MiB
          | 4.43 KiB          | 6.29 MiB          | 13.49 MiB       |
.
                                                                    .
.
                                                                    .
.
                                                                    .
|       812 | reinvent@10.0.4.4                       |              1259 | 6.38 MiB
          | 5.19 KiB          | 6.29 MiB          | 13.05 MiB       |
|       214 | reinvent@10.0.4.4                       |              1279 | 6.38 MiB
          | 5.10 KiB          | 6.29 MiB          | 12.90 MiB       |
|       325 | reinvent@10.0.4.4                       |              1254 | 6.38 MiB
          | 5.21 KiB          | 6.29 MiB          | 12.99 MiB       |
|       705 | reinvent@10.0.4.4                       |              1273 | 6.37 MiB
          | 5.13 KiB          | 6.29 MiB          | 13.03 MiB       |
|       530 | reinvent@10.0.4.4                       |              1268 | 6.37 MiB
          | 5.15 KiB          | 6.29 MiB          | 12.92 MiB       |
|       307 | reinvent@10.0.4.4                       |              1263 | 6.37 MiB
          | 5.17 KiB          | 6.29 MiB          | 12.87 MiB       |
|       738 | reinvent@10.0.4.4                       |              1260 | 6.37 MiB
          | 5.18 KiB          | 6.29 MiB          | 13.00 MiB       |
|       819 | reinvent@10.0.4.4                       |              1252 | 6.37 MiB
          | 5.21 KiB          | 6.29 MiB          | 13.01 MiB       |
|        31 | innodb/srv_purge_thread                 |             17810 | 3.14 MiB
          |  184 bytes        | 2.40 MiB          | 205.69 MiB      |
```

```
|        38 | rdsadmin@localhost                          |                599 | 1.76 MiB
          | 3.01 KiB          | 1.00 MiB          | 25.58 MiB       |
|         1 | sql/main                                    |               3756 | 1.32 MiB
          |  367 bytes        | 355.78 KiB        | 6.19 MiB        |
|       854 | rdsadmin@localhost                          |                 46 | 1.08 MiB
          | 23.98 KiB         | 1.00 MiB          | 5.10 MiB        |
|        30 | innodb/clone_gtid_thread                    |               1596 | 573.14
 KiB      |  367 bytes        | 254.91 KiB        | 970.69 KiB      |
|        40 | rdsadmin@localhost                          |                235 | 245.19
 KiB      | 1.04 KiB          | 128.88 KiB        | 808.64 KiB      |
|       853 | rdsadmin@localhost                          |                 96 | 94.63
 KiB      | 1009 bytes        | 29.73 KiB         | 422.45 KiB      |
|        36 | rdsadmin@localhost                          |                 33 | 36.29
 KiB      | 1.10 KiB          | 16.08 KiB         | 74.15 MiB       |
|        33 | sql/event_scheduler                         |                  3 | 16.27
 KiB      | 5.42 KiB          | 16.04 KiB         | 16.27 KiB       |
|        35 | sql/compress_gtid_table                     |                  8 | 14.20
 KiB      | 1.77 KiB          | 8.05 KiB          | 18.62 KiB       |
|        25 | innodb/fts_optimize_thread                  |                 12 | 1.86 KiB
          |  158 bytes        |  648 bytes        | 1.98 KiB        |
|        23 | innodb/srv_master_thread                    |                 11 | 1.23 KiB
          |  114 bytes        |  361 bytes        | 24.40 KiB       |
|        24 | innodb/dict_stats_thread                    |                 11 | 1.23 KiB
          |  114 bytes        |  361 bytes        | 1.35 KiB        |
|         5 | innodb/io_read_thread                       |                  1 |  144
 bytes    |  144 bytes        |  144 bytes        |  144 bytes      |
|         6 | innodb/io_read_thread                       |                  1 |  144
 bytes    |  144 bytes        |  144 bytes        |  144 bytes      |
|         2 | sql/aws_oscar_log_level_monitor             |                  0 |    0
 bytes    |    0 bytes        |    0 bytes        |    0 bytes      |
|         4 | innodb/io_ibuf_thread                       |                  0 |    0
 bytes    |    0 bytes        |    0 bytes        |    0 bytes      |
|         7 | innodb/io_write_thread                      |                  0 |    0
 bytes    |    0 bytes        |    0 bytes        |    0 bytes      |
|         8 | innodb/io_write_thread                      |                  0 |    0
 bytes    |    0 bytes        |    0 bytes        |    0 bytes      |
|         9 | innodb/io_write_thread                      |                  0 |    0
 bytes    |    0 bytes        |    0 bytes        |    0 bytes      |
|        10 | innodb/io_write_thread                      |                  0 |    0
 bytes    |    0 bytes        |    0 bytes        |    0 bytes      |
|        11 | innodb/srv_lra_thread                       |                  0 |    0
 bytes    |    0 bytes        |    0 bytes        |    0 bytes      |
|        12 | innodb/srv_akp_thread                       |                  0 |    0
 bytes    |    0 bytes        |    0 bytes        |    0 bytes      |
```

```
|        18 | innodb/srv_lock_timeout_thread            |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |  248 bytes      |
|        19 | innodb/srv_error_monitor_thread           |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
|        20 | innodb/srv_monitor_thread                 |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
|        21 | innodb/buf_resize_thread                  |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
|        22 | innodb/btr_search_sys_toggle_thread       |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
|        32 | innodb/dict_persist_metadata_table_thread |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
|        34 | sql/signal_handler                        |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
+-----------+-------------------------------------------+--------------------
+-------------------+------------------+------------------+-----------------+
831 rows in set (2.48 sec)
```

如前所述，此处的线程 ID (thd_id) 值可以指服务器后台线程或数据库连接。如果要将线程 ID 值映射到数据库连接 ID，则可以使用 performance_schema.threads 表或 sys.processlist 视图，其中 conn_id 是连接 ID。

```
mysql> SELECT thd_id,conn_id,user,db,command,state,time,last_wait FROM sys.processlist
 WHERE user='reinvent@10.0.4.4';

+--------+---------+------------------+----------+---------+----------------+------
+-------------------------------------------------+
| thd_id | conn_id | user             | db       | command | state          | time |
 last_wait                                        |
+--------+---------+------------------+----------+---------+----------------+------
+-------------------------------------------------+
|    590 |     562 | reinvent@10.0.4.4 | sysbench | Execute | closing tables |    0 |
 wait/io/redo_log_flush                           |
|    578 |     550 | reinvent@10.0.4.4 | sysbench | Sleep   | NULL           |    0 |
 idle                                             |
|    579 |     551 | reinvent@10.0.4.4 | sysbench | Execute | closing tables |    0 |
 wait/io/redo_log_flush                           |
|    580 |     552 | reinvent@10.0.4.4 | sysbench | Execute | updating       |    0 |
 wait/io/table/sql/handler                        |
|    581 |     553 | reinvent@10.0.4.4 | sysbench | Execute | updating       |    0 |
 wait/io/table/sql/handler                        |
|    582 |     554 | reinvent@10.0.4.4 | sysbench | Sleep   | NULL           |    0 |
 idle                                             |
```

```
|    583 |     555 | reinvent@10.0.4.4 | sysbench | Sleep   | NULL           |    0 |
 idle                                          |
|    584 |     556 | reinvent@10.0.4.4 | sysbench | Execute | updating       |    0 |
 wait/io/table/sql/handler                     |
|    585 |     557 | reinvent@10.0.4.4 | sysbench | Execute | closing tables |    0 |
 wait/io/redo_log_flush                        |
|    586 |     558 | reinvent@10.0.4.4 | sysbench | Execute | updating       |    0 |
 wait/io/table/sql/handler                     |
|    587 |     559 | reinvent@10.0.4.4 | sysbench | Execute | closing tables |    0 |
 wait/io/redo_log_flush                        |
.
                                          .
.
                                          .
.
                                          .
|    323 |     295 | reinvent@10.0.4.4 | sysbench | Sleep   | NULL           |    0 |
 idle                                          |
|    324 |     296 | reinvent@10.0.4.4 | sysbench | Execute | updating       |    0 |
 wait/io/table/sql/handler                     |
|    325 |     297 | reinvent@10.0.4.4 | sysbench | Execute | closing tables |    0 |
 wait/io/redo_log_flush                        |
|    326 |     298 | reinvent@10.0.4.4 | sysbench | Execute | updating       |    0 |
 wait/io/table/sql/handler                     |
|    438 |     410 | reinvent@10.0.4.4 | sysbench | Execute | System lock    |    0 |
 wait/lock/table/sql/handler                   |
|    280 |     252 | reinvent@10.0.4.4 | sysbench | Sleep   | starting       |    0 |
 wait/io/socket/sql/client_connection          |
|     98 |      70 | reinvent@10.0.4.4 | sysbench | Query   | freeing items  |    0 |
 NULL                                          |
+--------+---------+-------------------+----------+---------+----------------+------
+-----------------------------------------------+
804 rows in set (5.51 sec)
```

现在我们停止 sysbench 工作负载，这会关闭连接并释放内存。再次检查事件，我们可以确认内存已释放，但 high_alloc 仍然可以告诉我们历史最高水平是多少。high_alloc 列在识别内存使用量的短暂峰值时可能非常有用，在这种情况下，您可能无法根据 current_alloc 立即识别使用量，它仅显示当前分配的内存。

```
mysql> SELECT * FROM sys.memory_global_by_current_bytes WHERE event_name='memory/sql/
Prepared_statement::main_mem_root' LIMIT 10;
```

```
+-----------------------------------------------+---------------+---------------
+------------------+-----------+-----------+----------------+
| event_name                                    | current_count | current_alloc |
 current_avg_alloc | high_count | high_alloc | high_avg_alloc |
+-----------------------------------------------+---------------+---------------
+------------------+-----------+-----------+----------------+
| memory/sql/Prepared_statement::main_mem_root |            17 | 253.80 KiB    | 14.93
 KiB         |     512823 | 4.91 GiB   | 10.04 KiB      |
+-----------------------------------------------+---------------+---------------
+------------------+-----------+-----------+----------------+
1 row in set (0.00 sec)
```

如果要重置 high_alloc，可以截断 performance_schema 内存摘要表，但这会重置所有内存检测。有关更多信息，请参阅 MySQL 文档中的 Performance Schema general table characteristics。

在下面的示例中，我们可以看到截断后重置 high_alloc。

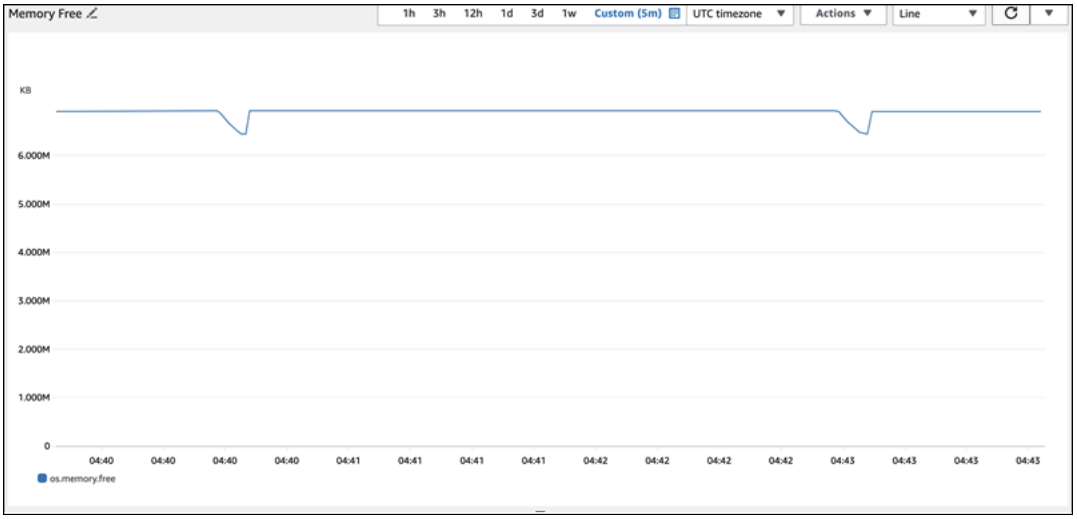
```
mysql> TRUNCATE `performance_schema`.`memory_summary_global_by_event_name`;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM sys.memory_global_by_current_bytes WHERE event_name='memory/sql/
Prepared_statement::main_mem_root' LIMIT 10;

+-----------------------------------------------+---------------+---------------
+------------------+-----------+-----------+----------------+
| event_name                                    | current_count | current_alloc |
 current_avg_alloc | high_count | high_alloc | high_avg_alloc |
+-----------------------------------------------+---------------+---------------
+------------------+-----------+-----------+----------------+
| memory/sql/Prepared_statement::main_mem_root |            17 | 253.80 KiB    | 14.93
 KiB         |         17 | 253.80 KiB | 14.93 KiB      |
+-----------------------------------------------+---------------+---------------
+------------------+-----------+-----------+----------------+
1 row in set (0.00 sec)
```

示例 2：短暂内存峰值

另一种常见的情况是数据库服务器上的内存使用量出现短暂的峰值。这些可能是可释放内存的周期性下降，使用 sys.memory_global_by_current_bytes 中的 current_alloc 很难排除故障，因为内存已经被释放。

> **Note**
>
> 如果 Performance Schema 统计数据已重置，或者数据库实例已重启，则这些信息将无法在 sys 或 performance_schema 中找到。为保留该信息，建议您配置外部指标收集。

下图展示了增强监控中的 os.memory.free 指标，其中显示了内存使用量短暂的 7 秒峰值。增强监控让您能够以短至 1 秒的间隔进行监控，这非常适合捕捉此类短暂峰值。



为协助诊断此处内存使用量的原因，我们可以结合使用 sys 内存摘要视图中的 high_alloc 和 Performance Schema statement summary tables 来尝试识别有问题的会话和连接。

正如预期的那样，由于目前内存使用量不高，因此我们在 sys 架构视图的 current_alloc 下看不到任何严重问题。

```
mysql> SELECT * FROM sys.memory_global_by_current_bytes LIMIT 10;

+-------------------------------------------------------------------------------
+---------------+---------------+-------------------+------------+------------
+----------------+
| event_name                                                                    |
 current_count | current_alloc | current_avg_alloc | high_count | high_alloc |
 high_avg_alloc |
+-------------------------------------------------------------------------------
+---------------+---------------+-------------------+------------+------------
+----------------+
| memory/innodb/hash0hash                                                       |
     4 | 79.07 MiB     | 19.77 MiB         |          4 | 79.07 MiB  | 19.77 MiB      |
```

```
| memory/innodb/os0event                                                    |
 439372 | 60.34 MiB     |  144 bytes        |     439372 | 60.34 MiB  |  144 bytes
 |
| memory/performance_schema/events_statements_summary_by_digest             |
     1 | 40.28 MiB     | 40.28 MiB         |          1 | 40.28 MiB  | 40.28 MiB      |
| memory/mysys/KEY_CACHE                                                    |
     3 | 16.00 MiB     | 5.33 MiB          |          3 | 16.00 MiB  | 5.33 MiB       |
| memory/performance_schema/events_statements_history_long                  |
     1 | 14.34 MiB     | 14.34 MiB         |          1 | 14.34 MiB  | 14.34 MiB      |
| memory/performance_schema/events_errors_summary_by_thread_by_error        |
   257 | 13.07 MiB     | 52.06 KiB         |        257 | 13.07 MiB  | 52.06 KiB      |
| memory/performance_schema/events_statements_summary_by_thread_by_event_name |
     1 | 11.81 MiB     | 11.81 MiB         |          1 | 11.81 MiB  | 11.81 MiB      |
| memory/performance_schema/events_statements_summary_by_digest.digest_text  |
     1 | 9.77 MiB      | 9.77 MiB          |          1 | 9.77 MiB   | 9.77 MiB       |
| memory/performance_schema/events_statements_history_long.digest_text       |
     1 | 9.77 MiB      | 9.77 MiB          |          1 | 9.77 MiB   | 9.77 MiB       |
| memory/performance_schema/events_statements_history_long.sql_text          |
     1 | 9.77 MiB      | 9.77 MiB          |          1 | 9.77 MiB   | 9.77 MiB       |
+---------------------------------------------------------------------------
+---------------+---------------+------------------+------------+------------
+----------------+
10 rows in set (0.01 sec)
```

将视图扩展为按 high_alloc 排序，我们现在可以看到，此处 memory/temptable/physical_ram 组件是一个非常好的候选组件。最高时，它消耗了 515.00 MiB。

顾名思义，memory/temptable/physical_ram 在 MySQL 中检测 TEMP 存储引擎的内存使用量（在 MySQL 8.0 中引入）。有关 MySQL 如何使用临时表的更多信息，请参阅 MySQL 文档中的 Internal temporary table use in MySQL。

> **Note**
>
> 我们在该示例中使用 sys.x$memory_global_by_current_bytes 视图。

```
mysql> SELECT event_name, format_bytes(current_alloc) AS "currently allocated",
 sys.format_bytes(high_alloc) AS "high-water mark"
FROM sys.x$memory_global_by_current_bytes ORDER BY high_alloc DESC LIMIT 10;

+---------------------------------------------------------------------------
+--------------------+-----------------+
```

```
| event_name                                                                    |
 currently allocated | high-water mark |
+-------------------------------------------------------------------------------
+--------------------+----------------+
| memory/temptable/physical_ram                                                 | 4.00
 MiB            | 515.00 MiB      |
| memory/innodb/hash0hash                                                       | 79.07
 MiB           | 79.07 MiB       |
| memory/innodb/os0event                                                        | 63.95
 MiB           | 63.95 MiB       |
| memory/performance_schema/events_statements_summary_by_digest                | 40.28
 MiB           | 40.28 MiB       |
| memory/mysys/KEY_CACHE                                                        | 16.00
 MiB           | 16.00 MiB       |
| memory/performance_schema/events_statements_history_long                     | 14.34
 MiB           | 14.34 MiB       |
| memory/performance_schema/events_errors_summary_by_thread_by_error           | 13.07
 MiB           | 13.07 MiB       |
| memory/performance_schema/events_statements_summary_by_thread_by_event_name | 11.81
 MiB           | 11.81 MiB       |
| memory/performance_schema/events_statements_summary_by_digest.digest_text    | 9.77
 MiB            | 9.77 MiB        |
| memory/performance_schema/events_statements_history_long.sql_text            | 9.77
 MiB            | 9.77 MiB        |
+-------------------------------------------------------------------------------
+--------------------+----------------+
10 rows in set (0.00 sec)
```

在示例 1：持续的高内存使用量中，我们检查了每个连接的当前内存使用量，来确定哪个连接负责使用相关内存。在此示例中，内存已被释放，因此检查当前连接的内存使用量并无用处。

为了更深入地挖掘并找到有问题的语句、用户和主机，我们使用 Performance Schema。Performance Schema 包含多个语句摘要表，这些摘要表按事件名称、语句摘要、主机、线程和用户等不同维度进行切片。每个视图都让您能够更深入地了解某些语句的运行位置以及它们的作用是什么。本节重点介绍 MAX_TOTAL_MEMORY，但您可以在 Performance Schema statement summary tables 文档中找到有关所有可用列的更多信息。

```
mysql> SHOW TABLES IN performance_schema LIKE 'events_statements_summary_%';

+-----------------------------------------------------------+
| Tables_in_performance_schema (events_statements_summary_%) |
+-----------------------------------------------------------+
| events_statements_summary_by_account_by_event_name         |
```

```
| events_statements_summary_by_digest                          |
| events_statements_summary_by_host_by_event_name              |
| events_statements_summary_by_program                         |
| events_statements_summary_by_thread_by_event_name            |
| events_statements_summary_by_user_by_event_name              |
| events_statements_summary_global_by_event_name               |
+--------------------------------------------------------------+
7 rows in set (0.00 sec)
```

首先，我们检查 events_statements_summary_by_digest 来查看 MAX_TOTAL_MEMORY。

从这里，我们可以看到以下内容：

- 带有摘要 20676ce4a690592ff05debcffcbc26faeb76f22005e7628364d7a498769d0c4a 的查询似乎是这种内存使用量的一个很好的候选查询。MAX_TOTAL_MEMORY 为 537450710，与我们在 sys.x$memory_global_by_current_bytes 中看到的 memory/temptable/physical_ram 事件的历史最高水平相符。
- 它已经运行了四次 (COUNT_STAR)，第一次是在 2024-03-26 04:08:34.943256，最后一次是在 2024-03-26 04:43:06.998310。

```
mysql> SELECT SCHEMA_NAME,DIGEST,COUNT_STAR,MAX_TOTAL_MEMORY,FIRST_SEEN,LAST_SEEN
FROM performance_schema.events_statements_summary_by_digest ORDER BY MAX_TOTAL_MEMORY
 DESC LIMIT 5;

+-------------+----------------------------------------------------------------
+------------+------------------+---------------------------
+----------------------------+
| SCHEMA_NAME | DIGEST                                                          |
 COUNT_STAR | MAX_TOTAL_MEMORY | FIRST_SEEN                 | LAST_SEEN
  |
+-------------+----------------------------------------------------------------
+------------+------------------+---------------------------
+----------------------------+
| sysbench    | 20676ce4a690592ff05debcffcbc26faeb76f22005e7628364d7a498769d0c4a |
     4 |        537450710 | 2024-03-26 04:08:34.943256 | 2024-03-26 04:43:06.998310 |
| NULL        | f158282ea0313fefd0a4778f6e9b92fc7d1e839af59ebd8c5eea35e12732c45d |
     4 |          3636413 | 2024-03-26 04:29:32.712348 | 2024-03-26 04:36:26.269329 |
| NULL        | 0046bc5f642c586b8a9afd6ce1ab70612dc5b1fd2408fa8677f370c1b0ca3213 |
     2 |          3459965 | 2024-03-26 04:31:37.674008 | 2024-03-26 04:32:09.410718 |
| NULL        | 8924f01bba3c55324701716c7b50071a60b9ceaf17108c71fd064c20c4ab14db |
     1 |          3290981 | 2024-03-26 04:31:49.751506 | 2024-03-26 04:31:49.751506 |
```

```
| NULL        | 90142bbcb50a744fcec03a1aa336b2169761597ea06d85c7f6ab03b5a4e1d841 |
     1 |          3131729 | 2024-03-26 04:15:09.719557 | 2024-03-26 04:15:09.719557 |
+-------------+-----------------------------------------------------------------
+-----------+-----------------+--------------------------
+---------------------------+
5 rows in set (0.00 sec)
```

现在我们知道了有问题的摘要，我们可以获得更多详细信息，例如查询文本、运行它的用户以及运行它的位置。根据返回的摘要文本，我们可以看到这是一个公用表表达式（CTE），它创建了四个临时表并执行了四次表扫描，效率非常低。

```
mysql> SELECT
 SCHEMA_NAME,DIGEST_TEXT,QUERY_SAMPLE_TEXT,MAX_TOTAL_MEMORY,SUM_ROWS_SENT,SUM_ROWS_EXAMINED,SUM
FROM performance_schema.events_statements_summary_by_digest
WHERE DIGEST='20676ce4a690592ff05debcffcbc26faeb76f22005e7628364d7a498769d0c4a'\G;

*************************** 1. row ***************************
           SCHEMA_NAME: sysbench
           DIGEST_TEXT: WITH RECURSIVE `cte` ( `n` ) AS ( SELECT ? FROM `sbtest1` UNION
 ALL SELECT `id` + ? FROM `sbtest1` ) SELECT * FROM `cte`
     QUERY_SAMPLE_TEXT: WITH RECURSIVE cte (n) AS (   SELECT 1  from sbtest1 UNION ALL
  SELECT id + 1 FROM sbtest1) SELECT * FROM cte
      MAX_TOTAL_MEMORY: 537450710
         SUM_ROWS_SENT: 80000000
     SUM_ROWS_EXAMINED: 80000000
SUM_CREATED_TMP_TABLES: 4
     SUM_NO_INDEX_USED: 4
1 row in set (0.01 sec)
```

有关 `events_statements_summary_by_digest` 表和其它 Performance Schema 语句摘要表的更多信息，请参阅 MySQL 文档中的 Statement summary tables。

您也可以运行 EXPLAIN 或 EXPLAIN ANALYZE 语句来查看更多详细信息。

> **Note**
>
> EXPLAIN ANALYZE 可以比 EXPLAIN 提供更多的信息，但它也会运行查询，所以要小心。

```
-- EXPLAIN
mysql> EXPLAIN WITH RECURSIVE cte (n) AS (SELECT 1  FROM sbtest1 UNION ALL SELECT id +
 1 FROM sbtest1) SELECT * FROM cte;
```

```
+----+-------------+-----------+-----------+-------+---------------+------+---------
+------+----------+----------+-------------+
| id | select_type | table      | partitions | type  | possible_keys | key  | key_len |
 ref  | rows     | filtered | Extra       |
+----+-------------+-----------+-----------+-------+---------------+------+---------
+------+----------+----------+-------------+
|  1 | PRIMARY     | <derived2> | NULL       | ALL   | NULL          | NULL | NULL    |
 NULL | 19221520 |   100.00 | NULL        |
|  2 | DERIVED     | sbtest1    | NULL       | index | NULL          | k_1  | 4       |
 NULL |  9610760 |   100.00 | Using index |
|  3 | UNION       | sbtest1    | NULL       | index | NULL          | k_1  | 4       |
 NULL |  9610760 |   100.00 | Using index |
+----+-------------+-----------+-----------+-------+---------------+------+---------
+------+----------+----------+-------------+
3 rows in set, 1 warning (0.00 sec)

-- EXPLAIN format=tree
mysql> EXPLAIN format=tree WITH RECURSIVE cte (n) AS (SELECT 1 FROM sbtest1 UNION ALL
 SELECT id + 1 FROM sbtest1) SELECT * FROM cte\G;

*************************** 1. row ***************************
EXPLAIN: -> Table scan on cte  (cost=4.11e+6..4.35e+6 rows=19.2e+6)
    -> Materialize union CTE cte  (cost=4.11e+6..4.11e+6 rows=19.2e+6)
        -> Index scan on sbtest1 using k_1  (cost=1.09e+6 rows=9.61e+6)
        -> Index scan on sbtest1 using k_1  (cost=1.09e+6 rows=9.61e+6)
1 row in set (0.00 sec)

-- EXPLAIN ANALYZE
mysql> EXPLAIN ANALYZE WITH RECURSIVE cte (n) AS (SELECT 1 from sbtest1 UNION ALL
 SELECT id + 1 FROM sbtest1) SELECT * FROM cte\G;

*************************** 1. row ***************************
EXPLAIN: -> Table scan on cte  (cost=4.11e+6..4.35e+6 rows=19.2e+6) (actual
 time=6666..9201 rows=20e+6 loops=1)
    -> Materialize union CTE cte  (cost=4.11e+6..4.11e+6 rows=19.2e+6) (actual
 time=6666..6666 rows=20e+6 loops=1)
        -> Covering index scan on sbtest1 using k_1  (cost=1.09e+6 rows=9.61e+6)
 (actual time=0.0365..2006 rows=10e+6 loops=1)
        -> Covering index scan on sbtest1 using k_1  (cost=1.09e+6 rows=9.61e+6)
 (actual time=0.0311..2494 rows=10e+6 loops=1)
1 row in set (10.53 sec)
```

但谁运行它呢？ 我们可以在 Performance Schema 中看到，destructive_operator 用户拥有 MAX_TOTAL_MEMORY 537450710，这再次与之前的结果相符。

> **Note**
>
> Performance Schema 存储在内存中，因此不应将其作为审计的唯一来源。如果您需要维护语句运行的历史记录以及语句是由哪些用户运行的，我们建议启用 Aurora 高级审计。如果您还需要维护有关内存使用量的信息，建议您将监控配置为导出和存储这些值。

```
mysql> SELECT USER,EVENT_NAME,COUNT_STAR,MAX_TOTAL_MEMORY FROM
 performance_schema.events_statements_summary_by_user_by_event_name
ORDER BY MAX_CONTROLLED_MEMORY DESC LIMIT 5;

+----------------------+---------------------------+------------+------------------+
| USER                 | EVENT_NAME                | COUNT_STAR | MAX_TOTAL_MEMORY |
+----------------------+---------------------------+------------+------------------+
| destructive_operator | statement/sql/select      |          4 |        537450710 |
| rdsadmin             | statement/sql/select      |       4172 |          3290981 |
| rdsadmin             | statement/sql/show_tables |          2 |          3615821 |
| rdsadmin             | statement/sql/show_fields |          2 |          3459965 |
| rdsadmin             | statement/sql/show_status |         75 |          1914976 |
+----------------------+---------------------------+------------+------------------+
5 rows in set (0.00 sec)

mysql> SELECT HOST,EVENT_NAME,COUNT_STAR,MAX_TOTAL_MEMORY FROM
 performance_schema.events_statements_summary_by_host_by_event_name
WHERE HOST != 'localhost' AND COUNT_STAR>0 ORDER BY MAX_CONTROLLED_MEMORY DESC LIMIT 5;

+------------+----------------------+------------+------------------+
| HOST       | EVENT_NAME           | COUNT_STAR | MAX_TOTAL_MEMORY |
+------------+----------------------+------------+------------------+
| 10.0.8.231 | statement/sql/select |          4 |        537450710 |
+------------+----------------------+------------+------------------+
1 row in set (0.00 sec)
```

示例 3：可释放内存持续下降且无法回收

InnoDB 数据库引擎为不同的组件使用了一系列专门的内存跟踪事件。这些特定事件允许精细跟踪 InnoDB 关键子系统的内存使用情况，例如：

- memory/innodb/buf0buf – 专门用于监控 InnoDB 缓冲池的内存分配。

- memory/innodb/ibuf0ibuf – 专门跟踪与 InnoDB 更改缓冲区相关的内存变化。

为了确定内存的最大使用者，我们可以查询 sys.memory_global_by_current_bytes：

```
mysql> SELECT event_name,current_alloc FROM sys.memory_global_by_current_bytes LIMIT
 10;

+-----------------------------------------------------------------+---------------+
| event_name                                                      | current_alloc |
+-----------------------------------------------------------------+---------------+
| memory/innodb/memory                                            | 5.28 GiB      |
| memory/performance_schema/table_io_waits_summary_by_index_usage | 495.00 MiB    |
| memory/performance_schema/table_shares                          | 488.00 MiB    |
| memory/sql/TABLE_SHARE::mem_root                                | 388.95 MiB    |
| memory/innodb/std                                               | 226.88 MiB    |
| memory/innodb/fil0fil                                           | 198.49 MiB    |
| memory/sql/binlog_io_cache                                      | 128.00 MiB    |
| memory/innodb/mem0mem                                           | 96.82 MiB     |
| memory/innodb/dict0dict                                         | 96.76 MiB     |
| memory/performance_schema/rwlock_instances                      | 88.00 MiB     |
+-----------------------------------------------------------------+---------------+
10 rows in set (0.00 sec)
```

结果表明，memory/innodb/memory 是最大使用者，使用了 5.28 GiB 当前分配的内存。此事件用作与更具体的等待事件（如前所述的 memory/innodb/buf0buf）无关的各种 InnoDB 组件的内存分配类别。

在确定了 InnoDB 组件是内存的主要使用者之后，我们就可以使用以下 MySQL 命令更深入地研究具体细节：

```
SHOW ENGINE INNODB STATUS \G;
```

SHOW ENGINE INNODB STATUS 命令为 InnoDB 存储引擎提供全面的状态报告，包括不同 InnoDB 组件的详细内存使用统计信息。它可以帮助确定哪些特定的 InnoDB 结构或操作消耗内存最多。有关更多信息，请参阅 MySQL 文档中的 InnoDB in-memory structures。

分析 InnoDB 状态报告的 BUFFER POOL AND MEMORY 部分后，我们可以看到，5,051,647,748 字节（4.7 GiB）分配给了字典对象缓存，占比为 memory/innodb/memory 所跟踪内存的 89%。

```
----------------------
BUFFER POOL AND MEMORY
```

```
----------------------
Total large memory allocated 0
Dictionary memory allocated 5051647748
Buffer pool size 170512
Free buffers 142568
Database pages 27944
Old database pages 10354
Modified db pages 6
Pending reads 0
```

字典对象缓存是一种共享全局缓存，它将以前访问过的数据字典对象存储在内存中，以实现对象的重复使用并提高性能。对字典对象缓存的高内存分配表明数据字典缓存中有大量数据库对象。

既然知道了数据字典缓存是主要使用者，我们接下来检查数据字典缓存中是否有打开的表。要查找表定义缓存中的表数量，请查询全局状态变量 open_table_definitions。

```
mysql> show global status like 'open_table_definitions';

+------------------------+-------+
| Variable_name          | Value |
+------------------------+-------+
| Open_table_definitions | 20000 |
+------------------------+-------+
1 row in set (0.00 sec)
```

有关更多信息，请参阅 MySQL 文档中的 MySQL 如何打开和关闭表。

您可以通过限制数据库集群或数据库实例参数组中的 table_definition_cache 参数，来限制数据字典缓存中表定义的数量。对于 Aurora MySQL，此值用作表定义缓存中表数量的软限制。默认值取决于实例类并设置为以下值：

```
LEAST({DBInstanceClassMemory/393040}, 20000)
```

当表的数量超过 table_definition_cache 限制时，最近最少使用（LRU）机制会将表从缓存中移出并删除。但是，涉及外键关系的表不会放在 LRU 列表中，以防止其被删除。

在我们当前的场景中，我们运行 FLUSH TABLES 来清除表定义缓存。此操作会导致 Open_table_definitions 全局状态变量从 20,000 大幅下降到 12，如下所示：

```
mysql> show global status like 'open_table_definitions';

+------------------------+-------+
```

```
| Variable_name          | Value |
+------------------------+-------+
| Open_table_definitions | 12    |
+------------------------+-------+
1 row in set (0.00 sec)
```

尽管有如此的减少，但我们观察到 memory/innodb/memory 的内存分配依然很高，为 5.18 GiB，且分配的字典内存也未变。从以下查询结果中可以明显看出这一点：

```
mysql> SELECT event_name,current_alloc FROM sys.memory_global_by_current_bytes LIMIT
 10;

+-----------------------------------------------------------------+---------------+
| event_name                                                      | current_alloc |
+-----------------------------------------------------------------+---------------+
| memory/innodb/memory                                            | 5.18 GiB      |
| memory/performance_schema/table_io_waits_summary_by_index_usage | 495.00 MiB    |
| memory/performance_schema/table_shares                          | 488.00 MiB    |
| memory/sql/TABLE_SHARE::mem_root                                | 388.95 MiB    |
| memory/innodb/std                                               | 226.88 MiB    |
| memory/innodb/fil0fil                                           | 198.49 MiB    |
| memory/sql/binlog_io_cache                                      | 128.00 MiB    |
| memory/innodb/mem0mem                                           | 96.82 MiB     |
| memory/innodb/dict0dict                                         | 96.76 MiB     |
| memory/performance_schema/rwlock_instances                      | 88.00 MiB     |
+-----------------------------------------------------------------+---------------+
10 rows in set (0.00 sec)
```

```
----------------------
BUFFER POOL AND MEMORY
----------------------
Total large memory allocated 0
Dictionary memory allocated 5001599639
Buffer pool size 170512
Free buffers 142568
Database pages 27944
Old database pages 10354
Modified db pages 6
Pending reads 0
```

这种持续的高内存使用量可以归因于涉及外键关系的表。这些表没有放在 LRU 列表中，无法删除，这就解释了为什么即使在刷新了表定义缓存之后，内存分配依然很高。

解决此问题：

1. 查看和优化您的数据库架构，尤其是外键关系。
2. 可以考虑改用具有更大内存的更大的数据库实例类来容纳您的字典对象。

通过执行这些步骤并了解内存分配模式，您可以更好地管理 Aurora MySQL 数据库实例中的内存使用情况，并防止由于内存压力而导致的潜在性能问题。

排查 Aurora MySQL 数据库内存不足问题

Aurora MySQL `aurora_oom_response` 实例级参数可以使数据库实例能够监控系统内存，并估计各种语句和连接消耗的内存。如果系统内存不足，它可能会执行一系列操作，来尝试释放该内存。这样做的目的是试图避免由于内存不足（OOM）问题而导致数据库重启。该实例级参数使用一串以逗号分隔的操作，在内存不足时，数据库实例将执行这些操作。Aurora MySQL 版本 2 和 3 支持 `aurora_oom_response` 参数。

以下值及其组合可用于 `aurora_oom_response` 参数。空字符串表示未执行任何操作，实际上是关闭此功能，使数据库易于 OOM 重启。

- `decline` – 在数据库实例出现内存不足时，拒绝新的查询。
- `kill_connect` – 关闭消耗大量内存的数据库连接，并结束当前事务和数据定义语言（DDL）语句。Aurora MySQL 版本 2 不支持此响应。

  有关更多信息，请参阅 MySQL 文档中的 KILL statement。
- `kill_query` – 按内存消耗量降序结束查询，直到实例内存高于下限阈值。未结束 DDL 语句。

  有关更多信息，请参阅 MySQL 文档中的 KILL statement。
- `print` – 仅输出占用大量内存的查询。
- `tune` – 调整内部表缓存以将一些内存释放回系统。在内存不足的情况下，Aurora MySQL 会减少用于缓存（例如 `table_open_cache` 和 `table_definition_cache`）的内存。最终，当系统不再内存不足时，Aurora MySQL 会将其内存使用率恢复正常。

  有关更多信息，请参阅 MySQL 文档中的 table_open_cache 和 table_definition_cache。
- `tune_buffer_pool` – 减小缓冲池的大小以释放一些内存，使数据库服务器可以使用这些内存来处理连接。Aurora MySQL 版本 3.06 及更高版本支持此响应。

您必须在 aurora_oom_response 参数值中将 tune_buffer_pool 或 kill_query 与 kill_connect 进行配对。否则，即使您在参数值中包含 tune_buffer_pool，也无法调整缓冲池的大小。

在低于 3.06 的 Aurora MySQL 版本中，对于内存小于或等于 4GiB 的数据库实例类，当实例面临内存压力时，默认操作包括 print、tune、decline 和 kill_query。对于内存大于 4GiB 的数据库实例类，该参数值默认为空（已禁用）。

在 Aurora MySQL 版本 3.06 及更高版本中，对于内存小于或等于 4 GiB 的数据库实例类，Aurora MySQL 还会关闭占用内存最高的连接（kill_connect）。对于内存大于 4 GiB 的数据库实例类，默认参数值为 print。

在 Aurora MySQL 版本 3.09 及更高版本中，对于内存大于 4 GiB 的数据库实例类，默认参数值为 print,decline,kill_connect。

如果您经常遇到内存不足问题，则启用 performance_schema 后，可以使用内存摘要表来监控内存使用情况。

有关与 OOM 相关的 Amazon CloudWatch 指标，请参阅 Amazon Aurora 的实例级指标。有关与 OOM 相关的全局状态变量，请参阅 Aurora MySQL 全局状态变量。

Aurora MySQL 数据库日志记录

Aurora MySQL 日志提供有关数据库活动和错误的基本信息。通过启用这些日志，您可以识别和排查问题，了解数据库性能并审计数据库活动。我们建议您为所有 Aurora MySQL 数据库实例启用这些日志，来确保数据库的最佳性能和可用性。可以启用以下类型的日志记录。每个日志都包含特定的信息，这些信息可以揭示对数据库处理的影响。

- 错误 – Aurora MySQL 仅在启动、关闭和遇到错误时，才向错误日志写入内容。数据库实例可以运行数小时或者数天，而不向错误日志中写入新项。如果看不到最近的条目，则原因是服务器未遇到导致生成日志条目的错误。默认情况下，启用错误日志记录。有关更多信息，请参阅 Aurora MySQL 错误日志。
- 常规 - 常规日志提供有关数据库活动的详细信息，包括数据库引擎执行的所有 SQL 语句。有关启用常规日志记录和设置日志记录参数的更多信息，请参阅 Aurora MySQL 慢速查询日志和一般日志和 MySQL 文档中的 The general query log。

  > **Note**
  >
  > 常规日志可能会增长到非常大，并占用您的存储空间。有关更多信息，请参阅 Aurora MySQL 的日志轮换和保留。

- 慢速查询 – 慢速查询日志包含运行时间超过 long_query_time 秒且需要至少检查 min_examined_row_limit 行的 SQL 语句。您可以使用慢速查询日志来查找运行时间长、因而适合进行优化的查询。

  long_query_time 的默认值是 10 秒。我们建议您从较高的值开始来确定最慢的查询，然后逐渐向下进行微调。

  您也可以使用相关参数，例如 log_slow_admin_statements 和 log_queries_not_using_indexes。将 rows_examined 与 rows_returned 进行比较。如果 rows_examined 比 rows_returned 大得多，则这些查询可能会被阻止。

  在 Aurora MySQL 版本 3 中，您可以启用 log_slow_extra 来获取更多详细信息。有关更多信息，请参阅 MySQL 文档中的 Slow query log contents。您还可以在会话级别修改 long_query_time，以便以交互方式调试查询执行，这在全局启用 log_slow_extra 时尤其有用。

  有关启用慢速查询日志记录和设置日志记录参数的更多信息，请参阅 Aurora MySQL 慢速查询日志和一般日志和 MySQL 文档中的 The slow query log。

- 审计 - 审计日志监控并记录数据库活动。Aurora MySQL 的审计日志记录称为高级审计。要启用高级审计，应设置某些数据库集群参数。有关更多信息，请参阅 在 Amazon Aurora MySQL 数据库集群中使用高级审计。
- 二进制 – 二进制日志（binlog）包含描述数据库更改的事件，例如表创建操作和对表数据的更改。它还包含可能已进行更改的语句的事件（例如，不匹配任何行的 DELETE），除非使用基于行的日志记录。二进制日志还包含有关更新数据的每条语句所花时间的信息。

  在启用了二进制日志记录的情况下运行服务器会使性能稍微下降。但是，二进制日志使您能够设置复制和执行还原操作，所带来的好处通常会超过这一轻微的性能下降。

  > **Note**
  >
  > Aurora MySQL 不需要使用二进制日志记录来执行还原操作。

  有关启用二进制日志记录和设置二进制日志格式的更多信息，请参阅为单可用区数据库配置 Aurora MySQL 二进制日志记录以及 MySQL 文档中的 The binary log。

您可以将错误日志、常规日志、慢速日志、查询日志和审计日志发布到 Amazon CloudWatch Logs。有关更多信息，请参阅 将数据库日志发布到 Amazon CloudWatch Logs。

另一个用于汇总慢速日志文件、常规日志文件和二进制日志文件的有用工具是 pt-query-digest。

排查 Aurora MySQL 数据库的连接问题

确保应用程序和 RDS 数据库实例之间的可靠连接对于工作负载的平稳运行至关重要。但是，各种因素（例如网络配置、身份验证问题或资源限制）可能会导致出现连接问题。本指南旨在提供一种全面的方法来排查 Aurora MySQL 的连接问题。

目录

- 识别 Aurora MySQL 的数据库连接问题
- 收集有关 Aurora MySQL 连接问题的数据
- 监控 Aurora MySQL 的数据库连接
  - Aurora MySQL 的额外监控
- Aurora MySQL 的连接错误代码
- Aurora MySQL 的参数调优建议
- 排查 Aurora MySQL 数据库连接问题的示例
  - 示例 1：排查失败的连接尝试
  - 示例 2：排查异常客户端断开连接问题

识别 Aurora MySQL 的数据库连接问题

识别连接问题的具体类别有助于缩小潜在原因范围并指导问题排查过程。每个类别可能需要不同的诊断和解决方法与技术。数据库连接问题可以大致分为以下几类。

连接错误和异常

出现连接错误和异常的原因有多种，例如连接字符串不正确、身份验证失败、网络中断或数据库服务器问题。原因可能包括连接参数配置错误、凭证无效、网络中断或数据库服务器崩溃或重新启动。安全组配置错误、虚拟私有云（VPC）设置、网络访问控制列表（ACL）以及与子网关联的路由表也可能导致连接问题。

已达到连接限制

当数据库服务器的并发连接数超过允许的最大限制时，就会出现此问题。数据库服务器通常具有可配置的最大连接限制，该限制由集群和实例参数组中的参数 max_connections 定义。通过施加连接限制，数据库服务器可确保拥有足够的资源（例如内存、CPU 和文件句柄）来有效地处理现有连接并提供可接受的性能。原因可能包括应用程序中的连接泄漏、连接池效率低下或连接请求意外激增。

连接超时

当客户端应用程序无法在指定的超时期间与数据库服务器建立连接时，就会发生连接超时。常见原因包括网络问题、服务器过载、防火墙规则和连接设置配置错误。

空闲连接超时

为了节省资源，数据库服务器可能会自动关闭长时间处于非活动状态的空闲连接。此超时通常可使用 `wait_timeout` 和 `interactive_timeout parameters` 进行配置，并且应根据应用程序的连接使用模式进行调整。原因可能包括应用程序逻辑使连接长时间处于空闲状态，或者连接管理不当。

间歇性断开现有连接

此类错误是指这样一种情况：尽管客户端应用程序和数据库之间已建立的连接处于活动状态并正在使用中，但仍意外终止或不定期断开连接。这些断开连接是间歇性发生的，意味着它们以不规则的间隔发生，而且不一致。原因可以包括：

- 数据库服务器问题，例如重新启动或失效转移
- 应用程序连接处理不当
- 负载均衡和代理问题
- 网络不稳定
- 连接路径中涉及第三方组件或中间件的问题
- 查询执行超时
- 服务器端或客户端的资源限制

通过全面的监控、日志记录和分析来确定根本原因至关重要，而实施适当的错误处理、连接池和重试机制有助于减轻这些间歇性断开连接对应用程序功能和用户体验的影响。

收集有关 Aurora MySQL 连接问题的数据

收集与应用程序、数据库、网络和基础设施组件相关的全面数据对于有效排查应用程序与 Aurora MySQL 数据库之间的连接问题至关重要。通过收集相关的日志、配置和诊断信息，可以获得宝贵的见解，这些见解能够帮助您确定连接问题的根本原因并指导您找到适当的解决方案。

网络日志和配置（例如安全组规则、VPC 设置和路由表）对于识别可能阻碍应用程序成功与数据库建立连接的潜在网络相关瓶颈或错误配置至关重要。通过分析这些网络组件，您可以确保已打开必要的端口，已允许 IP 地址，并已正确设置路由配置。

时间戳

记录发生连接问题时的确切时间戳。这有助于识别模式或将问题与其他事件或活动相关联。

数据库引擎日志

除了常规数据库日志外，还要查看数据库引擎日志（例如，MySQL 错误日志和慢速查询日志），以了解可能与间歇性连接问题相关的任何相关信息或错误。有关更多信息，请参阅 Aurora MySQL 数据库日志记录。

客户端应用程序日志

从连接到数据库的客户端应用程序收集详细日志。应用程序日志从应用程序的角度提供有关连接尝试和错误的信息以及任何相关信息，从而揭示与连接字符串、身份验证凭证或应用程序级连接处理有关的问题。

而数据库日志让您可以深入了解数据库端错误、慢速查询或可能导致连接问题的事件。有关更多信息，请参阅 Aurora MySQL 数据库日志记录。

客户端环境变量

检查客户端上的任何环境变量或配置设置是否可能影响数据库连接，例如代理设置、SSL/TLS 设置或任何其他相关变量。

客户端库版本

确保客户端使用的是用于数据库连接的所有数据库驱动程序、库或框架的最新版本。过时的版本可能存在已知问题或兼容性问题。

客户端网络捕获

出现连接问题时，使用诸如 Wireshark 或 `tcpdump` 之类的工具在客户端执行网络捕获。这有助于在客户端识别任何网络相关问题或异常。

客户端网络拓扑

了解客户端的网络拓扑，包括任何防火墙、负载均衡器或其他组件，例如 RDS 代理或 Proxy SQL，这些组件与数据库建立连接，而不是客户端直接建立连接。

客户端操作系统设置

检查可能影响网络连接的客户端操作系统设置，例如防火墙规则、网络适配器设置以及任何其他相关设置。

连接池配置

如果您在应用程序中使用连接池机制，请查看配置设置并监控池指标（例如，活动连接、空闲连接和连接超时），以确保池正常运行。还要查看池设置（例如最大池大小、最小池大小和连接验证设置）以确保配置正确。

连接字符串

连接字符串通常包括诸如主机名或端点、端口号、数据库名称和身份验证凭证之类的参数。分析连接字符串有助于识别可能导致连接问题的潜在错误配置或不正确设置。例如，错误的主机名或端口号可能会阻止客户端访问数据库实例，而无效的身份验证凭证可能导致身份验证失败和连接被拒绝。此外，连接字符串可以揭示与连接池、超时或其他特定于连接的设置（可能导致连接问题）有关的问题。提供客户端应用程序使用的完整连接字符串有助于查明客户端上的任何错误配置。

数据库指标

出现连接问题时，监控 CPU 使用率、内存使用量和磁盘 I/O 等数据库指标。这些有助于确定数据库实例是否存在资源争用或性能问题。

数据库引擎版本

请注意 Aurora MySQL 数据库引擎版本。AWS 定期发布更新以解决已知问题和安全漏洞，并引入性能增强功能。因此，强烈建议您升级到最新的可用版本，因为这些更新通常包含与连接性、性能和稳定性特别相关的错误修复和改进。提供数据库版本信息以及收集的其他详细信息可以帮助 支持有效地诊断和解决连接问题。

网络指标

出现连接问题时，收集延迟、数据包丢失和吞吐量等网络指标。`ping` 和 `traceroute` 等工具以及网络监控工具可以帮助收集这些数据。

源和客户端详细信息

确定正在启动数据库连接的应用程序服务器、负载均衡器或任何其他组件的 IP 地址。这可以是单个 IP 地址，也可以是 IP 地址范围（CIDR 表示法）。如果源是 Amazon EC2 实例，则查看实例类型、可用区、子网 ID 和与该实例关联的安全组以及网络接口详细信息（例如私有 IP 地址和公有 IP 地址）也会有所帮助。

通过全面分析所收集的数据，您可以识别导致间歇性或持续连接问题的错误配置、资源限制、网络中断或其他潜在问题。您可以根据这些信息采取有针对性的措施，例如调整配置、解决网络问题或进行应用程序级连接处理。

监控 Aurora MySQL 的数据库连接

要监控和排查连接问题，可以使用以下指标和功能。

CloudWatch 指标

- `CPUUtilization` – 如果数据库实例上的 CPU 使用率过高，可能会导致查询执行缓慢，从而导致连接超时或被拒绝。
- `DatabaseConnections` – 监控数据库实例的活动连接数。如果连接数接近配置的最大值，则表明可能存在连接问题或连接池已耗尽。
- `FreeableMemory` – 由于资源限制，可用内存不足可能会导致性能问题和连接问题。
- `NetworkReceiveThroughput` 和 `NetworkTransmitThroughput` – 网络吞吐量的异常峰值或下降可能表明存在连接问题或网络瓶颈。

性能详情指标

要利用性能详情排查 Aurora MySQL 中的连接问题，请分析如下所示的数据库指标：

- Aborted_clients
- Aborted_connects
- 连接
- max_connections
- Threads_connected
- Threads_created
- Threads_running

这些指标可帮助您识别连接瓶颈，检测网络或身份验证问题，优化连接池并确保高效的线程管理。有关更多信息，请参阅 Aurora MySQL 的性能详情计数器。

性能详情特征

- 数据库负载 – 可视化一段时间内的数据库负载，并将其与连接问题或性能下降相关联。
- SQL 统计数据 – 分析 SQL 统计数据，以识别可能导致连接问题的低效查询或数据库操作。
- 热门查询 – 识别和分析资源密集度高的查询，这有助于识别可能导致连接问题的潜在性能瓶颈或长时间运行的查询。

通过监控这些指标并利用性能详情，您可以了解数据库实例的性能、资源使用情况以及可能导致连接问题的潜在瓶颈。例如：

- 接近最大限制的高 DatabaseConnections 值可能表明连接池已耗尽或连接处理不当，从而导致连接问题。
- 高 CPUUtilization 或低 FreeableMemory 可能表明资源限制，这可能会导致查询执行缓慢以及连接超时或被拒绝。
- 分析热门查询和 SQL 统计数据有助于识别可能导致连接问题的低效查询或资源密集型查询。

此外，监控 CloudWatch Logs 和设置警报可帮助您在连接问题升级之前主动识别问题并做出响应。

值得注意的是，虽然这些指标和工具可以提供有价值的见解，但它们应与其他故障排除步骤结合使用。如果再查看网络配置、安全组规则和应用程序级连接处理，则可以全面诊断和解决 Aurora MySQL 数据库实例的连接问题。

Aurora MySQL 的额外监控

CloudWatch 指标

- AbortedClients – 跟踪未正确关闭的客户端连接数量。
- AuroraSlowConnectionHandleCount – 跟踪慢速连接句柄操作的数量，指出潜在的连接问题或性能瓶颈。
- AuroraSlowHandshakeCount – 测量慢速握手操作的数量，这也可以作为连接问题的指标。
- ConnectionAttempts – 测量尝试连接 Aurora MySQL 数据库实例的次数。

全局状态变量

Aurora_external_connection_count – 显示与数据库实例的数据库连接数，不包括用于数据库运行状况检查的 RDS 服务连接。

通过监控这些指标和全局状态变量，您可以了解可能导致 Amazon Aurora MySQL 实例出现连接问题的连接模式、错误和潜在瓶颈。

例如，AbortedClients 或 AuroraSlowConnectionHandleCount 数量过多可能表明存在连接问题。

此外，设置 CloudWatch 警报和通知可帮助您在连接问题升级并影响应用程序性能之前主动识别问题并做出响应。

Aurora MySQL 的连接错误代码

以下是 Aurora MySQL 数据库的一些常见连接错误及其错误代码和说明。

错误代码 1040：连接过多

当客户端尝试建立的连接数超过数据库服务器允许的最大连接数时，就会发生此错误。可能的原因包括：

- 连接池配置错误 – 如果使用连接池机制，请确保最大池大小不要设置得太高，并且连接正确释放回池中。
- 数据库实例配置 – 验证数据库实例的最大允许连接数设置，并在必要时通过设置 `max_connections` 参数进行调整。
- 高并发 – 如果多个客户端或应用程序同时连接到数据库，则可能会达到允许的最大连接限制。

错误代码 1045：用户“...”@“...”的访问被拒绝（使用密码：是/否）

此错误表示尝试连接到数据库时身份验证失败。可能的原因包括：

- 身份验证插件兼容性 – 检查客户端使用的身份验证插件是否与数据库服务器的身份验证机制兼容。
- 用户名或密码不正确 – 验证连接字符串或身份验证机制中使用的用户名和密码是否正确。
- 用户权限 – 确保用户拥有从指定主机或网络连接到数据库实例所需的权限。

错误代码 1049：未知数据库“...”

此错误表示客户端正试图连接到服务器上不存在的数据库。可能的原因包括：

- 数据库未创建 – 确保数据库服务器上已创建指定的数据库。
- 数据库名称不正确 – 仔细检查连接字符串或查询中使用的数据库名称是否准确。
- 用户权限 – 验证用户是否拥有访问指定数据库所需的权限。

错误代码 1153：数据包大于“max_allowed_packet”字节

当客户端尝试发送或接收超出数据库服务器允许的最大数据包大小的数据时，就会发生此错误。可能的原因包括：

- 大型查询或结果集 – 如果执行的查询涉及大量数据，则可能会超过数据包大小限制。
- 错误配置的数据包大小设置 – 检查数据库服务器上的 `max_allowed_packet` 设置并在必要时进行调整。
- 网络配置问题 – 确保网络配置（例如 MTU 大小）允许所需的数据包大小。

错误代码 1226：用户“...”已超出“max_user_connections”资源（当前值：...）

此错误表示用户已超过数据库服务器允许的最大并发连接数。可能的原因包括：

- 连接池配置错误 – 如果使用连接池机制，请确保最大池大小不要设置得太高，以免超出用户的连接限制。

- 数据库实例配置 – 验证数据库实例的 max_user_connections 设置，并在必要时进行调整。
- 高并发 – 如果多个客户端或应用程序使用相同的用户同时连接到数据库，则可能会达到特定于用户的连接限制。

错误代码 2003：无法通过“...”连接到 MySQL 服务器（10061）

当客户端无法与数据库服务器建立 TCP/IP 连接时，通常会发生此错误。这可能由多种问题引起，例如：

- 数据库实例状态 – 确保数据库实例处于 available 状态，且未进行任何维护或备份操作。
- 防火墙规则 – 检查是否有任何防火墙（操作系统、网络或安全组）在阻止指定端口（对于 MySQL，通常为 3306）上的连接。
- 主机名或端点不正确 – 确保连接字符串中使用的主机名或端点正确且与数据库实例相匹配。
- 网络连接问题 – 验证客户端计算机是否可以通过网络访问数据库实例。检查是否存在任何网络中断、路由问题或者 VPC 或子网配置错误。

错误代码 2005：未知的 MySQL 服务器主机“...”（11001）

当客户端无法将数据库服务器的主机名或端点解析为 IP 地址时，就会发生此错误。可能的原因包括：

- DNS 解析问题 – 验证客户端计算机是否可以使用 DNS 正确解析主机名。检查 DNS 设置、DNS 缓存，然后尝试使用 IP 地址而不是主机名。
- 主机名或端点不正确 – 仔细检查连接字符串中使用的主机名或端点是否准确。
- 网络配置问题 – 确保客户端的网络配置（例如 VPC、子网和路由表）允许进行 DNS 解析和连接到数据库实例。

错误代码 2026：SSL 连接错误

在尝试连接期间，当 SSL/TLS 配置或证书验证出现问题时，就会发生此错误。可能的原因包括：

- 证书过期 – 检查服务器使用的 SSL/TLS 证书是否已过期并需要续订。
- 证书验证问题 – 验证客户端是否能够正确验证服务器的 SSL/TLS 证书，以及该证书是否可信。
- 网络配置问题 – 确保网络配置允许 SSL/TLS 连接，并且不会阻塞或干扰 SSL/TLS 握手过程。
- SSL/TLS 配置不匹配 – 确保客户端和服务器上的 SSL/TLS 设置（例如，密码套件和协议版本）兼容。

通过了解每个错误代码的详细说明和潜在原因，在使用 Aurora MySQL 数据库时，您可以更好地排查和解决连接问题。

Aurora MySQL 的参数调优建议

最大连接数

调整这些参数有助于防止由于达到允许的最大连接限制而导致的连接问题。确保根据应用程序的并发要求和资源限制适当设置这些值。

- max_connections – 此参数指定数据库实例允许的最大并发连接数。
- max_user_connections – 可以在用户创建和修改过程中指定此参数，并设置特定用户账户允许的最大并发连接数。

网络缓冲区大小

增加这些值可以提高网络性能，特别是对于涉及大型数据传输或结果集的工作负载。但请谨慎行事，因为较大的缓冲区大小会占用更多内存。

- net_buffer_length – 此参数设置客户端连接和结果缓冲区的初始大小，从而平衡内存使用量和查询性能。
- max_allowed_packet – 此参数指定数据库实例可以发送或接收的单个网络数据包的最大大小。

网络压缩（客户端）

启用网络压缩可以减少网络带宽使用量，但会增加客户端和服务器端的 CPU 开销。

- compress – 此参数启用或禁用客户端/服务器通信的网络压缩。
- compress_protocol – 此参数指定用于网络通信的压缩协议。

网络性能调优

调整这些超时有助于管理空闲连接并防止资源耗尽，但要谨慎行事，因为较低的值可能会导致连接过早终止。

- interactive_timeout – 此参数指定服务器在关闭交互式连接之前等待活动的秒数。
- wait_timeout – 此参数确定服务器在关闭非交互式连接之前等待活动的秒数。

网络超时设置

调整这些超时有助于解决与连接速度慢或无响应有关的问题。但注意不要将它们设置得太低，否则可能会导致连接过早失败。

- net_read_timeout – 此参数指定在结束读取操作之前等待来自连接的更多数据的秒数。
- net_write_timeout – 此参数确定在结束写入操作之前等待数据块写入连接的秒数。

排查 Aurora MySQL 数据库连接问题的示例

以下示例说明如何识别和排查 Aurora MySQL 的数据库连接问题。

示例 1：排查失败的连接尝试

连接尝试失败可能有多种原因，包括身份验证失败、SSL/TLS 握手失败、达到 max_connections 限制以及数据库实例上的资源限制。

您可以通过性能详情或使用以下命令来跟踪失败的连接数。

```
mysql> show global status like 'aborted_connects';
+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| Aborted_connects | 7     |
+------------------+-------+
1 row in set (0.00 sec)
```

如果 Aborted_connects 数随着时间的推移而增加，则应用程序可能会出现间歇性连接问题。

您可以使用 Aurora 高级审计来记录客户端的连接和断开连接情况。您可以通过在数据库集群参数组中设置以下参数来执行此操作：

- server_audit_logging = 1
- server_audit_events = CONNECT

以下是登录失败的审计日志摘录。

```
1728498527380921,auora-mysql-node1,user_1,172.31.49.222,147189,0,FAILED_CONNECT,,,1045
1728498527380940,auora-mysql-node1,user_1,172.31.49.222,147189,0,DISCONNECT,,,0
```

其中：

- 1728498527380921 – 登录失败发生时的纪元时间戳
- aurora-mysql-node1 – 连接失败的 Aurora MySQL 集群节点的实例标识符
- user_1 – 登录失败的数据库用户的名称
- 172.31.49.222 – 建立连接的客户端的私有 IP 地址
- 147189 – 登录失败的连接 ID

- FAILED_CONNECT – 表示连接失败。
- 1045 – 返回代码。非零值表示错误。在本例中，1045 对应于访问被拒绝。

有关更多信息，请参阅 MySQL 文档中的服务器错误代码和客户端错误代码。

您还可以检查 Aurora MySQL 错误日志中是否存在任何相关的错误消息，例如：

```
2024-10-09T19:26:59.310443Z 220 [Note] [MY-010926] [Server] Access denied for user
 'user_1'@'172.31.49.222' (using password: YES) (sql_authentication.cc:1502)
```

示例 2：排查异常客户端断开连接问题

您可以通过性能详情或使用以下命令来跟踪异常客户端断开连接数。

```
mysql> show global status like 'aborted_clients';
+-----------------+-------+
| Variable_name   | Value |
+-----------------+-------+
| Aborted_clients | 9     |
+-----------------+-------+
1 row in set (0.01 sec)
```

如果 Aborted_clients 数随着时间的推移而增加，则说明应用程序无法正确关闭与数据库的连接。如果连接未正确关闭，则可能导致资源泄漏和潜在的性能问题。不必要地使连接保持打开状态会消耗系统资源（例如内存和文件描述符），最终可能导致应用程序或服务器失去响应或重新启动。

您可以使用以下查询来识别未正确关闭连接的账户。它检索用户账户名称、用户进行连接所在的主机、未关闭的连接数以及未关闭的连接所占的百分比。

```
SELECT
    ess.user,
    ess.host,
    (a.total_connections - a.current_connections) - ess.count_star AS not_closed,
    (((a.total_connections - a.current_connections) - ess.count_star) * 100) /
 (a.total_connections - a.current_connections) AS pct_not_closed
FROM
    performance_schema.events_statements_summary_by_account_by_event_name AS ess
    JOIN performance_schema.accounts AS a ON (ess.user = a.user AND ess.host = a.host)
WHERE
    ess.event_name = 'statement/com/quit'
```

```
    AND (a.total_connections - a.current_connections) > ess.count_star;

+----------+---------------+------------+----------------+
| user     | host          | not_closed | pct_not_closed |
+----------+---------------+------------+----------------+
| user1    | 172.31.49.222 |          1 |        33.3333 |
| user1    | 172.31.93.250 |       1024 |        12.1021 |
| user2    | 172.31.93.250 |         10 |        12.8551 |
+----------+---------------+------------+----------------+
3 rows in set (0.00 sec)
```

确定未关闭连接的用户账户和主机后，您可以继续检查未正常关闭连接的代码。

例如，通过 Python 中的 MySQL 连接器，使用连接对象的 `close()` 方法来关闭连接。以下是建立数据库连接、执行查询和关闭连接的示例函数：

```
import mysql.connector

def execute_query(query):
    # Establish a connection to the database
    connection = mysql.connector.connect(
        host="your_host",
        user="your_username",
        password="your_password",
        database="your_database"
    )

    try:
        # Create a cursor object
        cursor = connection.cursor()

        # Execute the query
        cursor.execute(query)

        # Fetch and process the results
        results = cursor.fetchall()
        for row in results:
            print(row)

    finally:
        # Close the cursor and connection
        cursor.close()
        connection.close()
```

在此示例中，在 finally 数据块中调用 connection.close() 方法以确保无论是否发生异常，连接均已关闭。

排除 Aurora MySQL 数据库的查询性能故障

MySQL 通过系统变量（它们影响评估查询计划的方式）、可切换的优化、优化器和索引提示以及优化器成本模型，来提供查询优化器控制。这些数据点不仅有助于比较不同的 MySQL 环境，还可以用来比较以前的查询执行计划和当前的执行计划，以及随时了解 MySQL 查询的总体执行情况。

查询性能取决于许多因素，包括执行计划、表架构和大小、统计信息、资源、索引和参数配置。查询调优要求识别瓶颈并优化执行路径。

- 找到查询的执行计划，并检查查询是否正在使用适当的索引。您可以通过使用 EXPLAIN 和查看每个计划的详细信息来优化查询。
- Aurora MySQL 版本 3（与 MySQL 8.0 社区版兼容）使用 EXPLAIN ANALYZE 语句。EXPLAIN ANALYZE 语句是一个分析工具，可显示 MySQL 在查询的哪些方面花费了时间以及原因。借助 EXPLAIN ANALYZE，Aurora MySQL 计划、准备和运行查询，同时计算行数并测量在执行计划的各个点花费的时间。查询完成后，EXPLAIN ANALYZE 输出计划及其测量值，而不是查询结果。
- 通过使用 ANALYZE 语句使架构统计信息保持更新。查询优化器有时会因为统计信息过时而选择较差的执行计划。这可能会导致查询性能不佳，因为表和索引的基数估计值都不准确。innodb_table_stats 表的 last_update 列显示了上次更新架构统计信息的时间，这可以很好地表明是否“过时”。
- 可能会出现其它问题，例如数据的分布偏斜，而表基数没有考虑这些问题。有关更多信息，请参阅 MySQL 文档中的 Estimating ANALYZE TABLE complexity for InnoDB tables 和 Histogram statistics in MySQL。

了解查询所花的时间

以下是确定查询所花时间的方法：

- 分析
- 性能模式
- 查询优化器

分析

默认情况下，分析功能处于禁用状态。启用分析功能，然后运行慢速查询并查看其分析结果。

```
SET profiling = 1;
Run your query.
```

```
SHOW PROFILE;
```

1. 确定花费时间最多的阶段。根据 MySQL 文档中的常规线程状态，读取和处理 SELECT 语句的行通常是在给定查询的生命周期内运行时间最长的状态。您可以使用 EXPLAIN 语句来了解 MySQL 如何运行此查询。
2. 查看慢速查询日志来评估 rows_examined 和 rows_sent，来确保每个环境中的工作负载相似。有关更多信息，请参阅 Aurora MySQL 数据库日志记录。
3. 对属于已识别查询的一部分的表运行以下命令：

   ```
   SHOW TABLE STATUS\G;
   ```

4. 在每个环境上运行查询之前和之后捕获以下输出：

   ```
   SHOW GLOBAL STATUS;
   ```

5. 在每个环境上运行以下命令，来查看是否有任何其它查询/会话影响此示例查询的性能。

   ```
   SHOW FULL PROCESSLIST;

   SHOW ENGINE INNODB STATUS\G;
   ```

   有时，当服务器上的资源繁忙时，它会影响服务器上的每个其它操作，包括查询。您还可以在运行查询时定期捕获信息，或者设置 cron 任务来按有用的间隔捕获信息。

性能模式

性能模式提供有关服务器运行时性能的有用信息，同时对该性能的影响最小。这与提供有关数据库实例的模式信息的 information_schema 不同。有关更多信息，请参阅 Aurora MySQL 上性能详情的性能架构概述。

查询优化器跟踪

为了理解为什么选择特定的查询计划来执行，您可以设置 optimizer_trace 来访问 MySQL 查询优化器。

运行优化器跟踪，来显示有关优化器可用的所有路径及其选择的大量信息。

```
SET SESSION OPTIMIZER_TRACE="enabled=on";
SET optimizer_trace_offset=-5, optimizer_trace_limit=5;

-- Run your query.
```

```
SELECT * FROM table WHERE x = 1 AND y = 'A';

-- After the query completes:
SELECT * FROM information_schema.OPTIMIZER_TRACE;
SET SESSION OPTIMIZER_TRACE="enabled=off";
```

查看查询优化器设置

Aurora MySQL 版本 3（与 MySQL 8.0 社区版兼容）与 Aurora MySQL 版本 2（与 MySQL 5.7 社区版兼容）相比，前者具有许多与优化器相关的更改。如果您对于 optimizer_switch 有一些自定义值，我们建议您查看默认值的差异，并设置最适合您的工作负载的 optimizer_switch 值。我们还建议您测试可用于 Aurora MySQL 版本 3 的选项，来检查查询的执行情况。

> **Note**
>
> Aurora MySQL 版本 3 对于 innodb_stats_persistent_sample_pages 参数使用社区默认值 20。

您可以使用以下命令来显示 optimizer_switch 值：

```
SELECT @@optimizer_switch\G;
```

下表显示 Aurora MySQL 版本 2 和 3 的默认 optimizer_switch 值。

设置	Aurora MySQL 版本 2	Aurora MySQL 版本 3
batched_key_access	off	off
block_nested_loop	on	on
condition_fanout_filter	on	on
derived_condition_pushdown	–	on
derived_merge	on	on
duplicateweedout	on	on
engine_condition_pushdown	on	on

设置	Aurora MySQL 版本 2	Aurora MySQL 版本 3
firstmatch	on	on
hash_join	off	on
hash_join_cost_based	on	–
hypergraph_optimizer	–	off
index_condition_pushdown	on	on
index_merge	on	on
index_merge_intersection	on	on
index_merge_sort_union	on	on
index_merge_union	on	on
loosescan	on	on
materialization	on	on
mrr	on	on
mrr_cost_based	on	on
prefer_ordering_index	on	on
semijoin	on	on
skip_scan	–	on
subquery_materialization_cost_based	on	on
subquery_to_derived	–	off
use_index_extensions	on	on
use_invisible_indexes	–	off

有关更多信息，请参阅 MySQL 文档中的 Switchable optimizations (MySQL 5.7) 和 Switchable optimizations (MySQL 8.0) 。

Amazon Aurora MySQL 参考

此参考包括有关 Aurora MySQL 参数、状态变量和常规 SQL 扩展或与社区 MySQL 数据库引擎的差异的信息。

主题

- Aurora MySQL 配置参数
- Aurora MySQL 全局状态变量
- Aurora MySQL 等待事件
- Aurora MySQL 线程状态
- Aurora MySQL 隔离级别
- Aurora MySQL 提示
- Aurora MySQL 存储过程参考
- Aurora MySQL 特定的 information_schema 表

Aurora MySQL 配置参数

您可以使用数据库参数组中的参数按照与管理其他 Amazon RDS MySQL 数据库实例相同的方法管理 Amazon Aurora 数据库集群。Amazon Aurora 不同于其他数据库引擎，因为您具有一个包含多个数据库实例的数据库集群。因此，您用于管理 Aurora MySQL 数据库集群的有些参数将应用于整个集群。其他参数则仅应用于数据库集群中的特定数据库实例。

要管理集群级参数，请使用数据库集群参数组。要管理实例级参数，请使用数据库参数组。Aurora MySQL 数据库集群中的每个数据库实例均与 MySQL 数据库引擎兼容。不过，您在集群级别应用某些 MySQL 数据库引擎参数，并使用数据库集群参数组管理这些参数。您无法在 Aurora 数据库集群中实例的数据库参数组中查找集群级参数。本主题后面提供了集群级参数的列表。

您可以使用AWS Management Console、AWS CLI 或 Amazon RDS API 管理集群级参数和实例级参数。您可以使用单独的命令管理集群级参数和实例级参数。例如，您可以使用 modify-db-cluster-parameter-group CLI 命令来管理数据库集群参数组中的集群级参数。您可以使用 modify-db-parameter-group CLI 命令来为数据库集群中的数据库实例管理数据库参数组中的实例级参数。

您可以在控制台中或者使用 CLI 或 RDS API 查看集群级别和实例级别的参数。例如，您可以使用 describe-db-cluster-parameters AWS CLI 命令来查看数据库集群参数组中的集群级参数。您可以使用 describe-db-parameters CLI 命令来查看数据库集群中数据库实例的数据库参数组中的实例级参数。

> **Note**
>
> 每个默认参数组包含参数组中所有参数的默认值。如果该参数具有此值的“引擎默认值”，请参阅特定版本的 MySQL 或 PostgreSQL 文档获取实际默认值。
> 除非另有说明，否则下表中列出的参数对于 Aurora MySQL 版本 2 和 3 有效。

有关数据库参数组的更多信息，请参阅 Amazon Aurora 的参数组。有关 Aurora Serverless v1 集群的规则和限制，请参阅Aurora Serverless v1 的参数组。

主题

- 集群级别的参数
- 实例级参数
- 不适用于 Aurora MySQL 的 MySQL 参数

集群级别的参数

下表显示了适用于整个 Aurora MySQL 数据库集群的所有参数。

参数名称	可修改	备注
aurora_binlog_read_buffer_size	是	仅影响使用二进制日志 (binlog) 复制的集群。有关二进制日志复制的信息，请参阅 Aurora 与 MySQL 之间或 Aurora 与其他 Aurora 数据库集群之间的复制（二进制日志复制）。已从 Aurora MySQL 版本 3 中删除。
aurora_binlog_replication_max_yield_seconds	是	仅影响使用二进制日志 (binlog) 复制的集群。有关二进制日志复制的信息，请参阅 Aurora 与 MySQL 之间或 Aurora 与其他 Aurora 数据库集群之间的复制（二进制日志复制）。
aurora_binlog_replication_sec_index_parallel_workers	是	设置在为具有多个二级索引的大型表复制事务时可用于应用二级索引更改的并

参数名称	可修改	备注
		行线程总数。默认情况下，此参数设置为 0（禁用）。 此参数可用于 Aurora MySQL 版本 3.06 及更高版本。有关更多信息，请参阅 优化 Aurora MySQL 的二进制日志复制。
aurora_binlog_use_large_read_buffer	是	仅影响使用二进制日志 (binlog) 复制的集群。有关二进制日志复制的信息，请参阅 Aurora 与 MySQL 之间或 Aurora 与其他 Aurora 数据库集群之间的复制（二进制日志复制）。已从 Aurora MySQL 版本 3 中删除。
aurora_disable_hash_join	是	将此参数设置为 ON 可在 Aurora MySQL 版本 2.09 或更高版本中关闭哈希联接优化。版本 3 不支持此参数。有关更多信息，请参阅 Amazon Aurora MySQL 的并行查询。
aurora_enable_replica_log_compression	是	有关更多信息，请参阅“Amazon Aurora MySQL 复制的性能注意事项”。不适用于作为 Aurora 全局数据库的一部分的集群。已从 Aurora MySQL 版本 3 中删除。
aurora_enable_repl_bin_log_filtering	是	有关更多信息，请参阅“Amazon Aurora MySQL 复制的性能注意事项”。不适用于作为 Aurora 全局数据库的一部分的集群。已从 Aurora MySQL 版本 3 中删除。
aurora_enable_staggered_replica_restart	是	此设置在 Aurora MySQL 版本 3 中可用，但并未使用。

参数名称	可修改	备注
aurora_enable_zdr	是	该设置在 Aurora MySQL 2.10 及更高版本中默认开启。有关更多信息，请参阅“Amazon Aurora MySQL 的零停机重启 (ZDR)”。
aurora_enhanced_binlog	是	将此参数的值设置为 1，以在 Aurora MySQL 版本 3.03.1 及更高版本中开启增强型二进制日志。有关更多信息，请参阅 为 Aurora MySQL 设置增强型二进制日志。
aurora_jemalloc_background_thread	是	使用此参数以使后台线程能够执行内存维护操作。允许的值为 0（禁用）和 1（启用）。默认值为 0。 此参数适用于 Aurora MySQL 版本 3.05 及更高版本。
aurora_jemalloc_dirty_decay_ms	是	使用此参数以将释放的内存保留一段时间（以毫秒为单位）。保留内存可以更快地重用。允许的值为 0–18446744073709551615 。默认值（0）将所有内存作为可释放内存返回给操作系统。 此参数适用于 Aurora MySQL 版本 3.05 及更高版本。
aurora_jemalloc_tcache_enabled	是	使用此参数，通过绕过内存领域，在线程本地缓存中处理小内存请求（最多 32KiB）。允许的值为 0（禁用）和 1（启用）。默认值为 1。 此参数适用于 Aurora MySQL 版本 3.05 及更高版本。

参数名称	可修改	备注
aurora_load_from_s3_role	是	有关更多信息，请参阅 将数据从 Amazon S3 存储桶中的文本文件加载到 Amazon Aurora MySQL 数据库集群。目前在 Aurora MySQL 版本 3 中不可用。使用 aws_default_s3_role 。
aurora_mask_password_hashes_type	是	原定设置情况下，该设置在 Aurora MySQL 2.11 及更高版本中开启。 使用此设置可在慢速查询和审核日志中屏蔽 Aurora MySQL 密码哈希。允许的值为 0 和 1（原定设置）。设置为 1 时，密码将记录为 <secret>。设置为 0 时，密码将记录为哈希（#）值。
aurora_select_into_s3_role	是	有关更多信息，请参阅 将数据从 Amazon Aurora MySQL 数据库集群保存到 Amazon S3 存储桶中的文本文件。目前在 Aurora MySQL 版本 3 中不可用。使用 aws_default_s3_role 。
authentication_kerberos_caseins_cmp	是	控制 authentication_kerberos 插件不区分大小写的用户名比较。将它设置为 true 以进行不区分大小写的比较。原定设置情况下，使用区分大小写的比较（false）。有关更多信息，请参阅 对 Aurora MySQL 使用 Kerberos 身份验证。 此参数适用于 Aurora MySQL 版本 3.03 及更高版本。
auto_increment_increment	是	
auto_increment_offset	是	

参数名称	可修改	备注
aws_default_lambda_role	是	有关更多信息，请参阅“从 Amazon Aurora MySQL 数据库集群中调用 Lambda 函数”。
aws_default_s3_role	是	从数据库集群调用 LOAD DATA FROM S3、LOAD XML FROM S3 或 SELECT INTO OUTFILE S3 语句时使用。 在 Aurora MySQL 版本 2 中，如果没有为相应语句的 aurora_load_from_s3_role 或 aurora_select_into_s3_role 指定 IAM 角色，则使用在该参数中指定的 IAM 角色。 在 Aurora MySQL 版本 3 中，始终使用为该参数指定的 IAM 角色。 有关更多信息，请参阅 将 IAM 角色与 Amazon Aurora MySQL 数据库集群关联。
binlog_backup	是	将此参数的值设置为 0，以在 Aurora MySQL 版本 3.03.1 及更高版本中开启增强型二进制日志。您只能在使用增强型二进制日志时关闭此参数。有关更多信息，请参阅 为 Aurora MySQL 设置增强型二进制日志。
binlog_checksum	是	如果未设置此参数，AWS CLI 和 RDS API 将报告 None 的值。在这种情况下，Aurora MySQL 会使用引擎默认值，即 CRC32。这与关闭校验和的显式设置 NONE 不同。
binlog-do-db	是	此参数适用于 Aurora MySQL 版本 3。

参数名称	可修改	备注
binlog_format	是	有关更多信息，请参阅 Aurora 与 MySQL 之间或 Aurora 与其他 Aurora 数据库集群之间的复制（二进制日志复制）。
binlog_group_commit_sync_delay	是	此参数适用于 Aurora MySQL 版本 3。
binlog_group_commit_sync_no_delay_count	是	此参数适用于 Aurora MySQL 版本 3。
binlog-ignore-db	是	此参数适用于 Aurora MySQL 版本 3。
binlog_replication_globaldb	是	将此参数的值设置为 0，以在 Aurora MySQL 版本 3.03.1 及更高版本中开启增强型二进制日志。您只能在使用增强型二进制日志时关闭此参数。有关更多信息，请参阅 为 Aurora MySQL 设置增强型二进制日志。
binlog_row_image	否	
binlog_row_metadata	是	此参数适用于 Aurora MySQL 版本 3。
binlog_row_value_options	是	此参数适用于 Aurora MySQL 版本 3。
binlog_rows_query_log_events	是	
binlog_transaction_compression	是	此参数适用于 Aurora MySQL 版本 3。
binlog_transaction_compression_level_zstd	是	此参数适用于 Aurora MySQL 版本 3。

参数名称	可修改	备注
binlog_transaction_dependency_history_size	是	此参数设置了保留在内存中的行哈希数的上限，这些哈希值用于查找上次修改给定行的事务。达到这个哈希数后，历史记录将被清除。 此参数适用于 Aurora MySQL 版本 2.12 及更高版本以及版本 3。
binlog_transaction_dependency_tracking	是	此参数适用于 Aurora MySQL 版本 3。
character-set-client-handshake	是	
character_set_client	是	
character_set_connection	是	
character_set_database	是	
character_set_filesystem	是	
character_set_results	是	
character_set_server	是	
collation_connection	是	
collation_server	是	
completion_type	是	
default_storage_engine	否	Aurora MySQL 集群对所有数据使用 InnoDB 存储引擎。
enforce_gtid_consistency	有时	在 Aurora MySQL 版本 2 及更高版本中可修改。

参数名称	可修改	备注
event_scheduler	是	指示事件计划程序的状态。 在 Aurora MySQL 版本 3 中，只能在集群级别修改。
gtid-mode	有时	在 Aurora MySQL 版本 2 及更高版本中可修改。
information_schema_stats_expiry	是	MySQL 数据库服务器从存储引擎获取数据并替换缓存中的数据前的秒数。允许的值为 0–31536000。 此参数适用于 Aurora MySQL 版本 3。
init_connect	是	服务器要对连接的每个客户端运行的命令。在设置中使用双引号 (") 以避免连接失败，例如： `SET optimizer_switch="hash_join =off"` 在 Aurora MySQL 版本 3 中，此参数不适用于具有 CONNECTION_ADMIN 权限的用户。这包括 Aurora 主用户。有关更多信息，请参阅 基于角色的权限模型。
innodb_adaptive_hash_index	是	在 Aurora MySQL 版本 2 和 3 中，您可以在数据库集群级别修改此参数。 读取器数据库实例不支持自适应哈希索引。

参数名称	可修改	备注
innodb_aurora_instant_alter_column_allowed	是	控制 INSTANT 算法是否可用于全局级别的 ALTER COLUMN 操作。允许的值如下所示： • 0 – 不允许使用 INSTANT 算法执行 ALTER COLUMN 操作（OFF）。恢复到其它算法。 • 1 – 允许使用 INSTANT 算法执行 ALTER COLUMN 操作（ON）。这是默认值。 有关更多信息，请参阅 MySQL 文档中的列操作。 此参数适用于 Aurora MySQL 版本 3.05 及更高版本。
innodb_autoinc_lock_mode	是	
innodb_checksums	不支持	已从 Aurora MySQL 版本 3 中删除。
innodb_cmp_per_index_enabled	是	
innodb_commit_concurrency	是	
innodb_data_home_dir	不支持	Aurora MySQL 使用不在其中直接访问文件系统的托管式实例。

参数名称	可修改	备注
innodb_deadlock_detect	是	此选项用于在 Aurora MySQL 版本 2.11 及更高版本以及版本 3 中禁用死锁检测。 在高并发系统中，当许多线程等待同一个锁时，死锁检测可能会导致速度下降。有关此参数的更多信息，请参阅 MySQL 文档。
innodb_default_row_format	是	此参数定义 InnoDB 表（包括用户创建的 InnoDB 临时表）的原定设置行格式。它适用于 Aurora MySQL 版本 2 和 3。 其值可以是 DYNAMIC、COMPACT 或 REDUNDANT。 。
innodb_file_per_table	是	此参数影响表存储的组织方式。有关更多信息，请参阅 存储扩展。
innodb_flush_log_at_trx_commit	是	我们强烈建议您使用默认值 1。 在 Aurora MySQL 版本 3 中，在将此参数设置为 1 以外的值之前，必须先将 innodb_trx_commit_allow_data_loss 的值设置为 1。 有关更多信息，请参阅 配置刷新日志缓冲区的频率。
innodb_ft_max_token_size	是	
innodb_ft_min_token_size	是	
innodb_ft_num_word_optimize	是	
innodb_ft_sort_pll_degree	是	

参数名称	可修改	备注
innodb_online_alter_log_max_size	是	
innodb_optimize_fulltext_only	是	
innodb_page_size	否	
innodb_print_all_deadlocks	是	开启后，在 Aurora MySQL 错误日志中记录有关所有 InnoDB 死锁的信息。有关更多信息，请参阅 最大限度地减少 Aurora MySQL 死锁以及排查相关问题。
innodb_purge_batch_size	是	
innodb_purge_threads	是	
innodb_rollback_on_timeout	是	
innodb_rollback_segments	是	
innodb_spin_wait_delay	是	
innodb_strict_mode	是	
innodb_support_xa	是	已从 Aurora MySQL 版本 3 中删除。
innodb_sync_array_size	是	
innodb_sync_spin_loops	是	
innodb_stats_include_delete_marked	是	启用此参数后，InnoDB 在计算持久优化器统计数据时会包括带有删除标记的记录。 此参数适用于 Aurora MySQL 版本 2.12 及更高版本以及版本 3。
innodb_table_locks	是	

参数名称	可修改	备注
innodb_trx_commit_allow_data_loss	是	在 Aurora MySQL 版本 3 中，将此参数的值设置为 1，这样您就可以更改 innodb_flush_log_at_trx_commit 的值。 innodb_trx_commit_allow_data_loss 的默认值为 0。 有关更多信息，请参阅 配置刷新日志缓冲区的频率。
innodb_undo_directory	否	Aurora MySQL 使用不在其中直接访问文件系统的托管式实例。
internal_tmp_disk_storage_engine	是	控制哪个内存存储引擎用于内部临时表。允许的值包括 INNODB和 MYISAM。 此参数适用于 Aurora MySQL 版本 2。
internal_tmp_mem_storage_engine	是	控制哪个内存存储引擎用于内部临时表。允许的值包括 MEMORY和 TempTable 。 此参数适用于 Aurora MySQL 版本 3。
key_buffer_size	是	MyISAM 表的密钥缓存。有关更多信息，请参阅密钥缓存 -> cache_lock 互斥锁。
lc_time_names	是	

参数名称	可修改	备注
log_error_suppression_list	是	指定未在 MySQL 错误日志中记录的错误代码列表。这允许您忽略某些非关键错误条件，以帮助保持错误日志干净整洁。有关更多信息，请参阅 MySQL 文档中的 log_error_suppression_list。 此参数适用于 Aurora MySQL 3.03 及更高版本。
low_priority_updates	是	INSERT、UPDATE、DELETE 和 LOCK TABLE WRITE 操作会等待，直至没有待处理的 SELECT 操作。此参数仅影响仅使用表级锁定的存储引擎（MyISAM、MEMORY、MERGE）。 此参数适用于 Aurora MySQL 版本 3。
lower_case_table_names	是（Aurora MySQL 版本 2） 仅在集群创建时（Aurora MySQL 版本 3）	在 Aurora MySQL 版本 2.10 及更高的 2.x 版本中，请确保在更改此设置并重启写入器实例后重启所有读取器实例。有关详细信息，请参阅在具有读取可用性的情况下重启 Aurora 集群。 在 Aurora MySQL 版本 3 中，此参数的值在创建集群时永久设置。如果对此选项使用非默认值，请在升级之前设置 Aurora MySQL 版本 3 自定义参数组，然后在创建版本 3 集群的快照还原操作期间指定参数组。 使用基于 Aurora MySQL 的 Aurora 全局数据库时，如果开启了 lower_case_table_names 参数，则无法执行从 Aurora MySQL 版本 2 到版本 3 的就地升级。有关可以使用的方法的更多信息，请参阅主要版本升级。

参数名称	可修改	备注
master-info-repository	是	已从 Aurora MySQL 版本 3 中删除。
master_verify_checksum	是	Aurora MySQL 版本 2。在 Aurora MySQL 版本 3 中使用 source_verify_checksum 。
max_delayed_threads	是	设置处理 INSERT DELAYED 语句的最大线程数。 此参数适用于 Aurora MySQL 版本 3。
max_error_count	是	存储以供显示的错误、警告和备注消息的最大数量。 此参数适用于 Aurora MySQL 版本 3。
max_execution_time	是	运行 SELECT 语句的超时时间，以毫秒为单位。值可以为 0 – 18446744073709551615 。设置为 0 时，没有超时。 有关更多信息，请参阅 MySQL 文档中的 max_execution_time。
min_examined_row_limit	是	使用此参数可防止记录所检查的行数少于指定行数的查询。 此参数适用于 Aurora MySQL 版本 3。
partial_revokes	否	此参数适用于 Aurora MySQL 版本 3。
preload_buffer_size	是	预加载索引时分配的缓冲区的大小。 此参数适用于 Aurora MySQL 版本 3。
query_cache_type	是	已从 Aurora MySQL 版本 3 中删除。

参数名称	可修改	备注
read_only	是	启用此参数时，除副本线程执行的更新外，服务器不允许任何更新。 对于 Aurora MySQL 版本 2，有效值如下所示： • 0 – OFF • 1 – ON • {TrueIfReplica} – ON 适用于只读副本。这是默认值。 • {TrueIfClusterReplica} – ON 适用于副本集群，例如跨区域只读副本、Aurora Global Database 中的辅助集群以及蓝绿部署。 对于 Aurora MySQL 版本 3，有效值如下所示： • 0 – OFF。这是默认值。 • 1 – ON • {TrueIfClusterReplica} – ON 适用于副本集群，例如跨区域只读副本、Aurora Global Database 中的辅助集群，以及蓝绿部署。 在 Aurora MySQL 版本 3 中，此参数不适用于具有 CONNECTION_ADMIN 权限的用户。这包括 Aurora 主用户。有关更多信息，请参阅 基于角色的权限模型。
relay-log-space-limit	是	此参数适用于 Aurora MySQL 版本 3。

参数名称	可修改	备注
replica_parallel_type	是	此参数对已处于准备阶段的所有未提交线程的副本启用并行执行，而不会违反一致性。它适用于 Aurora MySQL 版本 3。 在 Aurora MySQL 版本 3.03.* 及更低版本中，默认值为 DATABASE。在 Aurora MySQL 版本 3.04 及更高版本中，默认值为 LOGICAL_CLOCK。
replica_preserve_commit_order	是	此参数适用于 Aurora MySQL 版本 3。
replica_transaction_retries	是	此参数适用于 Aurora MySQL 版本 3。
replica_type_conversions	是	此参数决定了副本上使用的类型转换。允许的值为：ALL_LOSSY 、ALL_NON_LOSSY 、ALL_SIGNED 和 ALL_UNSIGNED 。有关更多信息，请参阅 MySQL 文档中的在源和副本上使用不同的表定义进行复制。 此参数适用于 Aurora MySQL 版本 3。
replicate-do-db	是	此参数适用于 Aurora MySQL 版本 3。
replicate-do-table	是	此参数适用于 Aurora MySQL 版本 3。
replicate-ignore-db	是	此参数适用于 Aurora MySQL 版本 3。
replicate-ignore-table	是	此参数适用于 Aurora MySQL 版本 3。
replicate-wild-do-table	是	此参数适用于 Aurora MySQL 版本 3。
replicate-wild-ignore-table	是	此参数适用于 Aurora MySQL 版本 3。

参数名称	可修改	备注
require_secure_transport	是	此参数适用于 Aurora MySQL 版本 2 和 3。有关更多信息，请参阅 与 Aurora MySQL 数据库集群的 TLS 连接。
rpl_read_size	是	此参数适用于 Aurora MySQL 版本 3。
server_audit_cw_upload	否	该参数在 Aurora MySQL 中已弃用。使用 server_audit_logs_upload 。 有关更多信息，请参阅 将 Amazon Aurora MySQL 日志发布到 Amazon CloudWatch Logs。
server_audit_events	是	有关更多信息，请参阅 在 Amazon Aurora MySQL 数据库集群中使用高级审计。
server_audit_excl_users	是	有关更多信息，请参阅 在 Amazon Aurora MySQL 数据库集群中使用高级审计。
server_audit_incl_users	是	有关更多信息，请参阅 在 Amazon Aurora MySQL 数据库集群中使用高级审计。
server_audit_logging	是	有关将日志上传到 Amazon CloudWatch Logs 的说明，请参阅将 Amazon Aurora MySQL 日志发布到 Amazon CloudWatch Logs。

参数名称	可修改	备注
server_audit_logs_upload	是	您可以通过启用高级审核并将此参数设置为 1，将审核日志发布到 CloudWatch Logs。server_audit_logs_upload 参数的默认值为 0。 有关更多信息，请参阅 将 Amazon Aurora MySQL 日志发布到 Amazon CloudWatch Logs。
server_id	否	
skip-character-set-client-handshake	是	
skip_name_resolve	否	
slave-skip-errors	是	仅适用于 Aurora MySQL 版本 2 的集群，具备 MySQL 5.7 兼容性。
source_verify_checksum	是	此参数适用于 Aurora MySQL 版本 3。
sync_frm	是	已从 Aurora MySQL 版本 3 中删除。
thread_cache_size	是	要缓存的线程数。此参数适用于 Aurora MySQL 版本 2 和 3。
time_zone	是	默认情况下，Aurora 数据库集群的时区是协调世界时（UTC）。您可以改为将数据库群集中实例的时区设置为您的应用程序的本地时区。有关更多信息，请参阅 Amazon Aurora 数据库集群的本地时区。
tls_version	是	有关更多信息，请参阅“Aurora MySQL 的 TLS 版本”。

实例级参数

下表显示了适用于 Aurora MySQL 数据库集群中特定数据库实例的所有参数。

参数名称	可修改	备注
activate_all_roles_on_login	是	此参数适用于 Aurora MySQL 版本 3。
allow-suspicious-udfs	否	
aurora_disable_hash_join	是	将此参数设置为 ON 可在 Aurora MySQL 版本 2.09 或更高版本中关闭哈希联接优化。版本 3 不支持此参数。有关更多信息，请参阅 Amazon Aurora MySQL 的并行查询。
aurora_lab_mode	是	有关更多信息，请参阅 Amazon Aurora MySQL 实验室模式。已从 Aurora MySQL 版本 3 中删除。
aurora_oom_response	是	Aurora MySQL 版本 2 和 3 支持此参数。有关更多信息，请参阅 排查 Aurora MySQL 数据库内存不足问题。
aurora_parallel_query	是	设置为 ON 可在 Aurora MySQL 版本 2.09 或更高版本中开启并行查询。旧的 aurora_pq 参数未在这些版本中使用。有关更多信息，请参阅“Amazon Aurora MySQL 的并行查询”。
aurora_pq	是	设置为 OFF 可在 2.09 之前的 Aurora MySQL 版本中为特定数据库实例关闭并行查询。在 2.09 或更高版本中，改为使用 aurora_parallel_query 打开和关闭并行查询。有关更多信息，请参阅 Amazon Aurora MySQL 的并行查询。
aurora_read_replica_read_committed	是	为 Aurora 副本启用 READ COMMITTED 隔离级别，并更改隔离行为，以在执行

参数名称	可修改	备注
		长时间运行的查询期间减少清除滞后。仅当您了解行为更改及其对查询结果的影响时，才启用该设置。例如，该设置使用了没有 MySQL 默认设置那么严格的隔离。启用该设置后，长时间运行的查询可能会看到同一行的多个副本，因为 Aurora 会在查询运行期间重组表数据。有关更多信息，请参阅 Aurora MySQL 隔离级别。
aurora_tmptable_enable_per_table_limit	是	确定 tmp_table_size 参数是否控制 Aurora MySQL 版本 3.04 及更高版本中由 TempTable 存储引擎创建的内存中临时表的最大大小。 有关更多信息，请参阅 限制内部内存中临时表的大小。
aurora_use_vector_instructions	是	启用此参数后，Aurora MySQL 将使用现代 CPU 提供的优化型矢量处理指令来提高 I/O 密集型工作负载的性能。 默认情况下，会在 Aurora MySQL 版本 3.05 及更高版本中启用该设置。
autocommit	是	
automatic_sp_privileges	是	
back_log	是	
basedir	不支持	Aurora MySQL 使用不在其中直接访问文件系统的托管式实例。
binlog_cache_size	是	

参数名称	可修改	备注
binlog_max_flush_queue_time	是	
binlog_order_commits	是	
binlog_stmt_cache_size	是	
binlog_transaction_compression	是	此参数适用于 Aurora MySQL 版本 3。
binlog_transaction_compression_level_zstd	是	此参数适用于 Aurora MySQL 版本 3。
bulk_insert_buffer_size	是	
concurrent_insert	是	
connect_timeout	是	
core-file	不支持	Aurora MySQL 使用不在其中直接访问文件系统的托管式实例。
datadir	否	Aurora MySQL 使用不在其中直接访问文件系统的托管式实例。
default_authentication_plugin	否	此参数适用于 Aurora MySQL 版本 3。
default_time_zone	否	
default_tmp_storage_engine	是	临时表的原定设置存储引擎。
default_week_format	是	
delay_key_write	是	
delayed_insert_limit	是	
delayed_insert_timeout	是	

参数名称	可修改	备注
delayed_queue_size	是	
div_precision_increment	是	
end_markers_in_json	是	
eq_range_index_dive_limit	是	
event_scheduler	有时	指示事件计划程序的状态。 在 Aurora MySQL 版本 3 中，只能在集群级别修改。
explicit_defaults_for_timestamp	是	
flush	否	
flush_time	是	
ft_boolean_syntax	否	
ft_max_word_len	是	
ft_min_word_len	是	
ft_query_expansion_limit	是	
ft_stopword_file	是	
general_log	是	有关将日志上传到 CloudWatch Logs 的说明，请参阅 将 Amazon Aurora MySQL 日志发布到 Amazon CloudWatch Logs。
general_log_file	否	Aurora MySQL 使用不在其中直接访问文件系统的托管式实例。
group_concat_max_len	是	

参数名称	可修改	备注
host_cache_size	是	
init_connect	是	服务器要对连接的每个客户端运行的命令。在设置中使用双引号 (") 以避免连接失败，例如： `SET optimizer_switch="hash_join =off"` 在 Aurora MySQL 版本 3 中，此参数不适用于具有 CONNECTION_ADMIN 权限的用户，包括 Aurora 主用户。有关更多信息，请参阅 基于角色的权限模型。
innodb_adaptive_hash_index	是	在 Aurora MySQL 版本 2 中，您可以在数据库实例级别修改此参数。在 Aurora MySQL 版本 3 中，只能在数据库集群级别修改此参数。 读取器数据库实例不支持自适应哈希索引。
innodb_adaptive_max_sleep_delay	是	修改此参数不起作用，因为 Aurora 的 innodb_thread_concurrency 始终为 0。

参数名称	可修改	备注
innodb_aurora_max_partitions_for_range	是	在某些无法获得持久统计数据的情况下，您可以使用此参数来提高分区表的行计数估计性能。 您可以将其设置为 0–8192 之间的值，该值决定了在行计数估计期间要检查的分区数。默认值为 0，这将使用所有分区进行估计，与原定设置 MySQL 行为一致。 此参数适用于 Aurora MySQL 版本 3.03.1 及更高版本。
innodb_autoextend_increment	是	
innodb_buffer_pool_dump_at_shutdown	否	
innodb_buffer_pool_dump_now	否	
innodb_buffer_pool_filename	否	
innodb_buffer_pool_load_abort	否	
innodb_buffer_pool_load_at_startup	否	
innodb_buffer_pool_load_now	否	
innodb_buffer_pool_size	是	默认值由公式表示。有关公式中 DBInstanceClassMemory 值的计算方式详细信息，请参阅 数据库参数公式变量。

参数名称	可修改	备注
innodb_change_buffer_max_size	否	Aurora MySQL 完全不使用 InnoDB 更改缓冲区。
innodb_compression_failure_threshold_pct	是	
innodb_compression_level	是	
innodb_compression_pad_pct_max	是	
innodb_concurrency_tickets	是	修改此参数不起作用，因为 Aurora 的 innodb_thread_concurrency 始终为 0。
innodb_deadlock_detect	是	此选项用于在 Aurora MySQL 版本 2.11 及更高版本以及版本 3 中禁用死锁检测。 在高并发系统中，当许多线程等待同一个锁时，死锁检测可能会导致速度下降。有关此参数的更多信息，请参阅 MySQL 文档。
innodb_file_format	是	已从 Aurora MySQL 版本 3 中删除。
innodb_flushing_avg_loops	否	
innodb_force_load_corrupted	否	
innodb_ft_aux_table	是	
innodb_ft_cache_size	是	
innodb_ft_enable_stopword	是	

参数名称	可修改	备注
innodb_ft_server_stopword_table	是	
innodb_ft_user_stopword_table	是	
innodb_large_prefix	是	已从 Aurora MySQL 版本 3 中删除。
innodb_lock_wait_timeout	是	
innodb_log_compressed_pages	否	
innodb_lru_scan_depth	是	
innodb_max_purge_lag	是	
innodb_max_purge_lag_delay	是	
innodb_monitor_disable	是	
innodb_monitor_enable	是	
innodb_monitor_reset	是	
innodb_monitor_reset_all	是	
innodb_old_blocks_pct	是	
innodb_old_blocks_time	是	
innodb_open_files	是	
innodb_print_all_deadlocks	是	开启后，在 Aurora MySQL 错误日志中记录有关所有 InnoDB 死锁的信息。有关更多信息，请参阅 最大限度地减少 Aurora MySQL 死锁以及排查相关问题。
innodb_random_read_ahead	是	

参数名称	可修改	备注
innodb_read_ahead_threshold	是	
innodb_read_io_threads	否	
innodb_read_only	否	Aurora MySQL 根据集群类型管理数据库实例的只读和读/写状态。例如，预置的集群具有一个读/写数据库实例（主实例），并且集群中的所有其他实例都是只读的（Aurora 副本）。
innodb_replication_delay	是	
innodb_sort_buffer_size	是	
innodb_stats_auto_recalc	是	
innodb_stats_method	是	
innodb_stats_on_metadata	是	
innodb_stats_persistent	是	
innodb_stats_persistent_sample_pages	是	
innodb_stats_transient_sample_pages	是	
innodb_thread_concurrency	否	
innodb_thread_sleep_delay	是	修改此参数不起作用，因为 Aurora 的 innodb_thread_concurrency 始终为 0。

参数名称	可修改	备注
interactive_timeout	是	Aurora 会估计 interactive_timeout 和 wait_timeout 的最小值，然后使用这个最小值作为超时来结束所有空闲会话，包括交互式会话和非交互式会话。
internal_tmp_disk_storage_engine	是	控制哪个内存存储引擎用于内部临时表。允许的值包括 INNODB和 MYISAM。 此参数适用于 Aurora MySQL 版本 2。
internal_tmp_mem_storage_engine	是	控制哪个内存存储引擎用于内部临时表。允许的值包括 MEMORY和 TempTable 。 此参数适用于 Aurora MySQL 版本 3。
join_buffer_size	是	
keep_files_on_create	是	
key_buffer_size	是	MyISAM 表的密钥缓存。有关更多信息，请参阅密钥缓存 -> cache_lock 互斥锁。
key_cache_age_threshold	是	
key_cache_block_size	是	
key_cache_division_limit	是	
local_infile	是	
lock_wait_timeout	是	

参数名称	可修改	备注
log-bin	否	将 binlog_format 设置为 STATEMENT 、MIXED 或 ROW 会自动将 log-bin 设置为 ON。将 binlog_format 设置为 OFF 会自动将 log-bin 设置为 OFF。有关更多信息，请参阅“Aurora 与 MySQL 之间或 Aurora 与其他 Aurora 数据库集群之间的复制（二进制日志复制）”。
log_bin_trust_function_creators	是	
log_bin_use_v1_row_events	是	已从 Aurora MySQL 版本 3 中删除。
log_error	否	
log_error_suppression_list	是	指定未在 MySQL 错误日志中记录的错误代码列表。这允许您忽略某些非关键错误条件，以帮助保持错误日志干净整洁。有关更多信息，请参阅 MySQL 文档中的 log_error_suppression_list。 此参数适用于 Aurora MySQL 3.03 及更高版本。
log_output	是	
log_queries_not_using_indexes	是	
log_slave_updates	不支持	Aurora MySQL 版本 2。在 Aurora MySQL 版本 3 中使用 log_replica_updates 。
log_replica_updates	否	Aurora MySQL 版本 3

参数名称	可修改	备注
log_throttle_queries_not_using_indexes	是	
log_warnings	是	已从 Aurora MySQL 版本 3 中删除。
long_query_time	是	
low_priority_updates	是	INSERT、UPDATE、DELETE 和 LOCK TABLE WRITE 操作会等待，直至没有待处理的 SELECT 操作。此参数仅影响仅使用表级锁定的存储引擎（MyISAM、MEMORY、MERGE）。 此参数适用于 Aurora MySQL 版本 3。
max_allowed_packet	是	
max_binlog_cache_size	是	
max_binlog_size	否	
max_binlog_stmt_cache_size	是	
max_connect_errors	是	
max_connections	是	默认值由公式表示。有关公式中 DBInstanceClassMemory 值的计算方式详细信息，请参阅 数据库参数公式变量。如需了解受实例类限制的默认值，请参阅 至 Aurora MySQL 数据库实例的最大连接数。
max_delayed_threads	是	设置处理 INSERT DELAYED 语句的最大线程数。 此参数适用于 Aurora MySQL 版本 3。

参数名称	可修改	备注
max_error_count	是	存储以供显示的错误、警告和备注消息的最大数量。 此参数适用于 Aurora MySQL 版本 3。
max_execution_time	是	运行 SELECT 语句的超时时间，以毫秒为单位。值可以为 0 – 18446744073709551615 。设置为 0 时，没有超时。 有关更多信息，请参阅 MySQL 文档中的 max_execution_time。
max_heap_table_size	是	
max_insert_delayed_threads	是	
max_join_size	是	
max_length_for_sort_data	是	已从 Aurora MySQL 版本 3 中删除。
max_prepared_stmt_count	是	
max_seeks_for_key	是	
max_sort_length	是	
max_sp_recursion_depth	是	
max_tmp_tables	是	已从 Aurora MySQL 版本 3 中删除。
max_user_connections	是	
max_write_lock_count	是	
metadata_locks_cache_size	是	已从 Aurora MySQL 版本 3 中删除。

参数名称	可修改	备注
min_examined_row_limit	是	使用此参数可防止记录所检查的行数少于指定行数的查询。 此参数适用于 Aurora MySQL 版本 3。
myisam_data_pointer_size	是	
myisam_max_sort_file_size	是	
myisam_mmap_size	是	
myisam_sort_buffer_size	是	
myisam_stats_method	是	
myisam_use_mmap	是	
net_buffer_length	是	
net_read_timeout	是	
net_retry_count	是	
net_write_timeout	是	
old-style-user-limits	是	
old_passwords	是	已从 Aurora MySQL 版本 3 中删除。
optimizer_prune_level	是	
optimizer_search_depth	是	
optimizer_switch	是	有关使用此开关的 Aurora MySQL 功能的信息，请参阅 Amazon Aurora MySQL 的最佳实践。
optimizer_trace	是	

参数名称	可修改	备注
optimizer_trace_features	是	
optimizer_trace_limit	是	
optimizer_trace_max_mem_size	是	
optimizer_trace_offset	是	
performance-schema-consumer-events-waits-current	是	
performance-schema-instrument	是	
performance_schema	是	
performance_schema_accounts_size	是	
performance_schema_consumer_global_instrumentation	是	
performance_schema_consumer_thread_instrumentation	是	
performance_schema_consumer_events_stages_current	是	
performance_schema_consumer_events_stages_history	是	
performance_schema_consumer_events_stages_history_long	是	
performance_schema_consumer_events_statements_current	是	

参数名称	可修改	备注
performance_schema_consumer_events_statements_history	是	
performance_schema_consumer_events_statements_history_long	是	
performance_schema_consumer_events_waits_history	是	
performance_schema_consumer_events_waits_history_long	是	
performance_schema_consumer_statements_digest	是	
performance_schema_digests_size	是	
performance_schema_events_stages_history_long_size	是	
performance_schema_events_stages_history_size	是	
performance_schema_events_statements_history_long_size	是	
performance_schema_events_statements_history_size	是	
performance_schema_events_transactions_history_long_size	是	

参数名称	可修改	备注
performance_schema_events_transactions_history_size	是	
performance_schema_events_waits_history_long_size	是	
performance_schema_events_waits_history_size	是	
performance_schema_hosts_size	是	
performance_schema_max_cond_classes	是	
performance_schema_max_cond_instances	是	
performance_schema_max_digest_length	是	
performance_schema_max_file_classes	是	
performance_schema_max_file_handles	是	
performance_schema_max_file_instances	是	
performance_schema_max_index_stat	是	
performance_schema_max_memory_classes	是	
performance_schema_max_metadata_locks	是	

参数名称	可修改	备注
performance_schema_max_mutex_classes	是	
performance_schema_max_mutex_instances	是	
performance_schema_max_prepared_statements_instances	是	
performance_schema_max_program_instances	是	
performance_schema_max_rwlock_classes	是	
performance_schema_max_rwlock_instances	是	
performance_schema_max_socket_classes	是	
performance_schema_max_socket_instances	是	
performance_schema_max_sql_text_length	是	
performance_schema_max_stage_classes	是	
performance_schema_max_statement_classes	是	
performance_schema_max_statement_stack	是	
performance_schema_max_table_handles	是	

参数名称	可修改	备注
performance_schema_max_table_instances	是	
performance_schema_max_table_lock_stat	是	
performance_schema_max_thread_classes	是	
performance_schema_max_thread_instances	是	
performance_schema_session_connect_attrs_size	是	
performance_schema_setup_actors_size	是	
performance_schema_setup_objects_size	是	
performance_schema_show_processlist	是	此参数确定要使用哪个 SHOW PROCESSLIST 实现： • 原定设置实现是在持有全局互斥锁的同时，从线程管理器内部跨活动的线程进行迭代。这可能会导致性能降低，尤其是在繁忙的系统上。 • 备选 SHOW PROCESSLIST 实现基于性能架构 processlist 表。此实现从性能架构而不是线程管理器查询活动的线程数据，并且不需要互斥锁。 此参数适用于 Aurora MySQL 版本 2.12 及更高版本以及版本 3。

参数名称	可修改	备注
performance_schema_users_size	是	
pid_file	否	
plugin_dir	否	Aurora MySQL 使用不在其中直接访问文件系统的托管式实例。
port	否	Aurora MySQL 管理连接属性，并为集群中的所有数据库实例强制执行一致的设置。
preload_buffer_size	是	预加载索引时分配的缓冲区的大小。 此参数适用于 Aurora MySQL 版本 3。
profiling_history_size	是	
query_alloc_block_size	是	
query_cache_limit	是	已从 Aurora MySQL 版本 3 中删除。
query_cache_min_res_unit	是	已从 Aurora MySQL 版本 3 中删除。
query_cache_size	是	默认值由公式表示。有关公式中 DBInstanceClassMemory 值的计算方式详细信息，请参阅 数据库参数公式变量。 已从 Aurora MySQL 版本 3 中删除。
query_cache_type	是	已从 Aurora MySQL 版本 3 中删除。
query_cache_wlock_invalidate	是	已从 Aurora MySQL 版本 3 中删除。
query_prealloc_size	是	
range_alloc_block_size	是	

参数名称	可修改	备注
read_buffer_size	是	
read_only	是	启用此参数时，除副本线程执行的更新外，服务器不允许任何更新。 对于 Aurora MySQL 版本 2，有效值如下所示： • 0 – OFF • 1 – ON • {TrueIfReplica} – ON 适用于只读副本。这是默认值。 • {TrueIfClusterReplica} – ON 适用于副本集群中的实例，例如跨区域只读副本、Aurora Global Database 中的辅助集群以及蓝绿部署。 建议您在 Aurora MySQL 版本 2 中使用数据库集群参数组，以确保在故障转移时将 read_only 参数应用于新的写入器实例。 Note 读取器实例始终是只读的，因为 Aurora MySQL 在所有读取器上将 innodb_read_only 设置为 1。因此，read_only 在读取器实例上是冗余的。 已从 Aurora MySQL 版本 3 的实例级别删除。
read_rnd_buffer_size	是	

参数名称	可修改	备注
relay-log	否	
relay_log_info_repository	是	已从 Aurora MySQL 版本 3 中删除。
relay_log_recovery	否	
replica_checkpoint_group	是	Aurora MySQL 版本 3
replica_checkpoint_period	是	Aurora MySQL 版本 3
replica_parallel_workers	是	Aurora MySQL 版本 3
replica_pending_jobs_size_max	是	Aurora MySQL 版本 3
replica_skip_errors	是	Aurora MySQL 版本 3
replica_sql_verify_checksum	是	Aurora MySQL 版本 3
safe-user-create	是	
secure_auth	是	在 Aurora MySQL 版本 2 中，此参数始终处于开启状态。尝试将其关闭会生成错误。 已从 Aurora MySQL 版本 3 中删除。
secure_file_priv	否	Aurora MySQL 使用不在其中直接访问文件系统的托管式实例。
show_create_table_verbosity	是	启用此变量会导致 SHOW_CREATE_TABLE 显示 ROW_FORMAT ，而不管它是否为原定设置格式。 此参数适用于 Aurora MySQL 版本 2.12 及更高版本以及版本 3。
skip-slave-start	否	

参数名称	可修改	备注
skip_external_locking	否	
skip_show_database	是	
slave_checkpoint_group	是	Aurora MySQL 版本 2。在 Aurora MySQL 版本 3 中使用 replica_checkpoint_group 。
slave_checkpoint_period	是	Aurora MySQL 版本 2。在 Aurora MySQL 版本 3 中使用 replica_checkpoint_period 。
slave_parallel_workers	是	Aurora MySQL 版本 2。在 Aurora MySQL 版本 3 中使用 replica_parallel_workers 。
slave_pending_jobs_size_max	是	Aurora MySQL 版本 2。在 Aurora MySQL 版本 3 中使用 replica_pending_jobs_size_max 。
slave_sql_verify_checksum	是	Aurora MySQL 版本 2。在 Aurora MySQL 版本 3 中使用 replica_sql_verify_checksum 。
slow_launch_time	是	
slow_query_log	是	有关将日志上传到 CloudWatch Logs 的说明，请参阅 将 Amazon Aurora MySQL 日志发布到 Amazon CloudWatch Logs。
slow_query_log_file	否	Aurora MySQL 使用不在其中直接访问文件系统的托管式实例。
socket	否	
sort_buffer_size	是	

参数名称	可修改	备注
sql_mode	是	
sql_select_limit	是	
stored_program_cache	是	
sync_binlog	否	
sync_master_info	是	
sync_source_info	是	此参数适用于 Aurora MySQL 版本 3。
sync_relay_log	是	已从 Aurora MySQL 版本 3 中删除。
sync_relay_log_info	是	
sysdate-is-now	是	
table_cache_element_entry_ttl	否	
table_definition_cache	是	默认值由公式表示。有关公式中 DBInstanceClassMemory 值的计算方式详细信息，请参阅 数据库参数公式变量。
table_open_cache	是	默认值由公式表示。有关公式中 DBInstanceClassMemory 值的计算方式详细信息，请参阅 数据库参数公式变量。
table_open_cache_instances	是	
temp-pool	是	已从 Aurora MySQL 版本 3 中删除。
temptable_max_mmap	是	此参数适用于 Aurora MySQL 版本 3。有关详细信息，请参阅Aurora MySQL 版本 3 中的新临时表行为。

参数名称	可修改	备注
temptable_max_ram	是	此参数适用于 Aurora MySQL 版本 3。有关详细信息，请参阅Aurora MySQL 版本 3 中的新临时表行为。
temptable_use_mmap	是	此参数适用于 Aurora MySQL 版本 3。有关详细信息，请参阅Aurora MySQL 版本 3 中的新临时表行为。
thread_cache_size	是	要缓存的线程数。此参数适用于 Aurora MySQL 版本 2 和 3。
thread_handling	否	
thread_stack	是	
timed_mutexes	是	
tmp_table_size	是	在 Aurora MySQL 版本 3 中，定义由 MEMORY 存储引擎创建的内部内存中临时表的最大大小。 在 Aurora MySQL 版本 3.04 及更高版本中，定义当 aurora_tmptable_enable_per_table_limit 为 ON 时，由 TempTable 存储引擎创建的内部内存中临时表的最大大小。 有关更多信息，请参阅 限制内部内存中临时表的大小。
tmpdir	否	Aurora MySQL 使用不在其中直接访问文件系统的托管式实例。
transaction_alloc_block_size	是	

参数名称	可修改	备注
transaction_isolation	是	此参数适用于 Aurora MySQL 版本 3。它将代替 tx_isolation 。
transaction_prealloc_size	是	
tx_isolation	是	已从 Aurora MySQL 版本 3 中删除。它将替换为 transaction_isolation 。
updatable_views_with_limit	是	
validate-password	否	
validate_password_dictionary_file	否	
validate_password_length	否	
validate_password_mixed_case_count	否	
validate_password_number_count	否	
validate_password_policy	否	
validate_password_special_char_count	否	
wait_timeout	是	Aurora 会估计 interactive_timeout 和 wait_timeout 的最小值，然后使用这个最小值作为超时来结束所有空闲会话，包括交互式会话和非交互式会话。

不适用于 Aurora MySQL 的 MySQL 参数

由于 Aurora MySQL 与 MySQL 之间存在架构差异，有些 MySQL 参数不适用于 Aurora MySQL。

以下 MySQL 参数不适用于 Aurora MySQL。此列表并不详尽。

- `activate_all_roles_on_login` – 此参数不适用于 Aurora MySQL 版本 2。它在 Aurora MySQL 版本 3 中可用。
- `big_tables`
- `bind_address`
- `character_sets_dir`
- `innodb_adaptive_flushing`
- `innodb_adaptive_flushing_lwm`
- `innodb_buffer_pool_chunk_size`
- `innodb_buffer_pool_instances`
- `innodb_change_buffering`
- `innodb_checksum_algorithm`
- `innodb_data_file_path`
- `innodb_dedicated_server`
- `innodb_doublewrite`
- `innodb_flush_log_at_timeout` – 此参数不适用于 Aurora MySQL。有关更多信息，请参阅 配置刷新日志缓冲区的频率。
- `innodb_flush_method`
- `innodb_flush_neighbors`
- `innodb_io_capacity`
- `innodb_io_capacity_max`
- `innodb_log_buffer_size`
- `innodb_log_file_size`
- `innodb_log_files_in_group`
- `innodb_log_spin_cpu_abs_lwm`
- `innodb_log_spin_cpu_pct_hwm`
- `innodb_log_writer_threads`
- `innodb_max_dirty_pages_pct`

- innodb_numa_interleave
- innodb_page_size
- innodb_redo_log_capacity
- innodb_redo_log_encrypt
- innodb_undo_log_encrypt
- innodb_undo_log_truncate
- innodb_undo_logs
- innodb_undo_tablespaces
- innodb_use_native_aio
- innodb_write_io_threads

Aurora MySQL 全局状态变量

Aurora MySQL 包括来自社区 MySQL 的状态变量和 Aurora 独有的变量。您可以检查这些变量，来了解数据库引擎内部正在发生的事情。有关社区 MySQL 中状态变量的更多信息，请参阅社区 MySQL 8.0 文档中的 Server Status Variables。

您可以使用如下语句查找 Aurora MySQL 全局状态变量的当前值：

```
show global status like '%aurora%';
```

下表描述了 Aurora MySQL 使用的全局状态变量。

名称	描述
AuroraDb_commits	自上次重启以来的提交总数。
AuroraDb_commit_latency	自上次重启以来的聚合提交延迟。
AuroraDb_ddl_stmt_duration	自上次重启以来的聚合 DDL 延迟。
AuroraDb_select_stmt_duration	自上次重启以来的聚合 SELECT 语句延迟。
AuroraDb_insert_stmt_duration	自上次重启以来的聚合 INSERT 语句延迟。
AuroraDb_update_stmt_duration	自上次重启以来的聚合 UPDATE 语句延迟。

名称	描述
AuroraDb_delete_stmt_duration	自上次重启以来的聚合 DELETE 语句延迟。
Aurora_binlog_io_cache_allocated	分配给二进制日志 I/O 缓存的字节数。
Aurora_binlog_io_cache_read_requests	向二进制日志 I/O 缓存发出的读取请求数。
Aurora_binlog_io_cache_reads	从二进制日志 I/O 缓存提供的读取请求数。
Aurora_enhanced_binlog	指示是为此数据库实例启用还是禁用了增强型二进制日志。有关更多信息，请参阅 为 Aurora MySQL 设置增强型二进制日志。
Aurora_external_connection_count	与数据库实例的数据库连接数，不包括用于数据库运行状况检查的 RDS 服务连接。
Aurora_fast_insert_cache_hits	在成功检索和验证缓存游标时递增的计数器。有关快速插入缓存的更多信息，请参阅Amazon Aurora MySQL 性能增强。
Aurora_fast_insert_cache_misses	当缓存游标不再有效且 Aurora 执行常规索引遍历时递增的计数器。有关快速插入缓存的更多信息，请参阅Amazon Aurora MySQL 性能增强。
Aurora_fts_cache_memory_used	InnoDB 全文搜索系统所使用的内存量（以字节为单位）。此变量适用于 Aurora MySQL 版本 3.07 及更高版本。
Aurora_fwd_master_dml_stmt_count	转发到此写入器数据库实例的 DML 语句总数。此变量适用于 Aurora MySQL 版本 2。
Aurora_fwd_master_dml_stmt_duration	转发到此写入器数据库实例的 DML 语句的总持续时间。此变量适用于 Aurora MySQL 版本 2。
Aurora_fwd_master_errors_rpc_timeout	在写入器上建立转发连接失败的次数。

名称	描述
Aurora_fwd_master_errors_session_limit	由于写入器上的 session full 而被拒绝的转发查询数。
Aurora_fwd_master_errors_session_timeout	由于写入器超时而结束转发会话的次数。
Aurora_fwd_master_open_sessions	写入器数据库实例上的转发会话数。此变量适用于 Aurora MySQL 版本 2。
Aurora_fwd_master_select_stmt_count	转发到此写入器数据库实例的 SELECT 语句总数。此变量适用于 Aurora MySQL 版本 2。
Aurora_fwd_master_select_stmt_duration	转发到此写入器数据库实例的 SELECT 语句的总持续时间。此变量适用于 Aurora MySQL 版本 2。
Aurora_fwd_writer_dml_stmt_count	转发到此写入器数据库实例的 DML 语句总数。此变量适用于 Aurora MySQL 版本 3。
Aurora_fwd_writer_dml_stmt_duration	转发到此写入器数据库实例的 DML 语句的总持续时间。此变量适用于 Aurora MySQL 版本 3。
Aurora_fwd_writer_errors_rpc_timeout	在写入器上建立转发连接失败的次数。
Aurora_fwd_writer_errors_session_limit	由于写入器上的 session full 而被拒绝的转发查询数。
Aurora_fwd_writer_errors_session_timeout	由于写入器超时而结束转发会话的次数。
Aurora_fwd_writer_open_sessions	写入器数据库实例上的转发会话数。此变量适用于 Aurora MySQL 版本 3。
Aurora_fwd_writer_select_stmt_count	转发到此写入器数据库实例的 SELECT 语句总数。此变量适用于 Aurora MySQL 版本 3。

名称	描述
Aurora_fwd_writer_select_stmt_duration	转发到此写入器数据库实例的 SELECT 语句的总持续时间。此变量适用于 Aurora MySQL 版本 3。
Aurora_lockmgr_buffer_pool_memory_used	Aurora MySQL 锁定管理器正在使用的缓冲池内存量（以字节为单位）。
Aurora_lockmgr_memory_used	Aurora MySQL 锁定管理器正在使用的内存量（以字节为单位）。
Aurora_ml_actual_request_cnt	在数据库实例用户运行的所有查询中，Aurora MySQL 对 Aurora 机器学习服务发出的请求次数总计。有关更多信息，请参阅 将 Amazon Aurora 机器学习与 Aurora MySQL 结合使用。
Aurora_ml_actual_response_cnt	在数据库实例用户运行的所有查询中，Aurora MySQL 从 Aurora 机器学习服务接收的响应次数总计。有关更多信息，请参阅 将 Amazon Aurora 机器学习与 Aurora MySQL 结合使用。
Aurora_ml_cache_hit_cnt	在数据库实例用户运行的所有查询中，Aurora MySQL 从 Aurora 机器学习服务接收的内部缓存命中次数总计。有关更多信息，请参阅 将 Amazon Aurora 机器学习与 Aurora MySQL 结合使用。
Aurora_ml_logical_request_cnt	自上次状态重置以来，数据库实例评估的要发送到 Aurora 机器学习服务的逻辑请求数。根据是否使用了批处理，此值可能高于 Aurora_ml_actual_request_cnt 。有关更多信息，请参阅 将 Amazon Aurora 机器学习与 Aurora MySQL 结合使用。

名称	描述
Aurora_ml_logical_response_cnt	在数据库实例用户运行的所有查询中，Aurora MySQL 从 Aurora 机器学习服务接收的响应次数总计。有关更多信息，请参阅 将 Amazon Aurora 机器学习与 Aurora MySQL 结合使用。
Aurora_ml_retry_request_cnt	自上次状态重置以来，数据库实例已向 Aurora 机器学习服务发送的重试请求数。有关更多信息，请参阅 将 Amazon Aurora 机器学习与 Aurora MySQL 结合使用。
Aurora_ml_single_request_cnt	在数据库实例用户运行的所有查询中，非批处理模式评估的 Aurora 机器学习函数总计。有关更多信息，请参阅 将 Amazon Aurora 机器学习与 Aurora MySQL 结合使用。
aurora_oom_avoidance_recovery_state	表示对于此数据库实例，Aurora 内存不足（OOM，Out-Of-Memory）避免恢复处于 ACTIVE 还是 INACTIVE 状态。
aurora_oom_reserved_mem_enter_kb	表示在 Aurora 的 OOM 处理机制中，进入 RESERVED 状态的阈值。 当服务器上的可用内存低于此阈值时，aurora_oom_status 会变为 RESERVED，表示服务器的内存使用量已接近临界水平。
aurora_oom_reserved_mem_exit_kb	表示在 Aurora 的 OOM 处理机制中，退出 RESERVED 状态的阈值。 当服务器上的可用内存上升超过此阈值时，aurora_oom_status 将恢复为 NORMAL，表示服务器已恢复到更稳定的状态，具有足够的内存资源。

名称	描述
aurora_oom_status	表示此数据库实例的当前 OOM 状态。当该值为 NORMAL 时，表示有足够的内存资源。 如果该值更改为 RESERVED，则表示服务器的可用内存不足。根据 aurora_oom_response 参数配置来采取操作。 有关更多信息，请参阅 排查 Aurora MySQL 数据库内存不足问题。
Aurora_pq_bytes_returned	在并行查询期间传输到头节点的元组数据结构的字节数。除以 16,384 以与 Aurora_pq_pages_pushed_down 进行比较。
Aurora_pq_max_concurrent_requests	可以在该 Aurora 数据库实例上并发运行的最大并行查询会话数。这是一个取决于 AWS 数据库实例类的固定数字。
Aurora_pq_pages_pushed_down	并行查询避免通过网络传输到头节点的数据页面数量（每个页面具有 16 KiB 的固定大小）。
Aurora_pq_request_attempted	请求的并行查询会话数。该值可能表示每个查询具有多个会话，具体取决于 SQL 结构，如子查询和联接。
Aurora_pq_request_executed	成功运行的并行查询会话数。
Aurora_pq_request_failed	向客户端返回错误的并行查询会话数。在某些情况下，并行查询请求可能会失败，例如，由于在存储层中出现问题。在这些情况下，将使用非并行查询机制重试失败的查询部分。如果重试的查询也失败，则会向客户端返回错误并增加该计数器。

名称	描述
Aurora_pq_request_in_progress	当前运行的并行查询会话数。该数字适用于您连接到的特定 Aurora 数据库实例，而不适用于整个 Aurora 数据库集群。要查看数据库实例是否接近其并发限制，请将该值与 Aurora_pq_max_concurrent_requests 进行比较。
Aurora_pq_request_not_chosen	未选择并行查询以满足查询条件的次数。该值是几个其他更精细的计数器的总和。即使没有实际执行查询，EXPLAIN 语句也可能增加此计数器。
Aurora_pq_request_not_chosen_below_min_rows	由于表中的行数而未选择并行查询的次数。即使没有实际执行查询，EXPLAIN 语句也可能增加此计数器。
Aurora_pq_request_not_chosen_column_bit	由于投影列的列表中的数据类型不受支持而使用非并行查询处理路径的并行查询请求数。
Aurora_pq_request_not_chosen_column_geometry	由于表具有 GEOMETRY 数据类型的列而使用非并行查询处理路径的并行查询请求数。有关删除此限制的 Aurora MySQL 版本的信息，请参阅将并行查询集群升级到 Aurora MySQL 版本 3。
Aurora_pq_request_not_chosen_column_lob	由于表具有 LOB 数据类型的列或具有（由于声明的长度）而在外部存储的 VARCHAR 列，因此使用非并行查询处理路径的并行查询请求数。有关删除此限制的 Aurora MySQL 版本的信息，请参阅 将并行查询集群升级到 Aurora MySQL 版本 3。
Aurora_pq_request_not_chosen_column_virtual	由于表包含虚拟列而使用非并行查询处理路径的并行查询请求数。
Aurora_pq_request_not_chosen_custom_charset	由于表具有带自定义字符集的列而使用非并行查询处理路径的并行查询请求数。

名称	描述
Aurora_pq_request_not_chosen_fast_ddl	由于表当前正在被快速 DDL ALTER 语句更改而使用非并行查询处理路径的并行查询请求数。
Aurora_pq_request_not_chosen_few_pages_outside_buffer_pool	由于没有足够的未缓冲表数据以值得运行并行查询而未选择并行查询的次数，即使缓冲池中的表数据少于 95%。
Aurora_pq_request_not_chosen_full_text_index	由于表具有全文索引而使用非并行查询处理路径的并行查询请求数。
Aurora_pq_request_not_chosen_high_buffer_pool_pct	由于在缓冲池中具有较高比例的表数据（目前大于 95%）而未选择并行查询的次数。在这些情况下，优化程序确定从缓冲池中读取数据更高效。即使没有实际执行查询，EXPLAIN 语句也可能增加此计数器。
Aurora_pq_request_not_chosen_index_hint	由于查询包含索引提示而使用非并行查询处理路径的并行查询请求数。
Aurora_pq_request_not_chosen_innodb_table_format	由于表使用不受支持的 InnoDB 行格式，因此使用非并行查询处理路径的并行查询请求数。Aurora 并行查询仅适用于 COMPACT、REDUNDANT 和 DYNAMIC 行格式。
Aurora_pq_request_not_chosen_long_trx	由于正在长时间运行的事务中启动查询而使用非并行查询处理路径的并行查询请求数。即使没有实际执行查询，EXPLAIN 语句也可能增加此计数器。
Aurora_pq_request_not_chosen_no_where_clause	由于查询不包含任何 WHERE 子句而使用非并行查询处理路径的并行查询请求数。
Aurora_pq_request_not_chosen_range_scan	由于查询对索引使用范围扫描而使用非并行查询处理路径的并行查询请求数。

名称	描述
Aurora_pq_request_not_chosen_row_length_too_long	由于所有列的总组合长度过长而使用非并行查询处理路径的并行查询请求数。
Aurora_pq_request_not_chosen_small_table	由于表的总大小（由行数和平均行长度确定）而未选择并行查询的次数。即使没有实际执行查询，EXPLAIN 语句也可能增加此计数器。
Aurora_pq_request_not_chosen_temporary_table	由于查询引用了临时表（这些临时表使用不受支持的 MyISAM 或 memory 表类型）而使用非并行查询处理路径的并行查询请求数。
Aurora_pq_request_not_chosen_tx_isolation	由于查询使用不受支持的事务隔离级别而使用非并行查询处理路径的并行查询请求数。在读取器数据库实例上，并行查询仅适用于 REPEATABLE READ 和 READ COMMITTED 隔离级别。
Aurora_pq_request_not_chosen_update_delete_stmts	由于查询是 UPDATE 或 DELETE 语句的一部分而使用非并行查询处理路径的并行查询请求数。
Aurora_pq_request_not_chosen_unsupported_access	由于 WHERE 子句不符合并行查询条件而使用非并行查询处理路径的并行查询请求数。如果查询不需要数据密集型扫描，或者查询是 DELETE 或 UPDATE 语句，则会出现该结果。
Aurora_pq_request_not_chosen_unsupported_storage_type	由于 Aurora MySQL 数据库集群未使用支持的 Aurora 集群存储配置，而使用非并行查询处理路径的并行查询请求的数量。有关更多信息，请参阅 限制。 此参数适用于 Aurora MySQL 版本 3.04 及更高版本。
Aurora_pq_request_throttled	由于在特定 Aurora 数据库实例上已运行的最大并发并行查询数而未选择并行查询的次数。

名称	描述
Aurora_repl_bytes_received	自上次重启以来复制到 Aurora MySQL 读取器数据库实例的字节数。有关更多信息，请参阅 使用 Amazon Aurora MySQL 进行复制。
Aurora_reserved_mem_exceeded_incidents	自上次重启以来引擎超过预留内存限制的次数。如果配置了 aurora_oom_response ，则此阈值定义何时触发避免内存不足（OOM）的活动。有关 Aurora MySQL OOM 响应的更多信息，请参阅排查 Aurora MySQL 数据库内存不足问题。
Aurora_thread_pool_thread_count	Aurora 线程池中的当前线程数。有关 Aurora MySQL 中线程池的更多信息，请参阅线程池。
Aurora_tmz_version	表示数据库集群使用的时区信息的当前版本。这些值遵循互联网号码分配机构（IANA）格式：YYYYsuffix ，例如 2022a 和 2023c。 此参数适用于 Aurora MySQL 版本 2.12 及更高版本和版本 3.04 及更高版本。
Aurora_zdr_oom_threshold	表示内存阈值，以千字节（KB）为单位，达到该阈值时，Aurora 数据库实例将启动零停机重启（ZDR，Zero Downtime Restart），以便从潜在的内存相关问题中恢复。
server_aurora_das_running	表示在此数据库实例上是启用还是禁用了数据库活动流（DAS）。有关更多信息，请参阅 使用数据库活动流监控 Amazon Aurora。

不适用于 Aurora MySQL 的 MySQL 状态变量

由于 Aurora MySQL 与 MySQL 之间存在架构差异，有些 MySQL 状态变量不适用于 Aurora MySQL。

以下 MySQL 状态变量不适用于 Aurora MySQL。此列表并不详尽。

- innodb_buffer_pool_bytes_dirty

- innodb_buffer_pool_pages_dirty
- innodb_buffer_pool_pages_flushed

Aurora MySQL 版本 3 删除了 Aurora MySQL 版本 2 中的以下状态变量：

- AuroraDb_lockmgr_bitmaps0_in_use
- AuroraDb_lockmgr_bitmaps1_in_use
- AuroraDb_lockmgr_bitmaps_mem_used
- AuroraDb_thread_deadlocks
- available_alter_table_log_entries
- Aurora_lockmgr_memory_used
- Aurora_missing_history_on_replica_incidents
- Aurora_new_lock_manager_lock_release_cnt
- Aurora_new_lock_manager_lock_release_total_duration_micro
- Aurora_new_lock_manager_lock_timeout_cnt
- Aurora_total_op_memory
- Aurora_total_op_temp_space
- Aurora_used_alter_table_log_entries
- Aurora_using_new_lock_manager
- Aurora_volume_bytes_allocated
- Aurora_volume_bytes_left_extent
- Aurora_volume_bytes_left_total
- Com_alter_db_upgrade
- Compression
- External_threads_connected
- Innodb_available_undo_logs
- Last_query_cost
- Last_query_partial_plans
- Slave_heartbeat_period
- Slave_last_heartbeat
- Slave_received_heartbeats

- Slave_retried_transactions
- Slave_running
- Time_since_zero_connections

这些 MySQL 状态变量在 Aurora MySQL 版本 2 中可用，但它们在 Aurora MySQL 版本 3 中不可用：

- Innodb_redo_log_enabled
- Innodb_undo_tablespaces_total
- Innodb_undo_tablespaces_implicit
- Innodb_undo_tablespaces_explicit
- Innodb_undo_tablespaces_active

Aurora MySQL 等待事件

以下是 Aurora MySQL 的一些常见等待事件。

> **Note**
>
> 有关使用等待事件调整 Aurora MySQL 性能的信息，请参阅使用等待事件优化 Aurora MySQL。
> 有关 MySQL 等待事件中使用的命名约定的信息，请参阅 MySQL 文档中的性能架构测试命名约定。

cpu

准备运行的活动连接数一直高于 vCPU 的数量。有关更多信息，请参阅 cpu。

io/aurora_redo_log_flush

会话将数据持久存储到 Aurora 存储。通常，该等待事件针对 Aurora MySQL 中的写入 I/O 操作。有关更多信息，请参阅 io/aurora_redo_log_flush。

io/aurora_respond_to_client

以下 Aurora MySQL 版本的查询处理已完成，结果将返回到应用程序客户端：2.10.2 及更高的 2.10 版本、2.09.3 及更高的 2.09 版本和 2.07.7 及更高的 2.07 版本。将数据库实例类的网络带宽与返回的结果集的大小进行比较。另外，请检查客户端响应时间。如果客户端无响应且无法处理 TCP 数

据包，则可能会发生数据包丢弃和 TCP 重新传输。这种情况会对网络带宽产生负面影响。在低于 2.10.2、2.09.3 和 2.07.7 的版本中，等待事件错误地包含空闲时间。要了解如何在此等待突出时优化数据库，请参阅 io/aurora_respond_to_client。

io/file/csv/data

线程以逗号分隔值 (CSV) 格式写入表。检查您的 CSV 表使用情况。此事件的典型原因是在表上设置 `log_output`。

io/file/sql/binlog

线程在等待正写入磁盘的二进制日志 (binlog) 文件。

io/redo_log_flush

会话将数据持久存储到 Aurora 存储。通常，该等待事件针对 Aurora MySQL 中的写入 I/O 操作。有关更多信息，请参阅 io/redo_log_flush。

io/socket/sql/client_connection

`mysqld` 程序正忙于创建线程来处理传入的新客户端连接。有关更多信息，请参阅 io/socket/sql/client_connection。

io/table/sql/handler

引擎正在等待访问表格。无论数据是缓存在缓冲池中还是可在磁盘上访问，都会发生此事件。有关更多信息，请参阅 io/table/sql/handler。

lock/table/sql/handler

该等待事件是表锁定等待事件处理程序。有关性能架构中的“原子”和“分子”事件的更多信息，请参阅 MySQL 文档中的性能架构原子和分子事件。

synch/cond/innodb/row_lock_wait

多个数据操作语言 (DML) 语句同时访问相同的数据库行。有关更多信息，请参阅 synch/cond/innodb/row_lock_wait。

synch/cond/innodb/row_lock_wait_cond

多条 DML 语句同时访问相同的数据库行。有关更多信息，请参阅 synch/cond/innodb/row_lock_wait_cond。

synch/cond/sql/MDL_context::COND_wait_status

线程正等待表元数据锁定。引擎使用这种类型的锁定来管理对数据库架构的并发访问并确保数据的一致性。有关更多信息，请参阅 MySQL 文档中的优化锁定操作。要了解如何在此事件突出时优化数据库，请参阅 synch/cond/sql/MDL_context::COND_wait_status。

synch/cond/sql/MYSQL_BIN_LOG::COND_done

您已开启二进制日志记录。可能存在较高的提交吞吐量、大量事务提交或读取二进制日志的副本。考虑使用多行语句或将语句捆绑到一个事务中。在 Aurora 中，使用全局数据库而不是二进制日志复制，或者使用 `aurora_binlog_*` 参数。

synch/mutex/innodb/aurora_lock_thread_slot_futex

多条 DML 语句同时访问相同的数据库行。有关更多信息，请参阅 synch/mutex/innodb/aurora_lock_thread_slot_futex。

synch/mutex/innodb/buf_pool_mutex

缓冲池不够大，无法容纳正常工作的数据集。或者，工作负载访问特定表中的页面，这会导致缓冲池中的争用。有关更多信息，请参阅 synch/mutex/innodb/buf_pool_mutex。

synch/mutex/innodb/fil_system_mutex

该进程正在等待对表空间内存缓存的访问。有关更多信息，请参阅 synch/mutex/innodb/fil_system_mutex。

synch/mutex/innodb/trx_sys_mutex

操作正在以一致或受控的方式在 InnoDB 中检查、更新、删除或添加事务 ID。这些操作需要 `trx_sys` 互斥调用，该调用由性能架构工具跟踪。操作包括在数据库启动或关闭时管理事务系统、回滚、撤消清理、行读取访问和缓冲池加载。高数据库负载和大量事务导致此等待事件频繁出现。有关更多信息，请参阅 synch/mutex/innodb/trx_sys_mutex。

synch/mutex/mysys/KEY_CACHE::cache_lock

`keycache->cache_lock` 互斥控制对 MyISAM 表的密钥缓存的访问。虽然 Aurora MySQL 不允许使用 MyISAM 表来存储持久数据，但它们用于存储内部临时表。考虑检查 `created_tmp_tables` 或 `created_tmp_disk_tables` 状态计数器，因为在某些情况下，当临时表不再适合放入内存中时，会将其写入磁盘。

synch/mutex/sql/FILE_AS_TABLE::LOCK_offsets

在打开或创建表元数据文件时，引擎会获取此互斥。当此等待事件发生频率过高时，创建或打开的表的数量会激增。

synch/mutex/sql/FILE_AS_TABLE::LOCK_shim_lists

引擎在跟踪打开的表的内部结构上执行以下操作（如 `reset_size`、`detach_contents` 或 `add_contents`）时获取此互斥。该互斥可同步对列表内容的访问。当此等待事件以高频发生时，它表示之前访问的表集突然发生变化。引擎需要访问新表或放弃与之前访问的表相关的上下文。

synch/mutex/sql/LOCK_open

会话打开的表数超过了表定义缓存或表打开缓存的大小。增加这些缓存的大小。有关更多信息，请参阅 MySQL 如何打开和关闭表。

synch/mutex/sql/LOCK_table_cache

会话打开的表数量超过了表定义缓存或表打开缓存的大小。增加这些缓存的大小。有关更多信息，请参阅 MySQL 如何打开和关闭表。

synch/mutex/sql/LOG

在该等待事件中，有正等待日志锁定的线程。例如，线程可能等待锁定写入慢速查询日志文件。

synch/mutex/sql/MYSQL_BIN_LOG::LOCK_commit

在该等待事件中，有正等待带着提交到二进制日志的意图获取锁定的线程。二进制日志记录争用可能出现在更改率非常高的数据库上。根据您的 MySQL 版本，有特定锁定用于保护二进制日志的一致性和持续性。在 RDS for MySQL 中，二进制日志用于复制和自动备份过程。在 Aurora MySQL 中，本机复制或备份不需要二进制日志。它们默认情况下处于禁用状态，但可以启用或用于外部复制或更改数据捕获。有关更多信息，请参阅 MySQL 文档中的二进制日志。

sync/mutex/sql/MYSQL_BIN_LOG::LOCK_dump_thread_metrics_collection

如果打开了二进制日志记录，则当引擎将活动转储线程指标打印到引擎错误日志和内部操作映射时，将获得此互斥。

sync/mutex/sql/MYSQL_BIN_LOG::LOCK_inactive_binlogs_map

如果打开了二进制日志记录，引擎在添加、删除或搜索最新二进制日志文件后面的二进制日志文件列表时获取此互斥。

sync/mutex/sql/MYSQL_BIN_LOG::LOCK_io_cache

如果打开二进制日志记录，引擎将在以下 Aurora 二进制日志 IO 缓存操作期间获取此互斥：分配、调整大小、释放、写入、读取、清除和访问缓存信息。如果此事件频繁发生，则引擎在访问存储二进制日志事件的缓存。为了减少等待时间，请减少提交。尝试将多个语句分组到一个事务中。

synch/mutex/sql/MYSQL_BIN_LOG::LOCK_log

您已开启二进制日志记录。可能存在较高的提交吞吐量、很多事务提交或读取二进制日志的副本。考虑使用多行语句或将语句捆绑到一个事务中。在 Aurora 中，使用全局数据库而不是二进制日志复制，或使用 `aurora_binlog_*` 参数。

synch/mutex/sql/SERVER_THREAD::LOCK_sync

互斥锁 SERVER_THREAD::LOCK_sync 在调度、处理或启动线程以进行文件写入的过程中获取。此等待事件发生过多表示数据库中的写入活动增加。

synch/mutex/sql/TABLESPACES:lock

引擎在以下表空间操作期间获取 TABLESPACES:lock 互斥：创建、删除、截断和扩展。此等待事件发生过多表示表空间操作频率很高。例如，将大量数据加载到数据库中。

synch/rwlock/innodb/dict

在该等待事件中，有正等待 InnoDB 数据字典中保留的 rwlock 的线程。

synch/rwlock/innodb/dict_operation_lock

在该等待事件中，有在 InnoDB 数据字典操作中保留锁定的线程。

synch/rwlock/innodb/dict sys RW lock

同时触发数据定义语言代码 (DDL) 中的大量并发数据控制语言语句 (DCL)。在常规应用程序活动期间，减少应用程序对 DDL 的依赖。

synch/rwlock/innodb/index_tree_rw_lock

大量类似的数据操作语言 (DML) 语句同时访问相同的数据库对象。尝试使用多行语句。此外，将工作负载分散到不同的数据库对象上。例如，实施分区。

synch/sxlock/innodb/dict_operation_lock

同时触发数据定义语言代码 (DDL) 中的大量并发数据控制语言语句 (DCL)。在常规应用程序活动期间，减少应用程序对 DDL 的依赖。

synch/sxlock/innodb/dict_sys_lock

同时触发数据定义语言代码 (DDL) 中的大量并发数据控制语言语句 (DCL)。在常规应用程序活动期间，减少应用程序对 DDL 的依赖。

synch/sxlock/innodb/hash_table_locks

会话在缓冲池中找不到页面。引擎要么需要读取文件，要么需要修改缓冲池的最近使用最少 (LRU) 列表。考虑增加缓冲区缓存大小并改进相关查询的访问路径。

synch/sxlock/innodb/index_tree_rw_lock

很多类似的数据操作语言 (DML) 语句同时访问相同的数据库对象。尝试使用多行语句。此外，将工作负载分散到不同的数据库对象上。例如，实施分区。

有关同步等待事件问题排查的更多信息，请参阅[为什么我的 MySQL 数据库实例在 Performance Insights 中的 SYNCH 等待事件显示有大量活动会话正在等待？]

Aurora MySQL 线程状态

以下是 Aurora MySQL 的一些常见的线程状态。

检查权限

线程正在检查服务器是否具有运行语句所需的权限。

检查查询缓存以进行查询

服务器正在检查查询缓存中是否存在当前查询。

已清除

这是连接的最终状态，其工作已完成但客户端尚未关闭。最好的解决方案是在代码中明确关闭连接。或者您可以在参数组中设置较低的 `wait_timeout` 值。

关闭表

线程正在将更改后的表数据刷新到磁盘并关闭使用过的表。如果这不是快速操作，请根据实例类网络带宽验证网络带宽消耗指标。另外，请检查 `table_open_cache` 和 `table_definition_cache` 参数的参数值是否允许同时打开足够的表，以使引擎不需要频繁地打开和关闭表。这些参数会影响实例的内存消耗。

将 HEAP 转换为 MyISAM

该查询正在将临时表从内存中的表转换为磁盘上的表。这种转换是必要的，因为 MySQL 在查询处理的中间步骤中创建的临时表对内存来说太大了。检查 `tmp_table_size` 和 `max_heap_table_size` 的值。在更高版本中，此线程状态名称为 `converting HEAP to ondisk`。

将 HEAT 转换为磁盘上

线程正在将内部临时表从内存中的表转换为磁盘上的表。

复制到 tmp 表

线程正在处理 `ALTER TABLE` 语句。此状态发生在具有新结构的表创建之后，但在将行复制到该表中之前。对于处于此状态的线程，您可以使用性能架构获取有关复制操作进度的信息。

创建排序索引

Aurora MySQL 正在执行一种排序，因为它不能使用现有的索引来满足查询的 `ORDER BY` 或 `GROUP BY` 子句。有关更多信息，请参阅 [创建排序索引]。

创建表

该线程正在创建一个永久表或临时表。

延迟提交确定完成

Aurora MySQL 中的异步提交已收到确认并已完成。

延迟提交确定启动

Aurora MySQL 线程已启动异步提交过程，但正在等待确认。这通常是事务的真正提交时间。

延迟发送确定完成

在向客户端发送响应时，可以释放绑定到连接的 Aurora MySQL 工作线程。线程可以开始其他工作。状态 `delayed send ok` 意味着对客户端的异步确认已完成。

延迟发送确定已启动

Aurora MySQL 工作线程已异步向客户端发送响应，现在可以自由地为其他连接工作。事务已启动一个尚未确认的异步提交过程。

executing

线程已经开始运行语句。

释放项目

线程已运行命令。在此状态下完成的一些项目释放涉及到查询缓存。这种状态之后通常是清理。

init

此状态发生在 `ALTER TABLE`、`DELETE`、`INSERT`、`SELECT` 或者 `UPDATE` 语句的初始化之前。处于此状态的操作包括刷新二进制日志或 InnoDB 日志，以及清理查询缓存。

源已将所有二进制日志发送给副本；正在等待更多更新

主节点已完成其复制的一部分。线程正在等待更多查询运行，以便可以写入二进制日志 (binlog)。

打开表

线程正在尝试打开一张表。除非 `ALTER TABLE` 或者 `LOCK TABLE` 语句需要完成，或者它超出了 `table_open_cache` 的值，否则此操作会快速完成。

正在优化

服务器正在对查询执行初始优化。

正在准备

在查询优化期间会出现此状态。

查询结束

此状态发生在处理查询之后但在释放项目状态之前。

删除重复项

Aurora MySQL 无法在查询的早期阶段优化 DISTINCT 操作。Aurora MySQL 必须先删除所有重复的行，然后才能将结果发送给客户端。

搜索行以进行更新

在更新它们之前，线程会查找所有匹配的行。如果 UPDATE 正在更改引擎用来查找行的索引，这个阶段有必要。

向从节点发送二进制日志事件

线程从二进制日志中读取事件并将其发送到副本。

向客户端发送缓存的结果

服务器正在从查询缓存中获取查询的结果并将其发送到客户端。

发送数据

线程正在读取和处理 SELECT 语句的行，但尚未开始向客户端发送数据。该过程是确定哪些页面包含满足查询所需的结果。有关更多信息，请参阅 发送数据。

发送给客户端

服务器正在向客户端写入数据包。在早期的 MySQL 版本中，此等待事件被标记为 `writing to net`。

starting

这是语句执行开始的第一阶段。

statistics

服务器正在计算统计数据以制定查询执行计划。如果线程长期处于此状态，则在执行其他工作时，服务器可能会绑定磁盘。

将结果存储在查询缓存中

服务器正在将查询结果存储在查询缓存中。

系统锁定

线程已调用 `mysql_lock_tables`，但是自调用以来，线程状态尚未更新。出现这种普遍状态的原因很多。

更新

线程正在准备开始更新表格。

更新

线程正在搜索行并更新它们。

用户锁定

该线程发出 GET_LOCK 调用。该线程在请求一个咨询锁并在等待该锁，或者计划请求咨询锁。

等待更多更新

主节点已完成其复制的一部分。线程正在等待更多查询运行，以便可以写入二进制日志 (binlog)。

等待架构元数据锁定

这是等待元数据锁定。

等待存储的函数元数据锁定

这是等待元数据锁定。

等待存储的过程元数据锁定

这是等待元数据锁定。

等待表刷新

线程正在执行 FLUSH TABLES 并且正在等待所有线程关闭他们的表。或者线程收到通知，指示表格的底层结构发生了变化，因此它必须重新打开表格以获得新结构。要重新打开表格，线程必须等到所有其他线程都关闭表格。如果另一个线程使用了表格上的以下语句之一，则会发出此通知：FLUSH TABLES、ALTER TABLE、RENAME TABLE、REPAIR TABLE、ANALYZE TABLE 或 OPTIMIZE TABLE。

等待表级锁定

一个会话在表上保持锁定，而另一个会话则试图在同一个表上获取同一个锁定。

等待表元数据锁定

Aurora MySQL 使用元数据锁定来管理对数据库对象的并发访问并确保数据一致性。在此等待事件中，一个会话在表上保持元数据锁定，而另一个会话则试图在同一个表上获取同一个锁定。启用性能架构后，此线程状态将报告为等待事件 synch/cond/sql/MDL_context::COND_wait_status。

写入网络

服务器正在向网络写入数据包。在以后的 MySQL 版本中，此等待事件被标记为 `Sending to client`。

Aurora MySQL 隔离级别

了解 Aurora MySQL 集群中的数据库实例如何实现隔离的数据库属性。本主题说明 Aurora MySQL 默认行为如何在严格一致性和高性能之间取得平衡。您可以根据工作负载的特性，使用这些信息帮助您决定何时更改默认。

写入器实例的可用隔离级别

您可以在 Aurora MySQL 数据库集群的主实例上使用隔离级别 `REPEATABLE READ`、`READ COMMITTED`、`READ UNCOMMITTED` 和 `SERIALIZABLE`。这些隔离级别在 Aurora MySQL 中的工作方式与在 RDS for MySQL 中的工作方式相同。

读取器实例的 REPEATABLE READ 隔离级别

默认情况下，配置为只读 Aurora 副本的 Aurora MySQL 数据库实例始终使用 `REPEATABLE READ` 隔离级别。这些数据库实例会忽略任何 `SET TRANSACTION ISOLATION LEVEL` 语句并继续使用 `REPEATABLE READ` 隔离级别。

读取器实例的 READ COMMITTED 隔离级别

如果您的应用程序包括主实例上的写入密集型工作负载和 Aurora 副本上的长时间运行的查询，则可能会产生大量的清除滞后。当内部垃圾回收被长时间运行的查询阻止时，就会发生清除滞后。您看到的症状是 `SHOW ENGINE INNODB STATUS` 命令输出中的 `history list length` 值很高。可以使用 CloudWatch 中的 `RollbackSegmentHistoryListLength` 指标监控该值。大量的清除滞后可能会降低二级索引的效用，降低整体查询性能并导致浪费存储空间。

如果遇到此类问题，可以设置 Aurora MySQL 会话级别配置设置 `aurora_read_replica_read_committed`，以在 Aurora 副本上使用 `READ COMMITTED` 隔离级别。应用此设置时，可以帮助减少在修改表的事务的同时执行长时间运行的查询可能导致的速度下降和空间浪费情况。

建议您在使用此设置之前一定要了解 `READ COMMITTED` 隔离的特定 Aurora MySQL 行为。Aurora 副本 `READ COMMITTED` 行为符合 ANSI SQL 标准。但是，隔离没有您可能熟悉的典型 MySQL `READ COMMITTED` 行为那么严格。因此，Aurora MySQL 只读副本上 `READ COMMITTED` 之下的查询结果与

您可能看到的 Aurora MySQL 主实例或 RDS for MySQL 上 READ COMMITTED 之下的同一查询的结果可能不同。您可以考虑将 aurora_read_replica_read_committed 设置用于诸如扫描超大型数据库的综合报告之类的情况。相比之下，在精度和可重复性很重要的小型结果集的短查询中，您可能会避免使用它。

READ COMMITTED 隔离级别不适用于 Aurora 全局数据库的辅助集群中使用写入转发特征的会话。有关写入转发的信息，请参阅 在 Amazon Aurora Global Database 中使用写入转发。

对读取器使用 READ COMMITTED

要对 Aurora 副本使用 READ COMMITTED 隔离级别，请将 aurora_read_replica_read_committed 配置设置设为 ON。在连接到特定的 Aurora 副本时，在会话级别使用此设置。为此，请运行以下 SQL 命令。

```
set session aurora_read_replica_read_committed = ON;
set session transaction isolation level read committed;
```

您可能会临时使用此配置设置，以执行交互式、一次性查询。您可能还想运行一个从 READ COMMITTED 隔离级别中受益的报告或数据分析应用程序，而其他应用程序的默认保持不变。

开启 aurora_read_replica_read_committed 设置后，使用 SET TRANSACTION ISOLATION LEVEL 命令为适当的事务指定隔离级别。

```
set transaction isolation level read committed;
```

Aurora 副本上的 READ COMMITTED 行为差异

aurora_read_replica_read_committed 设置使 READ COMMITTED 隔离级别可用于 Aurora 副本，并具有针对长时间运行的事务进行优化的一致性行为。Aurora 副本上的 READ COMMITTED 隔离级别没有 Aurora 主实例上的隔离那么严格。因此，仅在您知道查询可接受某些类型不一致结果的可能性的 Aurora 副本上启用此设置。

当 aurora_read_replica_read_committed 设置打开时，您的查询可能会遇到某些类型的读取异常。理解并处理应用程序代码中的两种异常特别重要。在查询运行期间提交另一个事务时，将发生不可重复的读取。长时间运行的查询在查询开始时看到的数据可能与在结束时看到的数据不同。当其他事务导致在查询运行期间将对现有行进行重组，并且查询将两次读取一行或多行时，将发生幻读。

您的查询可能会因幻读而导致行数不一致；也可能由于不可重复的读取而返回不完整或不一致的结果。例如，假设联接操作引用由 SQL 语句并发修改的表，如 INSERT 或 DELETE。在这种情况下，联接查询可能从一个表读取一行，但不从另一个表读取对应的行。

ANSI SQL 标准允许 READ COMMITTED 隔离级别存在这两种行为。但是，这些行为与 READ COMMITTED 的典型 MySQL 实现不同。因此，启用 aurora_read_replica_read_committed 设置之前，请先检查任何现有的 SQL 代码，以验证其在更宽松的一致性模型下是否按预期运行。

启用此设置时，READ COMMITTED 隔离级别下的行数和其他结果可能不具有强一致性。因此，通常只在运行聚合大量数据且无需绝对精度的分析查询时才启用该设置。如果没有这些类型的长时间运行的查询以及写入密集型工作负载，则可能不需要 aurora_read_replica_read_committed 设置。如果没有长时间运行的查询和写入密集型工作负载的组合，就不太可能遇到历史记录列表长度的问题。

Example 显示 READ COMMITTED 针对 Aurora 副本的隔离行为的查询

以下示例展示了如果事务同时修改关联表，针对 Aurora 副本的 READ COMMITTED 查询如何返回不可重复的结果。表 BIG_TABLE 在任何查询开始之前包含 100 万行。其他数据操作语言（DML）语句在运行时会添加、删除或更改行。

READ COMMITTED 隔离级别下针对 Aurora 主实例的查询生成可预测的结果。但是，在每个长时间运行的查询的生命周期内保持一致的读取视图，这样所产生的开销可能会导致以后的垃圾回收成本高昂。

我们优化了 READ COMMITTED 隔离级别下对 Aurora 副本的查询，以最大程度减少这种垃圾回收开销。权衡之下，结果可能有所不同，具体取决于查询是否检索在查询运行期间提交的事务所添加、删除或重组的行。允许查询考虑这些行，但不要求这样做。出于演示目的，查询仅使用 COUNT(*) 函数检查表中的行数。

Time	Aurora 主实例上的 DML 语句	针对 Aurora 主实例（具有 READ COMMITTED）的查询	针对 Aurora 副本（具有 READ COMMITTED）的查询
T1	`INSERT INTO big_table SELECT * FROM other_table LIMIT 1000000; COMMIT;`		
T2		Q1：`SELECT COUNT(*) FROM big_table;`	Q2：`SELECT COUNT(*) FROM big_table;`

Time	Aurora 主实例上的 DML 语句	针对 Aurora 主实例（具有 READ COMMITTED）的查询	针对 Aurora 副本（具有 READ COMMITTED）的查询
T3	`INSERT INTO big_table (c1, c2) VALUES (1, 'one more row'); COMMIT;`		
T4		如果 Q1 现在完成，则结果为 1,000,000。	如果 Q2 现在完成，则结果为 1,000,000 或 1,000,001。
T5	`DELETE FROM big_table LIMIT 2; COMMIT;`		
T6		如果 Q1 现在完成，则结果为 1,000,000。	如果 Q2 现在完成，则结果为 1,000,000 或 1,000,001 或 999,999 或 999,998。
T7	`UPDATE big_table SET c2 = CONCAT(c2 ,c2,c2); COMMIT;`		
T8		如果 Q1 现在完成，则结果为 1,000,000。	如果 Q2 现在完成，则结果为 1,000,000 或 1,000,001 或 999,999 或可能某个更大的数字。

Time	Aurora 主实例上的 DML 语句	针对 Aurora 主实例（具有 READ COMMITTED）的查询	针对 Aurora 副本（具有 READ COMMITTED）的查询
T9		Q3：`SELECT COUNT(*) FROM big_table;`	Q4：`SELECT COUNT(*) FROM big_table;`
T10		如果 Q3 现在完成，则结果为 999,999。	如果 Q4 现在完成，则结果为 999,999。
T11		Q5：`SELECT COUNT(*) FROM parent_table p JOIN child_table c ON (p.id = c.id) WHERE p.id = 1000;`	Q6：`SELECT COUNT(*) FROM parent_table p JOIN child_table c ON (p.id = c.id) WHERE p.id = 1000;`
T12	`INSERT INTO parent_table (id, s) VALUES (1000, 'hello'); INSERT INTO child_table (id, s) VALUES (1000, 'world'); COMMIT;`		
T13		如果 Q5 现在完成，则结果为 0。	如果 Q6 现在完成，则结果为 0 或 1。

如果查询快速完成，则在任何其他事务执行 DML 语句并提交之前，结果是可预测的，并且主实例与 Aurora 副本之间也是如此。让我们从第一个查询开始详细研究行为的差异。

Q1 的结果高度可预测，因为主实例上的 READ COMMITTED 使用类似于 REPEATABLE READ 隔离级别的强一致性模型。

Q2 的结果可能会因在该查询运行期间提交的事务而异。例如，假设其他事务执行 DML 语句并在查询运行期间提交。在这种情况下，针对具有 READ COMMITTED 隔离级别的 Aurora 副本的查询可能考虑更改，也可能不考虑更改。不能像在 REPEATABLE READ 隔离级别下那样预测行计数。它们也不像针对 READ COMMITTED 隔离级别下的主实例或针对 RDS for MySQL 实例运行的查询那样可预测。

T7 处的 UPDATE 语句实际上并未更改表中的行数。但是，通过更改可变长度列的长度，该语句可能导致在内部对行进行重组。长时间运行的 READ COMMITTED 事务可能会看到某行的旧版本，随后又在同一查询中看到该行的新版本。查询还可以跳过该行的旧版本和新版本，因此行计数可能与预期的不同。

Q5 和 Q6 的结果可能相同，也可能略有不同。READ COMMITTED 下针对 Aurora 副本的查询 Q6 可以查看（但不是必须查看）查询运行期间提交的新行。它也可能从一个表中看到该行，但从另一表中看不到该行。如果联接查询在两个表中均未找到匹配的行，则返回的计数为零。如果查询的确在 PARENT_TABLE 和 CHILD_TABLE 中找到了新行，则该查询返回的计数为一。在长时间运行的查询中，从联接的表进行查找的时间可能相隔很远。

> **Note**
>
> 这些行为上的差异取决于事务何时提交以及查询何时处理底层表行。因此，在耗时数分钟或数小时的报告查询以及同时在处理 OLTP 事务的 Aurora 集群上运行的报表查询中，您最有可能看到这样的差异。这些类型的混合工作负载从 Aurora 副本上的 READ COMMITTED 隔离级别获益最大。

Aurora MySQL 提示

您可以将 SQL 提示与 Aurora MySQL 查询结合使用来微调性能。您还可以使用提示来防止重要查询的执行计划由于不可预知的条件而发生变化。

> **Tip**
>
> 要验证提示对查询的影响，请查看 EXPLAIN 语句生成的查询计划。比较包含和不包含提示的查询计划。

在 Aurora MySQL 版本 3 中，您可以使用 MySQL 社群版 8.0 中提供的所有提示。有关这些提示的更多信息，请参阅《MySQL 参考手册》中的优化程序提示。

以下提示在 Aurora MySQL 版本 2 中可用。这些提示适用于使用 Aurora MySQL 版本 2 中的哈希联接功能的查询，尤其是使用并行查询优化的查询。

PQ、NO_PQ

指定是否强制优化程序在每个表或每个查询的基础上使用并行查询。

PQ 强制优化程序对指定的表或整个查询（块）使用并行查询。NO_PQ 防止优化程序对指定表或整个查询（块）使用并行查询。

此提示在 Aurora MySQL 版本 2.11 及更高版本中可用。以下示例向您显示如何使用此提示。

> **Note**
>
> 指定表名称会强制优化程序仅对那些选择的表应用 PQ/NO_PQ 提示。不指定表名称会强制对受查询块影响的所有表应用 PQ/NO_PQ 提示。

```
EXPLAIN SELECT /*+ PQ() */ f1, f2
    FROM num1 t1 WHERE f1 > 10 and f2 < 100;

EXPLAIN SELECT /*+ PQ(t1) */ f1, f2
    FROM num1 t1 WHERE f1 > 10 and f2 < 100;

EXPLAIN SELECT /*+ PQ(t1,t2) */ f1, f2
    FROM num1 t1, num1 t2 WHERE t1.f1 = t2.f21;

EXPLAIN SELECT /*+ NO_PQ() */ f1, f2
    FROM num1 t1 WHERE f1 > 10 and f2 < 100;

EXPLAIN SELECT /*+ NO_PQ(t1) */ f1, f2
    FROM num1 t1 WHERE f1 > 10 and f2 < 100;

EXPLAIN SELECT /*+ NO_PQ(t1,t2) */ f1, f2
    FROM num1 t1, num1 t2 WHERE t1.f1 = t2.f21;
```

HASH_JOIN、NO_HASH_JOIN

开启或关闭并行查询优化程序的功能来选择是否对查询使用哈希联接优化方法。HASH_JOIN 可让优化程序使用哈希联接（如果该机制更高效）。NO_HASH_JOIN 阻止优化程序对查询使用哈希联接。此提示在 Aurora MySQL 版本 2.08 及更高版本中可用。它在 Aurora MySQL 版本 3 中没有效果。

以下示例向您显示如何使用此提示。

```
EXPLAIN SELECT/*+ HASH_JOIN(t2) */ f1, f2
  FROM t1, t2 WHERE t1.f1 = t2.f1;

EXPLAIN SELECT /*+ NO_HASH_JOIN(t2) */ f1, f2
  FROM t1, t2 WHERE t1.f1 = t2.f1;
```

HASH_JOIN_PROBING、NO_HASH_JOIN_PROBING

在哈希联接查询中，指定是否将指定的表用于联接的探查端。查询测试构建表中的列值是否存在于探查表中，而不是读取探查表的全部内容。您可以使用 HASH_JOIN_PROBING 和 HASH_JOIN_BUILDING 指定如何处理哈希联接查询，而无需重新排序查询文本中的表。此提示在 Aurora MySQL 版本 2.08 及更高版本中可用。它在 Aurora MySQL 版本 3 中没有效果。

以下示例显示如何使用此提示。为表 HASH_JOIN_PROBING 指定 T2 提示与为表 NO_HASH_JOIN_PROBING 指定 T1 具有相同的效果。

```
EXPLAIN SELECT /*+ HASH_JOIN(t2) HASH_JOIN_PROBING(t2) */ f1, f2
  FROM t1, t2 WHERE t1.f1 = t2.f1;

EXPLAIN SELECT /*+ HASH_JOIN(t2) NO_HASH_JOIN_PROBING(t1) */ f1, f2
  FROM t1, t2 WHERE t1.f1 = t2.f1;
```

HASH_JOIN_BUILDING、NO_HASH_JOIN_BUILDING

在哈希联接查询中，指定是否将指定的表用于联接的构建端。查询处理此表中的所有行来构建列值列表，以便与其他表进行交叉引用。您可以使用 HASH_JOIN_PROBING 和 HASH_JOIN_BUILDING 指定如何处理哈希联接查询，而无需重新排序查询文本中的表。此提示在 Aurora MySQL 版本 2.08 及更高版本中可用。它在 Aurora MySQL 版本 3 中没有效果。

以下示例向您显示如何使用此提示。为表 HASH_JOIN_BUILDING 指定 T2 提示与为表 NO_HASH_JOIN_BUILDING 指定 T1 具有相同的效果。

```
EXPLAIN SELECT /*+ HASH_JOIN(t2) HASH_JOIN_BUILDING(t2) */ f1, f2
  FROM t1, t2 WHERE t1.f1 = t2.f1;

EXPLAIN SELECT /*+ HASH_JOIN(t2) NO_HASH_JOIN_BUILDING(t1) */ f1, f2
  FROM t1, t2 WHERE t1.f1 = t2.f1;
```

JOIN_FIXED_ORDER

指定查询中的表按它们在查询中的列出顺序进行联接。它对于涉及三个或更多表的查询很有用。它旨在替代 MySQL STRAIGHT_JOIN 提示，作用等同于 MySQL JOIN_FIXED_ORDER 提示。此提示在 Aurora MySQL 版本 2.08 及更高版本中可用。

以下示例向您显示如何使用此提示。

```
EXPLAIN SELECT /*+ JOIN_FIXED_ORDER() */ f1, f2
  FROM t1 JOIN t2 USING (id) JOIN t3 USING (id) JOIN t4 USING (id);
```

JOIN_ORDER

指定查询中表的联接顺序。它对于涉及三个或更多表的查询很有用。它等效于 MySQL JOIN_ORDER 提示。此提示在 Aurora MySQL 版本 2.08 及更高版本中可用。

以下示例向您显示如何使用此提示。

```
EXPLAIN SELECT /*+ JOIN_ORDER (t4, t2, t1, t3) */ f1, f2
  FROM t1 JOIN t2 USING (id) JOIN t3 USING (id) JOIN t4 USING (id);
```

JOIN_PREFIX

指定联接顺序中首先放置的表。它对于涉及三个或更多表的查询很有用。它等效于 MySQL JOIN_PREFIX 提示。此提示在 Aurora MySQL 版本 2.08 及更高版本中可用。

以下示例向您显示如何使用此提示。

```
EXPLAIN SELECT /*+ JOIN_PREFIX (t4, t2) */ f1, f2
  FROM t1 JOIN t2 USING (id) JOIN t3 USING (id) JOIN t4 USING (id);
```

JOIN_SUFFIX

指定联接顺序中最后放置的表。它对于涉及三个或更多表的查询很有用。它等效于 MySQL JOIN_SUFFIX 提示。此提示在 Aurora MySQL 版本 2.08 及更高版本中可用。

以下示例向您显示如何使用此提示。

```
EXPLAIN SELECT /*+ JOIN_SUFFIX (t1) */ f1, f2
  FROM t1 JOIN t2 USING (id) JOIN t3 USING (id) JOIN t4 USING (id);
```

有关使用哈希联接查询的信息，请参阅使用哈希联接优化大型 Aurora MySQL 联接查询。

Aurora MySQL 存储过程参考

您可以通过调用内置的存储过程管理 Aurora MySQL 数据库集群。

主题

- 收集和维护全局状态历史记录
- 配置、启动和停止二进制日志（binlog）复制
- 结束会话或查询
- 使用 GTID 复制事务
- 轮换查询日志
- 设置和显示二进制日志配置

收集和维护全局状态历史记录

Amazon RDS 提供了一组过程，这些过程可以随时间推移对状态变量的值创建快照，并将它们及上次创建快照后所做的任何更改写入一个表中。该基础设施称为全局状态历史记录。有关更多信息，请参阅管理全局状态历史记录。

以下存储过程管理全局状态历史记录的收集和维护方式。

主题

- mysql.rds_collect_global_status_history
- mysql.rds_disable_gsh_collector
- mysql.rds_disable_gsh_rotation
- mysql.rds_enable_gsh_collector
- mysql.rds_enable_gsh_rotation
- mysql.rds_rotate_global_status_history
- mysql.rds_set_gsh_collector
- mysql.rds_set_gsh_rotation

mysql.rds_collect_global_status_history

按需为全局状态历史记录创建快照。

语法

```
CALL mysql.rds_collect_global_status_history;
```

mysql.rds_disable_gsh_collector

关闭全局状态历史记录创建的快照。

语法

```
CALL mysql.rds_disable_gsh_collector;
```

mysql.rds_disable_gsh_rotation

关闭 `mysql.global_status_history` 表的交替。

语法

```
CALL mysql.rds_disable_gsh_rotation;
```

mysql.rds_enable_gsh_collector

开启全局状态历史记录，以 `rds_set_gsh_collector` 指定的间隔创建原定设置快照。

语法

```
CALL mysql.rds_enable_gsh_collector;
```

mysql.rds_enable_gsh_rotation

按照 `rds_set_gsh_rotation` 指定的间隔，开启将 `mysql.global_status_history` 表的内容交替到 `mysql.global_status_history_old`

语法

```
CALL mysql.rds_enable_gsh_rotation;
```

mysql.rds_rotate_global_status_history

根据需求将 `mysql.global_status_history` 表的内容交替到 `mysql.global_status_history_old`。

语法

```
CALL mysql.rds_rotate_global_status_history;
```

mysql.rds_set_gsh_collector

指定全局状态历史记录创建的快照之间的间隔，以分钟为单位。

语法

```
CALL mysql.rds_set_gsh_collector(intervalPeriod);
```

参数

intervalPeriod

快照之间的间隔，以分钟为单位。默认值为 5。

mysql.rds_set_gsh_rotation

指定 mysql.global_status_history 表轮换之间的间隔，以天为单位。

语法

```
CALL mysql.rds_set_gsh_rotation(intervalPeriod);
```

参数

intervalPeriod

表轮换之间的间隔，以天为单位。默认值为 7。

配置、启动和停止二进制日志（binlog）复制

您可以在连接到 Aurora MySQL 集群中的主实例时，调用以下存储过程。这些过程控制事务如何从外部数据库复制到 Aurora MySQL，或从 Aurora MySQL 复制到外部数据库。

主题

- mysql.rds_disable_session_binlog（Aurora MySQL 版本 2）
- mysql.rds_enable_session_binlog（Aurora MySQL 版本 2）
- mysql.rds_import_binlog_ssl_material
- mysql.rds_next_master_log（Aurora MySQL 版本 2）
- mysql.rds_next_source_log（Aurora MySQL 版本 3）
- mysql.rds_remove_binlog_ssl_material
- mysql.rds_reset_external_master（Aurora MySQL 版本 2）
- mysql.rds_reset_external_source（Aurora MySQL 版本 3）
- mysql.rds_set_binlog_source_ssl（Aurora MySQL 版本 3）
- mysql.rds_set_external_master（Aurora MySQL 版本 2）
- mysql.rds_set_external_source（Aurora MySQL 版本 3）
- mysql.rds_set_external_master_with_auto_position（Aurora MySQL 版本 2）
- mysql.rds_set_external_source_with_auto_position（Aurora MySQL 版本 3）
- mysql.rds_set_master_auto_position（Aurora MySQL 版本 2）
- mysql.rds_set_read_only（Aurora MySQL 版本 3）
- mysql.rds_set_session_binlog_format（Aurora MySQL 版本 2）
- mysql.rds_set_source_auto_position（Aurora MySQL 版本 3）
- mysql.rds_skip_repl_error
- mysql.rds_start_replication
- mysql.rds_start_replication_until（Aurora MySQL 版本 3）
- mysql.rds_stop_replication

mysql.rds_disable_session_binlog（Aurora MySQL 版本 2）

通过将 `sql_log_bin` 变量设置为 `OFF` 来关闭当前会话的二进制日志记录。

语法

```
CALL mysql.rds_disable_session_binlog;
```

参数

无

使用说明

对于 Aurora MySQL 数据库集群，您将在连接到主实例时调用此存储过程。

对于 Aurora，Aurora MySQL 版本 2.12 及更高的与 MySQL 5.7 兼容的版本支持此过程。

> **Note**
>
> 在 Aurora MySQL 版本 3 中，如果您具有 SESSION_VARIABLES_ADMIN 权限，则可以使用以下命令禁用当前会话的二进制日志记录：
>
> ```
> SET SESSION sql_log_bin = OFF;
> ```

mysql.rds_enable_session_binlog（Aurora MySQL 版本 2）

通过将 sql_log_bin 变量设置为 ON 来开启当前会话的二进制日志记录。

语法

```
CALL mysql.rds_enable_session_binlog;
```

参数

无

使用说明

对于 Aurora MySQL 数据库集群，您将在连接到主实例时调用此存储过程。

对于 Aurora，Aurora MySQL 版本 2.12 及更高的与 MySQL 5.7 兼容的版本支持此过程。

> **Note**
>
> 在 Aurora MySQL 版本 3 中，如果您具有 SESSION_VARIABLES_ADMIN 权限，则可以使用以下命令启用当前会话的二进制日志记录：
>
> ```
> SET SESSION sql_log_bin = ON;
> ```

mysql.rds_import_binlog_ssl_material

将证书颁发机构证书、客户端证书和客户端密钥导入到 Aurora MySQL 数据库集群中。SSL 通信和加密复制需要此类信息。

> **Note**
>
> 目前，Aurora MySQL 版本 2（2.09.2、2.10.0、2.10.1 和 2.11.0）以及版本 3（3.01.1 及更高版本）支持此过程。

语法

```
CALL mysql.rds_import_binlog_ssl_material (
  ssl_material
);
```

参数

ssl_material

包含 MySQL 客户端的以下 .pem 格式文件内容的 JSON 负载：

- "ssl_ca":"*########*"
- "ssl_cert":"*#####*"
- "ssl_key":"*#####*"

使用说明

在运行该过程之前，为加密复制做好以下准备：

- 如果您没有在外部 MySQL 源数据库实例上启用 SSL，并且没有准备客户端密钥和客户端证书，请在 MySQL 数据库服务器上启用 SSL 并生成所需的客户端密钥和客户端证书。
- 如果在外部源数据库实例上启用了 SSL，请提供 Aurora MySQL 数据库集群的客户端密钥和证书。如果未提供，请为 Aurora MySQL 数据库集群生成新密钥和证书。要对客户端证书进行签名，您必须拥有您用来在外部 MySQL 源数据库实例上配置 SSL 的证书颁发机构密钥。

有关更多信息，请参阅 MySQL 文档中的使用 openssl 创建 SSL 证书和密钥。

> **Important**
>
> 为加密复制做好准备后，使用 SSL 连接来运行该过程。不得通过不安全的连接传输客户端密钥。

该过程会将 SSL 信息从外部 MySQL 数据库导入到 Aurora MySQL 数据库集群中。SSL 信息是 .pem 格式文件，其中包含 Aurora MySQL 数据库集群的 SSL 信息。在加密复制期间，Aurora MySQL 数据库集群充当 MySQL 数据库服务器的客户端。Aurora MySQL 客户端的证书和密钥必须是 .pem 格式的文件。

您可以将这些文件中的信息复制到正确 JSON 负载的 `ssl_material` 参数中。要支持加密复制，请将此类 SSL 信息导入到 Aurora MySQL 数据库集群中。

JSON 负载必须为以下格式。

```
'{"ssl_ca":"-----BEGIN CERTIFICATE-----
ssl_ca_pem_body_code
-----END CERTIFICATE-----\n","ssl_cert":"-----BEGIN CERTIFICATE-----
ssl_cert_pem_body_code
-----END CERTIFICATE-----\n","ssl_key":"-----BEGIN RSA PRIVATE KEY-----
ssl_key_pem_body_code
-----END RSA PRIVATE KEY-----\n"}'
```

示例

以下示例将 SSL 信息导入到 Aurora MySQL。在 .pem 格式文件中，主体代码通常比示例中显示的主体代码长。

```
call mysql.rds_import_binlog_ssl_material(
```

```
'{"ssl_ca":"-----BEGIN CERTIFICATE-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS7O6V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSFJuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz221CBt5IMucxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END CERTIFICATE-----\n","ssl_cert":"-----BEGIN CERTIFICATE-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS7O6V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSFJuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz221CBt5IMucxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END CERTIFICATE-----\n","ssl_key":"-----BEGIN RSA PRIVATE KEY-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS7O6V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSFJuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz221CBt5IMucxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END RSA PRIVATE KEY-----\n"}');
```

mysql.rds_next_master_log（Aurora MySQL 版本 2）

将源数据库实例日志位置更改为源数据库实例上的下一个二进制日志的开始位置。只有在只读副本上收到复制 I/O 错误 1236 时，才能使用该过程。

语法

```
CALL mysql.rds_next_master_log(
curr_master_log
);
```

参数

curr_master_log

当前主日志文件的索引。例如，如果当前文件名为 mysql-bin-changelog.012345，则索引为 12345。若要确定当前主日志文件名，请运行 SHOW REPLICA STATUS 命令并查看 Master_Log_File 字段。

使用说明

主用户必须运行 mysql.rds_next_master_log 过程。

> **Warning**
>
> 仅在对作为复制源的多可用区数据库实例进行故障转移后复制失败，并且 `mysql.rds_next_master_log` 的 `Last_IO_Errno` 字段报告 I/O 错误 1236 时调用 `SHOW REPLICA STATUS`。
> 如果在发生失效转移事件之前，源实例中的事务未写入到磁盘上的二进制日志，调用 `mysql.rds_next_master_log` 可能会导致只读副本丢失数据。

示例

假设复制在 Aurora MySQL 只读副本上失败。对只读副本运行 `SHOW REPLICA STATUS\G` 会返回以下结果：

```
*************************** 1. row ***************************
             Replica_IO_State:
                  Source_Host: myhost.XXXXXXXXXXXXXX.rr-rrrr-1.rds.amazonaws.com
                  Source_User: MasterUser
                  Source_Port: 3306
                Connect_Retry: 10
              Source_Log_File: mysql-bin-changelog.012345
          Read_Source_Log_Pos: 1219393
               Relay_Log_File: relaylog.012340
                Relay_Log_Pos: 30223388
        Relay_Source_Log_File: mysql-bin-changelog.012345
           Replica_IO_Running: No
          Replica_SQL_Running: Yes
              Replicate_Do_DB:
          Replicate_Ignore_DB:
           Replicate_Do_Table:
       Replicate_Ignore_Table:
      Replicate_Wild_Do_Table:
  Replicate_Wild_Ignore_Table:
                   Last_Errno: 0
                   Last_Error:
                 Skip_Counter: 0
          Exec_Source_Log_Pos: 30223232
              Relay_Log_Space: 5248928866
              Until_Condition: None
               Until_Log_File:
                Until_Log_Pos: 0
           Source_SSL_Allowed: No
```

```
           Source_SSL_CA_File:
           Source_SSL_CA_Path:
              Source_SSL_Cert:
            Source_SSL_Cipher:
               Source_SSL_Key:
        Seconds_Behind_Master: NULL
Source_SSL_Verify_Server_Cert: No
                Last_IO_Errno: 1236
                Last_IO_Error: Got fatal error 1236 from master when reading data from
 binary log: 'Client requested master to start replication from impossible position;
 the first event 'mysql-bin-changelog.013406' at 1219393, the last event read from
 '/rdsdbdata/log/binlog/mysql-bin-changelog.012345' at 4, the last byte read from '/
rdsdbdata/log/binlog/mysql-bin-changelog.012345' at 4.'
               Last_SQL_Errno: 0
               Last_SQL_Error:
  Replicate_Ignore_Server_Ids:
             Source_Server_Id: 67285976
```

Last_IO_Errno 字段显示该实例收到 I/O 错误 1236。Master_Log_File 字段显示文件名为 mysql-bin-changelog.012345，这意味着日志文件索引为 12345。要纠正该错误，您可以使用以下参数调用 mysql.rds_next_master_log：

```
CALL mysql.rds_next_master_log(12345);
```

mysql.rds_next_source_log（Aurora MySQL 版本 3）

将源数据库实例日志位置更改为源数据库实例上的下一个二进制日志的开始位置。只有在只读副本上收到复制 I/O 错误 1236 时，才能使用该过程。

语法

```
CALL mysql.rds_next_source_log(
curr_source_log
);
```

参数

curr_source_log

当前源日志文件的索引。例如，如果当前文件名为 mysql-bin-changelog.012345，则索引为 12345。要确定当前源日志文件名，请运行 SHOW REPLICA STATUS 命令并查看 Source_Log_File 字段。

使用说明

管理用户必须运行 mysql.rds_next_source_log 过程。

> **Warning**
>
> 仅在对作为复制源的多可用区数据库实例进行故障转移后复制失败，并且 mysql.rds_next_source_log 的 Last_IO_Errno 字段报告 I/O 错误 1236 时调用 SHOW REPLICA STATUS。
> 如果在发生失效转移事件之前，源实例中的事务未写入到磁盘上的二进制日志，调用 mysql.rds_next_source_log 可能会导致只读副本丢失数据。可以通过将源实例参数 sync_binlog 和 innodb_support_xa 设置为 1 来减少发生这种情况的几率，尽管这可能会降低性能。

示例

假设复制在 Aurora MySQL 只读副本上失败。对只读副本运行 SHOW REPLICA STATUS\G 会返回以下结果：

```
*************************** 1. row ***************************
             Replica_IO_State:
                  Source_Host: myhost.XXXXXXXXXXXXXX.rr-rrrr-1.rds.amazonaws.com
                  Source_User: MasterUser
                  Source_Port: 3306
                Connect_Retry: 10
              Source_Log_File: mysql-bin-changelog.012345
          Read_Source_Log_Pos: 1219393
               Relay_Log_File: relaylog.012340
                Relay_Log_Pos: 30223388
        Relay_Source_Log_File: mysql-bin-changelog.012345
           Replica_IO_Running: No
          Replica_SQL_Running: Yes
```

```
               Replicate_Do_DB:
           Replicate_Ignore_DB:
            Replicate_Do_Table:
        Replicate_Ignore_Table:
       Replicate_Wild_Do_Table:
  Replicate_Wild_Ignore_Table:
                   Last_Errno: 0
                   Last_Error:
                 Skip_Counter: 0
          Exec_Source_Log_Pos: 30223232
              Relay_Log_Space: 5248928866
              Until_Condition: None
               Until_Log_File:
                Until_Log_Pos: 0
           Source_SSL_Allowed: No
           Source_SSL_CA_File:
           Source_SSL_CA_Path:
              Source_SSL_Cert:
            Source_SSL_Cipher:
               Source_SSL_Key:
        Seconds_Behind_Source: NULL
Source_SSL_Verify_Server_Cert: No
                Last_IO_Errno: 1236
                Last_IO_Error: Got fatal error 1236 from source when reading data from
 binary log: 'Client requested source to start replication from impossible position;
 the first event 'mysql-bin-changelog.013406' at 1219393, the last event read from
 '/rdsdbdata/log/binlog/mysql-bin-changelog.012345' at 4, the last byte read from '/
rdsdbdata/log/binlog/mysql-bin-changelog.012345' at 4.'
               Last_SQL_Errno: 0
               Last_SQL_Error:
  Replicate_Ignore_Server_Ids:
             Source_Server_Id: 67285976
```

Last_IO_Errno 字段显示该实例收到 I/O 错误 1236。Source_Log_File 字段显示文件名为 mysql-bin-changelog.012345，这意味着日志文件索引为 12345。要纠正该错误，您可以使用以下参数调用 mysql.rds_next_source_log：

```
CALL mysql.rds_next_source_log(12345);
```

mysql.rds_remove_binlog_ssl_material

删除用于 SSL 通信和加密复制的证书颁发机构证书、客户端证书与客户端密钥。此类信息可通过使用 mysql.rds_import_binlog_ssl_material 来导入。

语法

```
CALL mysql.rds_remove_binlog_ssl_material;
```

mysql.rds_reset_external_master（Aurora MySQL 版本 2）

重新配置 Aurora MySQL 数据库实例，使其不再是在 Amazon RDS 之外运行的某个 MySQL 实例的只读副本。

> **Important**
>
> 要运行此过程，必须启用 `autocommit`。要启用它，请将 `autocommit` 参数设置为 1。有关修改参数的信息，请参阅 在 Amazon Aurora 中修改数据库参数组中的参数。

语法

```
CALL mysql.rds_reset_external_master;
```

使用说明

主用户必须运行 `mysql.rds_reset_external_master` 过程。此过程必须运行于一个 MySQL 数据库实例上，后者要作为在 Amazon RDS 之外运行的 MySQL 实例的只读副本而被删除。

> **Note**
>
> 我们提供这些存储过程主要是为了与在 Amazon RDS 外部运行的 MySQL 实例之间启用复制。建议您尽可能使用 Aurora 副本来管理 Aurora MySQL 数据库集群中的复制。有关在 Aurora MySQL 数据库集群中管理复制的信息，请参阅 使用 Aurora 副本。

想要了解更多有关使用复制从在 Aurora MySQL 之外运行的 MySQL 实例导入数据的信息，请参阅Aurora 与 MySQL 之间或 Aurora 与其他 Aurora 数据库集群之间的复制（二进制日志复制）。

mysql.rds_reset_external_source（Aurora MySQL 版本 3）

重新配置 Aurora MySQL 数据库实例，使其不再是在 Amazon RDS 之外运行的某个 MySQL 实例的只读副本。

> **Important**
>
> 要运行此过程，必须启用 autocommit。要启用它，请将 autocommit 参数设置为 1。有关修改参数的信息，请参阅 在 Amazon Aurora 中修改数据库参数组中的参数。

语法

```
CALL mysql.rds_reset_external_source;
```

使用说明

管理用户必须运行 mysql.rds_reset_external_source 过程。此过程必须运行于一个 MySQL 数据库实例上，后者要作为在 Amazon RDS 之外运行的 MySQL 实例的只读副本而被删除。

> **Note**
>
> 我们提供这些存储过程主要是为了与在 Amazon RDS 外部运行的 MySQL 实例之间启用复制。建议您尽可能使用 Aurora 副本来管理 Aurora MySQL 数据库集群中的复制。有关在 Aurora MySQL 数据库集群中管理复制的信息，请参阅 使用 Aurora 副本。

mysql.rds_set_binlog_source_ssl（Aurora MySQL 版本 3）

为二进制日志复制启用 SOURCE_SSL 加密。有关更多信息，请参阅 MySQL 文档中的 CHANGE REPLICATION SOURCE TO statement。

语法

```
CALL mysql.rds_set_binlog_source_ssl(mode);
```

参数

mode

指示是否启用 SOURCE_SSL 加密的值：

- 0 – SOURCE_SSL 加密已禁用。默认为 0。

- 1 – SOURCE_SSL 加密已启用。您可以使用 SSL 或 TLS 配置加密。

使用说明

Aurora MySQL 版本 3.06 及更高版本支持该过程。

mysql.rds_set_external_master（Aurora MySQL 版本 2）

将 Aurora MySQL 数据库实例配置为在 Amazon RDS 之外运行的 MySQL 实例的只读副本。

mysql.rds_set_external_master 过程已弃用，并会在将来的版本中删除。请改用 mysql.rds_set_external_source。

> **Important**
>
> 要运行此过程，必须启用 autocommit。要启用它，请将 autocommit 参数设置为 1。有关修改参数的信息，请参阅 在 Amazon Aurora 中修改数据库参数组中的参数。

语法

```
CALL mysql.rds_set_external_master (
  host_name
  , host_port
  , replication_user_name
  , replication_user_password
  , mysql_binary_log_file_name
  , mysql_binary_log_file_location
  , ssl_encryption
);
```

参数

host_name

在 Amazon RDS 之外运行以变为源数据库实例的 MySQL 实例的主机名或 IP 地址。

host_port

在 Amazon RDS 之外运行的要配置为源数据库实例的 MySQL 实例使用的端口。如果网络配置包括转换端口号的安全 Shell (SSH) 端口复制，请指定由 SSH 公开的端口号。

replication_user_name

对在 Amazon RDS 外部运行的 MySQL 实例具有 REPLICATION CLIENT 和 REPLICATION SLAVE 权限的用户的 ID。建议您向专用于复制的账户提供外部实例。

replication_user_password

在 replication_user_name 中指定的用户 ID 的密码。

mysql_binary_log_file_name

源数据库实例上包含复制信息的二进制日志的名称。

mysql_binary_log_file_location

mysql_binary_log_file_name 二进制日志中复制将开始读取复制信息的位置。

您可以通过在源数据库实例上运行 SHOW MASTER STATUS 来确定二进制日志文件名和位置。

ssl_encryption

指定是否在复制连接中使用安全套接字层（SSL）加密的值。1 表示使用 SSL 加密，0 表示不使用加密。默认值为 0。

> **Note**
>
> 不支持 MASTER_SSL_VERIFY_SERVER_CERT 选项。此选项设置为 0，这意味着连接已加密，但未验证证书。

使用说明

主用户必须运行 mysql.rds_set_external_master 过程。必须在要配置为在 Amazon RDS 外部运行的 MySQL 实例的只读副本的 MySQL 数据库实例上运行该过程。

运行 mysql.rds_set_external_master 之前，您必须先将在 Amazon RDS 之外运行的 MySQL 实例配置为源数据库实例。要连接到在 Amazon RDS 之外运行的 MySQL 实例，您必须指定 replication_user_name 和 replication_user_password 值，这些值指示对 MySQL 的外部实例具有 REPLICATION CLIENT 和 REPLICATION SLAVE 权限的复制用户。

将 MySQL 的外部实例配置为源数据库实例

1. 通过使用所选的 MySQL 客户端，连接到 MySQL 的外部实例并创建要用于复制的用户账户。示例如下：

MySQL 5.7

```
CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED BY 'password';
```

MySQL 8.0

```
CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED WITH mysql_native_password BY
 'password';
```

> **Note**
>
> 作为安全最佳实践，请指定除此处所示提示以外的密码。

2. 对于 MySQL 的外部实例，向复制用户授予 REPLICATION CLIENT 和 REPLICATION SLAVE 权限。以下示例向您所在域的“repl_user”用户授予对所有数据库的 REPLICATION CLIENT 和 REPLICATION SLAVE 权限。

   MySQL 5.7

   ```
   GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com'
    IDENTIFIED BY 'password';
   ```

   MySQL 8.0

   ```
   GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com';
   ```

要使用加密复制，请将源数据库实例配置为使用 SSL 连接。此外，使用 mysql.rds_import_binlog_ssl_material 过程，将证书颁发机构证书、客户端证书和客户端密钥导入到数据库实例或数据库集群中。

> **Note**
>
> 我们提供这些存储过程主要是为了与在 Amazon RDS 外部运行的 MySQL 实例之间启用复制。建议您尽可能使用 Aurora 副本来管理 Aurora MySQL 数据库集群中的复制。有关在 Aurora MySQL 数据库集群中管理复制的信息，请参阅 使用 Aurora 副本。

在调用 mysql.rds_set_external_master 将 Amazon RDS 数据库实例配置为只读副本后，可对该只读副本调用 mysql.rds_start_replication 开始复制过程。您可以调用 mysql.rds_reset_external_master（Aurora MySQL 版本 2）删除只读副本配置。

调用 mysql.rds_set_external_master 时，Amazon RDS 将时间、用户和 set master 的操作记录在 mysql.rds_history 和 mysql.rds_replication_status 表中。

示例

在 MySQL 数据库实例上运行时，下例将该数据库实例配置为在 Amazon RDS 之外运行的某个 MySQL 实例的只读副本。

```
call mysql.rds_set_external_master(
  'Externaldb.some.com',
  3306,
  'repl_user',
  'password',
  'mysql-bin-changelog.0777',
  120,
  1);
```

mysql.rds_set_external_source（Aurora MySQL 版本 3）

将 Aurora MySQL 数据库实例配置为在 Amazon RDS 之外运行的 MySQL 实例的只读副本。

> **Important**
>
> 要运行此过程，必须启用 autocommit。要启用它，请将 autocommit 参数设置为 1。有关修改参数的信息，请参阅 在 Amazon Aurora 中修改数据库参数组中的参数。

语法

```
CALL mysql.rds_set_external_source (
  host_name
  , host_port
  , replication_user_name
  , replication_user_password
  , mysql_binary_log_file_name
  , mysql_binary_log_file_location
  , ssl_encryption
```

```
);
```

参数

host_name

在 Amazon RDS 之外运行以变为源数据库实例的 MySQL 实例的主机名或 IP 地址。

host_port

在 Amazon RDS 之外运行的要配置为源数据库实例的 MySQL 实例使用的端口。如果网络配置包括转换端口号的安全 Shell (SSH) 端口复制，请指定由 SSH 公开的端口号。

replication_user_name

对在 Amazon RDS 外部运行的 MySQL 实例具有 REPLICATION CLIENT 和 REPLICATION SLAVE 权限的用户的 ID。建议您向专用于复制的账户提供外部实例。

replication_user_password

在 replication_user_name 中指定的用户 ID 的密码。

mysql_binary_log_file_name

源数据库实例上包含复制信息的二进制日志的名称。

mysql_binary_log_file_location

mysql_binary_log_file_name 二进制日志中复制将开始读取复制信息的位置。

您可以通过在源数据库实例上运行 SHOW MASTER STATUS 来确定二进制日志文件名和位置。

ssl_encryption

指定是否在复制连接中使用安全套接字层（SSL）加密的值。1 表示使用 SSL 加密，0 表示不使用加密。默认值是 0。

> **Note**
>
> 要启用此选项，您必须已使用 mysql.rds_import_binlog_ssl_material 导入自定义 SSL 证书。如果您尚未导入自定义 SSL 证书，请将此参数设置为 0，然后使用 mysql.rds_set_binlog_source_ssl（Aurora MySQL 版本 3）启用 SSL 以进行二进制日志复制。
>
> 不支持 SOURCE_SSL_VERIFY_SERVER_CERT 选项。此选项设置为 0，这意味着连接已加密，但未验证证书。

使用说明

管理用户必须运行 mysql.rds_set_external_source 过程。该过程必须在以下 Aurora MySQL 数据库实例上运行：也即，该实例将配置为在 Amazon RDS 外部运行的 MySQL 实例的只读副本。

运行 mysql.rds_set_external_source 之前，必须先将在 Amazon RDS 之外运行的 MySQL 实例配置为源数据库实例。要连接到在 Amazon RDS 之外运行的 MySQL 实例，您必须指定 replication_user_name 和 replication_user_password 值，这些值指示对 MySQL 的外部实例具有 REPLICATION CLIENT 和 REPLICATION SLAVE 权限的复制用户。

将 MySQL 的外部实例配置为源数据库实例

1. 通过使用所选的 MySQL 客户端，连接到 MySQL 的外部实例并创建要用于复制的用户账户。示例如下：

```
CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED BY 'password';
```

> **Note**
>
> 作为安全最佳实践，请指定除此处所示提示以外的密码。

2. 对于 MySQL 的外部实例，向复制用户授予 REPLICATION CLIENT 和 REPLICATION SLAVE 权限。以下示例向您所在域的“repl_user”用户授予对所有数据库的 REPLICATION CLIENT 和 REPLICATION SLAVE 权限。

```
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com';
```

要使用加密复制，请将源数据库实例配置为使用 SSL 连接。此外，使用 mysql.rds_import_binlog_ssl_material 过程，将证书颁发机构证书、客户端证书和客户端密钥导入到数据库实例或数据库集群中。

> **Note**
>
> 我们提供这些存储过程主要是为了与在 Amazon RDS 外部运行的 MySQL 实例之间启用复制。建议您尽可能使用 Aurora 副本来管理 Aurora MySQL 数据库集群中的复制。有关在 Aurora MySQL 数据库集群中管理复制的信息，请参阅 使用 Aurora 副本。

在调用 mysql.rds_set_external_source 将 Aurora MySQL 数据库实例配置为只读副本后，可对该只读副本调用 mysql.rds_start_replication 以开始复制过程。您可以调用 mysql.rds_reset_external_source（Aurora MySQL 版本 3）来删除只读副本配置。

调用 mysql.rds_set_external_source 时，Amazon RDS 将时间、用户和 set master 的操作记录在 mysql.rds_history 和 mysql.rds_replication_status 表中。

示例

在 Aurora MySQL 数据库实例上运行时，以下示例将该数据库实例配置为在 Amazon RDS 外部运行的某个 MySQL 实例的只读副本。

```
call mysql.rds_set_external_source(
  'Externaldb.some.com',
  3306,
  'repl_user',
  'password',
  'mysql-bin-changelog.0777',
  120,
  1);
```

mysql.rds_set_external_master_with_auto_position（Aurora MySQL 版本 2）

将 Aurora MySQL 主实例配置为接受来自外部 MySQL 实例的传入复制。此过程还会根据全局事务标识符 (GTID) 配置复制。

此过程不会配置延迟复制，因为 Aurora MySQL 不支持延迟复制。

语法

```
CALL mysql.rds_set_external_master_with_auto_position (
  host_name
  , host_port
  , replication_user_name
  , replication_user_password
  , ssl_encryption
);
```

参数

host_name

在 Aurora 之外运行以变为复制源的 MySQL 实例的主机名或 IP 地址。

host_port

在 Aurora 之外运行的要配置为复制源的 MySQL 实例使用的端口。如果网络配置包括转换端口号的安全 Shell (SSH) 端口复制，请指定由 SSH 公开的端口号。

replication_user_name

在对 Aurora 外部运行的 MySQL 实例上具有 REPLICATION CLIENT 和 REPLICATION SLAVE 权限的用户的 ID。建议您向专用于复制的账户提供外部实例。

replication_user_password

在 replication_user_name 中指定的用户 ID 的密码。

ssl_encryption

目前未实施该选项。默认值是 0。

使用说明

对于 Aurora MySQL 数据库集群，您将在连接到主实例时调用此存储过程。

主用户必须运行 mysql.rds_set_external_master_with_auto_position 过程。主用户在充当复制目标的 Aurora MySQL 数据库集群的主实例上运行此过程。这可能是外部 MySQL 数据库实例或 Aurora MySQL 数据库集群的复制目标。

Aurora MySQL 版本 2 支持该过程。对于 Aurora MySQL 版本 3，请改为使用程序 mysql.rds_set_external_source_with_auto_position（Aurora MySQL 版本 3）。

在运行 mysql.rds_set_external_master_with_auto_position 前，请将外部 MySQL 数据库实例配置为复制源。要连接到外部 MySQL 实例，应指定 replication_user_name 和 replication_user_password 值。这些值必须指示具有外部 MySQL 实例上的 REPLICATION CLIENT 和 REPLICATION SLAVE 权限的复制用户。

要将外部 MySQL 实例配置为复制源

1. 通过使用所选的 MySQL 客户端，连接到外部 MySQL 实例并创建要用于复制的用户账户。以下是示例。

```
CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED BY 'SomePassW0rd'
```

2. 在外部 MySQL 实例上，向复制用户授予 REPLICATION CLIENT 和 REPLICATION SLAVE 权限。以下示例为您的域的 REPLICATION CLIENT 用户授予所有数据库的 REPLICATION SLAVE 和 'repl_user' 权限。

```
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com'
IDENTIFIED BY 'SomePassW0rd'
```

在调用 mysql.rds_set_external_master_with_auto_position 时，Amazon RDS 将记录某些信息。这些信息包括 "set master" 和 mysql.rds_history 表中的时间、用户和 mysql.rds_replication_status 操作。

要跳过已知会导致问题的基于 GTID 的特定事务，您可以使用 mysql.rds_skip_transaction_with_gtid（Aurora MySQL 版本 2 和 3）存储过程。有关使用基于 GTID 的复制的更多信息，请参阅使用基于 GTID 的复制。

示例

在 Aurora 主实例上运行时，以下示例将 Aurora 集群配置为充当在 Aurora 之外运行的某个 MySQL 实例的只读副本。

```
call mysql.rds_set_external_master_with_auto_position(
  'Externaldb.some.com',
  3306,
  'repl_user'@'mydomain.com',
  'SomePassW0rd');
```

mysql.rds_set_external_source_with_auto_position（Aurora MySQL 版本 3）

将 Aurora MySQL 主实例配置为接受来自外部 MySQL 实例的传入复制。此过程还会根据全局事务标识符 (GTID) 配置复制。

语法

```
CALL mysql.rds_set_external_source_with_auto_position (
  host_name
  , host_port
  , replication_user_name
  , replication_user_password
  , ssl_encryption
);
```

参数

host_name

在 Aurora 之外运行以变为复制源的 MySQL 实例的主机名或 IP 地址。

host_port

在 Aurora 之外运行的要配置为复制源的 MySQL 实例使用的端口。如果网络配置包括转换端口号的安全 Shell (SSH) 端口复制，请指定由 SSH 公开的端口号。

replication_user_name

在对 Aurora 外部运行的 MySQL 实例上具有 `REPLICATION CLIENT` 和 `REPLICATION SLAVE` 权限的用户的 ID。建议您向专用于复制的账户提供外部实例。

replication_user_password

在 `replication_user_name` 中指定的用户 ID 的密码。

ssl_encryption

目前未实施该选项。默认值是 0。

> **Note**
>
> 使用 mysql.rds_set_binlog_source_ssl（Aurora MySQL 版本 3）启用 SSL 以进行二进制日志复制。

使用说明

对于 Aurora MySQL 数据库集群，您将在连接到主实例时调用此存储过程。

管理用户必须运行 `mysql.rds_set_external_source_with_auto_position` 过程。管理用户在充当复制目标的 Aurora MySQL 数据库集群的主实例上运行此过程。这可能是外部 MySQL 数据库实例或 Aurora MySQL 数据库集群的复制目标。

仅 Aurora MySQL 版本 3 支持此过程。此过程不会配置延迟复制，因为 Aurora MySQL 不支持延迟复制。

在运行 `mysql.rds_set_external_source_with_auto_position` 前，请将外部 MySQL 数据库实例配置为复制源。要连接到外部 MySQL 实例，应指定 `replication_user_name` 和

replication_user_password 值。这些值必须指示具有外部 MySQL 实例上的 REPLICATION CLIENT 和 REPLICATION SLAVE 权限的复制用户。

要将外部 MySQL 实例配置为复制源

1. 通过使用所选的 MySQL 客户端，连接到外部 MySQL 实例并创建要用于复制的用户账户。以下是示例。

```
CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED BY 'SomePassW0rd'
```

2. 在外部 MySQL 实例上，向复制用户授予 REPLICATION CLIENT 和 REPLICATION SLAVE 权限。以下示例为您的域的 REPLICATION CLIENT 用户授予所有数据库的 REPLICATION SLAVE 和 'repl_user' 权限。

```
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com'
IDENTIFIED BY 'SomePassW0rd'
```

在调用 mysql.rds_set_external_source_with_auto_position 时，Amazon RDS 将记录某些信息。这些信息包括 "set master" 和 mysql.rds_history 表中的时间、用户和 mysql.rds_replication_status 操作。

要跳过已知会导致问题的基于 GTID 的特定事务，您可以使用 mysql.rds_skip_transaction_with_gtid（Aurora MySQL 版本 2 和 3）存储过程。有关使用基于 GTID 的复制的更多信息，请参阅使用基于 GTID 的复制。

示例

在 Aurora 主实例上运行时，以下示例将 Aurora 集群配置为充当在 Aurora 之外运行的某个 MySQL 实例的只读副本。

```
call mysql.rds_set_external_source_with_auto_position(
  'Externaldb.some.com',
  3306,
  'repl_user'@'mydomain.com',
  'SomePassW0rd');
```

mysql.rds_set_master_auto_position（Aurora MySQL 版本 2）

将复制模式设置为基于二进制日志文件位置或全局事务标识符 (GTID)。

语法

```
CALL mysql.rds_set_master_auto_position (
auto_position_mode
);
```

参数

auto_position_mode

该值指示是使用日志文件位置复制还是基于 GTID 的复制：

- 0 – 使用基于二进制日志文件位置的复制方法。默认为 0。
- 1 – 使用基于 GTID 的复制方法。

使用说明

主用户必须运行 mysql.rds_set_master_auto_position 过程。

Aurora MySQL 版本 2 支持此过程。

mysql.rds_set_read_only（Aurora MySQL 版本 3）

为数据库实例全局开启或关闭 read_only 模式。

语法

```
CALL mysql.rds_set_read_only(mode);
```

参数

mode

指示是否为数据库实例全局启用 read_only 模式的值：

- 0 – OFF。默认值为 0。
- 1 – ON

使用说明

mysql.rds_set_read_only 存储过程仅修改 read_only 参数。无法在读取器数据库实例上更改 innodb_read_only 参数。

read_only 参数更改在重启时不会保留。要对 read_only 进行永久更改，必须使用 read_only 数据库集群参数。

Aurora MySQL 版本 3.06 及更高版本支持此过程。

mysql.rds_set_session_binlog_format（Aurora MySQL 版本 2）

设置当前会话的二进制日志格式。

语法

```
CALL mysql.rds_set_session_binlog_format(format);
```

参数

format

一个表示当前会话的二进制日志格式的值：

- STATEMENT – 复制源基于 SQL 语句将事件写入二进制日志。
- ROW – 复制源将事件写入二进制日志，指示各个表行发生更改。
- MIXED – 日志记录通常基于 SQL 语句，但在某些条件下切换到行。有关更多信息，请参阅 MySQL 文档中的混合二进制日志记录格式。

使用说明

对于 Aurora MySQL 数据库集群，您将在连接到主实例时调用此存储过程。

要使用此存储过程，必须为当前会话配置二进制日志记录。

对于 Aurora，Aurora MySQL 版本 2.12 及更高的与 MySQL 5.7 兼容的版本支持此过程。

mysql.rds_set_source_auto_position（Aurora MySQL 版本 3）

将复制模式设置为基于二进制日志文件位置或全局事务标识符 (GTID)。

语法

```
CALL mysql.rds_set_source_auto_position (auto_position_mode);
```

参数

auto_position_mode

该值指示是使用日志文件位置复制还是基于 GTID 的复制：

- 0 – 使用基于二进制日志文件位置的复制方法。默认为 0。
- 1 – 使用基于 GTID 的复制方法。

使用说明

对于 Aurora MySQL 数据库集群，您将在连接到主实例时调用此存储过程。

管理用户必须运行 mysql.rds_set_source_auto_position 过程。

mysql.rds_skip_repl_error

跳过并删除 MySQL 数据库只读副本上的复制错误。

语法

```
CALL mysql.rds_skip_repl_error;
```

使用说明

主用户必须对只读副本运行 mysql.rds_skip_repl_error 过程。有关此过程的更多信息，请参阅跳过当前的复制错误。

要确定是否存在错误，请运行 MySQL SHOW REPLICA STATUS\G 命令。如果复制错误不太严重，您可以运行 mysql.rds_skip_repl_error 以跳过该错误。如果有多个错误，mysql.rds_skip_repl_error 会删除第一个错误，并警告存在其他错误。然后，您可以使用 SHOW REPLICA STATUS\G 确定要对下一个错误采取的适当操作。有关返回的值的信息，请参阅 MySQL 文档中的 SHOW REPLICA STATUS 语法。

有关解决 Aurora MySQL 的复制错误的信息，请参阅诊断并解决 MySQL 读取复制故障。

复制已停止错误

调用 mysql.rds_skip_repl_error 过程时，您可能会收到一条错误消息，指出副本已关闭或禁用。

如果您对于主实例而不是只读副本运行该过程，则会出现此错误消息。您必须对只读副本运行此过程，该过程才能正常运行。

如果您对只读副本运行该过程，但无法成功重新启动复制，也可能会出现此错误消息。

如果您需要跳过大量错误，复制滞后时间可能会超出二进制日志 (binlog) 文件的默认保留期。在这种情况下，您可能会遇到一个因二进制日志文件在只读副本上进行重放之前被清除而引发的严重错误。此清除会导致复制停止，而您将无法再调用 `mysql.rds_skip_repl_error` 命令以跳过复制错误。

您可以增加在源数据库实例上保留 binlog 文件的小时数以缓解该问题。在增加二进制日志保留时间后，您可以重新启动复制进程，并根据需要调用 `mysql.rds_skip_repl_error` 命令。

要设置 binlog 保留时间，请使用 mysql.rds_set_configuration 过程，并指定 `'binlog retention hours'` 配置参数以及在数据库集群上保留 binlog 文件的小时数。以下示例将 binlog 文件的保留期设置为 48 个小时。

```
CALL mysql.rds_set_configuration('binlog retention hours', 48);
```

mysql.rds_start_replication

从 Aurora MySQL 数据库集群发起复制。

> **Note**
>
> 您可以使用 mysql.rds_start_replication_until（Aurora MySQL 版本 3）或 mysql.rds_start_replication_until_gtid（Aurora MySQL 版本 3）存储过程从 Aurora MySQL 数据库实例中启动复制，并在指定的二进制日志文件位置停止复制。

语法

```
CALL mysql.rds_start_replication;
```

使用说明

主用户必须运行 `mysql.rds_start_replication` 过程。

要从 Amazon RDS 外部的 MySQL 实例导入数据，请在调用 mysql.rds_set_external_master（Aurora MySQL 版本 2）或 mysql.rds_set_external_source（Aurora MySQL 版本 3）以构建复制配置后，

再对只读副本调用 mysql.rds_start_replication 以开始复制过程。有关更多信息，请参阅 Aurora 与 MySQL 之间或 Aurora 与其他 Aurora 数据库集群之间的复制（二进制日志复制）。

要将数据导出到 Amazon RDS 外部的 MySQL 实例，请对只读副本调用 mysql.rds_start_replication 和 mysql.rds_stop_replication 以控制某些复制操作，如清除二进制日志。有关更多信息，请参阅 Aurora 与 MySQL 之间或 Aurora 与其他 Aurora 数据库集群之间的复制（二进制日志复制）。

您还可以对只读副本调用 mysql.rds_start_replication，以便重新启动您先前通过调用 mysql.rds_stop_replication 停止的任何复制过程。有关更多信息，请参阅 复制已停止错误。

mysql.rds_start_replication_until（Aurora MySQL 版本 3）

从 Aurora MySQL 数据库集群发起复制，并在指定的二进制日志文件位置停止复制。

语法

```
CALL mysql.rds_start_replication_until (
replication_log_file
  , replication_stop_point
);
```

参数

replication_log_file

源数据库实例上包含复制信息的二进制日志的名称。

replication_stop_point

replication_log_file 二进制日志中复制将停止的位置。

使用说明

主用户必须运行 mysql.rds_start_replication_until 过程。

Aurora MySQL 版本 3.04 及更高版本支持该过程。

托管式复制不支持 mysql.rds_start_replication_until 存储过程，其中包括以下内容：

- 跨 AWS 区域复制 Amazon Aurora MySQL 数据库集群

- 使用 Aurora 只读副本将数据从 RDS for MySQL 数据库实例迁移到 Amazon Aurora MySQL 数据库集群

为 replication_log_file 参数指定的文件名必须与源数据库实例二进制日志文件名匹配。

当 replication_stop_point 参数指定位于过去的某个停止位置时，即会立即停止复制。

示例

以下示例将启动复制并复制更改，直到它到达 120 二进制日志文件中的 mysql-bin-changelog.000777 位置。

```
call mysql.rds_start_replication_until(
  'mysql-bin-changelog.000777',
  120);
```

mysql.rds_stop_replication

停止从 MySQL 数据库实例中进行复制。

语法

```
CALL mysql.rds_stop_replication;
```

使用说明

主用户必须运行 mysql.rds_stop_replication 过程。

如果要配置复制，使其从在 Amazon RDS 外部运行的 MySQL 实例导入数据，则要在导入完成后，再对只读副本调用 mysql.rds_stop_replication 以停止复制过程。有关更多信息，请参阅 Aurora 与 MySQL 之间或 Aurora 与其他 Aurora 数据库集群之间的复制（二进制日志复制）。

如果要配置复制，使其将数据导出到在 Amazon RDS 外部运行的 MySQL 实例，则要对只读副本调用 mysql.rds_start_replication 和 mysql.rds_stop_replication 以控制某些复制操作，如清除二进制日志。有关更多信息，请参阅 Aurora 与 MySQL 之间或 Aurora 与其他 Aurora 数据库集群之间的复制（二进制日志复制）。

托管式复制不支持 mysql.rds_stop_replication 存储过程，其中包括以下内容：

- 跨 AWS 区域复制 Amazon Aurora MySQL 数据库集群

- 使用 Aurora 只读副本将数据从 RDS for MySQL 数据库实例迁移到 Amazon Aurora MySQL 数据库集群

结束会话或查询

以下存储过程结束会话或查询。

主题

- mysql.rds_kill
- mysql.rds_kill_query

mysql.rds_kill

结束与 MySQL 服务器的连接。

语法

```
CALL mysql.rds_kill(processID);
```

参数

processID

要结束的连接线程的标识。

使用说明

与 MySQL 服务器的每个连接在单独的线程中运行。要结束连接，请使用 `mysql.rds_kill` 过程并传入该连接的线程 ID。要获取线程 ID，请使用 MySQL SHOW PROCESSLIST 命令。

示例

以下示例结束线程 ID 为 4243 的连接：

```
CALL mysql.rds_kill(4243);
```

mysql.rds_kill_query

结束针对 MySQL 服务器运行的查询。

语法

```
CALL mysql.rds_kill_query(processID);
```

参数

processID

运行正要结束的查询的进程或线程的身份。

使用说明

要停止针对 MySQL 服务器运行的查询，请使用 mysql_rds_kill_query 过程并传入正在运行查询的线程的连接 ID。然后，该过程将终止连接。

要获取 ID，请查询 MySQL INFORMATION_SCHEMA PROCESSLIST 表或使用 MySQL SHOW PROCESSLIST 命令。SHOW PROCESSLIST 或 SELECT * FROM INFORMATION_SCHEMA.PROCESSLIST 中 ID 列的值为 *processID*。

示例

以下示例停止查询线程 ID 为 230040 的查询：

```
CALL mysql.rds_kill_query(230040);
```

使用 GTID 复制事务

以下存储过程控制如何使用 Aurora MySQL 中的全局事务标识符（GTID）复制事务。要了解如何根据 Aurora MySQL 中的 GTID 使用复制，请参阅使用基于 GTID 的复制。

主题

- mysql.rds_assign_gtids_to_anonymous_transactions（Aurora MySQL 版本 3）
- mysql.rds_gtid_purged（Aurora MySQL 版本 3）
- mysql.rds_skip_transaction_with_gtid（Aurora MySQL 版本 2 和 3）
- mysql.rds_start_replication_until_gtid（Aurora MySQL 版本 3）

mysql.rds_assign_gtids_to_anonymous_transactions（Aurora MySQL 版本 3）

配置 `CHANGE REPLICATION SOURCE TO` 语句的 `ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS` 选项。它使得复制通道将 GTID 分配给没有 GTID 的复制事务。这样，您就可以将二进制日志从不使用基于 GTID 的复制的源复制到使用该复制的副本。有关更多信息，请参阅 MySQL 参考手册中的 CHANGE REPLICATION SOURCE TO 语句和从没有 GTID 的源复制到有 GTID 的副本。

语法

```
CALL mysql.rds_assign_gtids_to_anonymous_transactions(gtid_option);
```

参数

gtid_option

字符串值。允许的值为 `OFF`、`LOCAL` 或者指定的 UUID。

使用说明

此过程与在社群 MySQL 中发布语句 `CHANGE REPLICATION SOURCE TO ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS = gtid_option` 的效果相同。

GTID 必须转向 `ON` 才能使 *gtid_option* 设置为 `LOCAL` 或指定的 UUID。

默认值为 `OFF`，这意味着不使用该功能。

`LOCAL` 会分配一个 GTID，其中包括副本自己的 UUID（`server_uuid` 设置）。

传递一个作为 UUID 的参数会分配一个包含指定 UUID 的 GTID，例如复制源服务器的 server_uuid 设置。

示例

要关闭此功能：

```
mysql> call mysql.rds_assign_gtids_to_anonymous_transactions('OFF');
+-------------------------------------------------------------+
| Message  |
+-------------------------------------------------------------+
| ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS has been set to: OFF |
+-------------------------------------------------------------+
1 row in set (0.07 sec)
```

要使用副本自己的 UUID：

```
mysql> call mysql.rds_assign_gtids_to_anonymous_transactions('LOCAL');
+--------------------------------------------------------------+
| Message  |
+--------------------------------------------------------------+
| ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS has been set to: LOCAL |
+--------------------------------------------------------------+
1 row in set (0.07 sec)
```

要使用指定的 UUID：

```
mysql> call mysql.rds_assign_gtids_to_anonymous_transactions('317a4760-
f3dd-3b74-8e45-0615ed29de0e');
+-----------------------------------------------------------------------------------------------------
+
| Message |
+-----------------------------------------------------------------------------------------------------
+
| ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS has been set to: 317a4760-
f3dd-3b74-8e45-0615ed29de0e |
+-----------------------------------------------------------------------------------------------------
+
1 row in set (0.07 sec)
```

mysql.rds_gtid_purged（Aurora MySQL 版本 3）

将系统变量 gtid_purged 的全局值设置为给定的全局事务标识符（GTID）集。gtid_purged 系统变量是一个 GTID 集，由服务器上已提交但不存在于服务器上任何二进制日志文件中的所有事务的 GTID 组成。

为了与 MySQL 8.0 兼容，有两种方法可以设置 gtid_purged 的值：

- 将 gtid_purged 的值替换为指定的 GTID 集。
- 将您指定的 GTID 集附加到 gtid_purged 已经包含的 GTID 集。

语法

要将 gtid_purged 的值替换为您指定的 GTID 集，请执行以下操作：

```
CALL mysql.rds_gtid_purged (gtid_set);
```

要将 gtid_purged 的值附加到您指定的 GTID 集，请执行以下操作：

```
CALL mysql.rds_gtid_purged (+gtid_set);
```

参数

gtid_set

gtid_set 的值必须是 gtid_purged 的当前值的超集，并且不能与 gtid_subtract(gtid_executed,gtid_purged) 相交。也就是说，新的 GTID 集必须包括 gtid_purged 中已经存在的任何 GTID，并且不能包括 gtid_executed 中尚未清除的任何 GTID。*gtid_set* 参数也不能包括全局 gtid_owned 集中的任何 GTID，即当前服务器上正在处理的事务的 GTID。

使用说明

主用户必须运行 mysql.rds_gtid_purged 过程。

Aurora MySQL 版本 3.04 及更高版本支持此过程。

示例

以下示例将 GTID 3E11FA47-71CA-11E1-9E33-C80AA9429562:23 分配给 gtid_purged 全局变量。

```
CALL mysql.rds_gtid_purged('3E11FA47-71CA-11E1-9E33-C80AA9429562:23');
```

mysql.rds_skip_transaction_with_gtid（Aurora MySQL 版本 2 和 3）

在 Aurora 主实例上跳过复制具有指定全局事务标识符 (GTID) 的事务。

在已知特定 GTID 事务导致问题时，您可以使用该过程进行灾难恢复。请使用该存储过程跳过有问题的事务。有问题的事务示例包括禁用复制、删除重要数据或导致数据库实例变得不可用的事务。

语法

```
CALL mysql.rds_skip_transaction_with_gtid (
gtid_to_skip
);
```

参数

gtid_to_skip

要跳过的复制事务的 GTID。

使用说明

主用户必须运行 mysql.rds_skip_transaction_with_gtid 过程。

Aurora MySQL 版本 2 和 3 支持此过程。

示例

以下示例将跳过复制具有 GTID 3E11FA47-71CA-11E1-9E33-C80AA9429562:23 的事务。

```
CALL mysql.rds_skip_transaction_with_gtid('3E11FA47-71CA-11E1-9E33-C80AA9429562:23');
```

mysql.rds_start_replication_until_gtid（Aurora MySQL 版本 3）

从 Aurora MySQL 数据库集群中启动复制，并在指定的全局事务标识符（GTID）后面立即停止复制。

语法

```
CALL mysql.rds_start_replication_until_gtid(gtid);
```

参数

gtid

停止复制前的 GTID。

使用说明

主用户必须运行 mysql.rds_start_replication_until_gtid 过程。

Aurora MySQL 版本 3.04 及更高版本支持该过程。

托管式复制不支持 mysql.rds_start_replication_until_gtid 存储过程，其中包括以下内容：

- 跨 AWS 区域复制 Amazon Aurora MySQL 数据库集群
- 使用 Aurora 只读副本将数据从 RDS for MySQL 数据库实例迁移到 Amazon Aurora MySQL 数据库集群

在 gtid 参数指定副本已运行的事务时，将会立即停止复制。

示例

以下示例启动复制并复制更改，直至到达 GTID 3E11FA47-71CA-11E1-9E33-C80AA9429562:23。

```
call mysql.rds_start_replication_until_gtid('3E11FA47-71CA-11E1-9E33-C80AA9429562:23');
```

轮换查询日志

以下存储过程将 MySQL 日志交替到备份表。有关更多信息，请参阅 Aurora MySQL 数据库日志文件。

主题

- mysql.rds_rotate_general_log
- mysql.rds_rotate_slow_log

mysql.rds_rotate_general_log

将 mysql.general_log 表轮换到备份表。

语法

```
CALL mysql.rds_rotate_general_log;
```

使用说明

您可以通过调用 mysql.general_log 过程将 mysql.rds_rotate_general_log 表轮换到备份表。轮换日志表时，会将当前日志表复制到备份日志表，随后删除当前日志表中的条目。如果备份日志表已存在，则先将其删除，然后将当前日志表复制到备份。如果需要，您可以查询备份日志表。mysql.general_log 表的备份日志表名为 mysql.general_log_backup。

只有当 log_output 参数设置为 TABLE 时，您才能运行此过程。

mysql.rds_rotate_slow_log

将 mysql.slow_log 表轮换到备份表。

语法

```
CALL mysql.rds_rotate_slow_log;
```

使用说明

您可以通过调用 mysql.slow_log 过程将 mysql.rds_rotate_slow_log 表轮换到备份表。轮换日志表时，会将当前日志表复制到备份日志表，随后删除当前日志表中的条目。如果备份日志表已存在，则先将其删除，然后将当前日志表复制到备份。

如果需要，您可以查询备份日志表。mysql.slow_log 表的备份日志表名为 mysql.slow_log_backup。

设置和显示二进制日志配置

以下存储过程设置和显示配置参数，例如用于二进制日志文件保留。

主题

- mysql.rds_set_configuration
- mysql.rds_show_configuration

mysql.rds_set_configuration

指定要保留二进制日志的小时数或要延迟复制的秒数。

语法

```
CALL mysql.rds_set_configuration(name,value);
```

参数

name

要设置的配置参数的名称。

#

配置参数的值。

使用说明

`mysql.rds_set_configuration` 过程支持以下配置参数：

- 二进制日志保留小时数

配置参数将永久存储，可在任何数据库实例重启或失效转移后继续使用。

二进制日志保留小时数

`binlog retention hours` 参数用于指定要保留二进制日志文件的小时数。Amazon Aurora 通常会尽快清除一个二进制日志，但对于 Amazon 外部的 MySQL 数据库的复制，该二进制日志可能仍是必需的。

binlog retention hours 的默认值为 NULL。对于 Aurora MySQL，NULL 表示延时清理二进制日志。Aurora MySQL 二进制日志可能会在系统中保留一段时间，但这通常不超过一天。

要指定在数据库集群上保留二进制日志的小时数，请使用 mysql.rds_set_configuration 存储过程并指定足以让复制发生的时段，如以下示例中所示。

```
call mysql.rds_set_configuration('binlog retention hours', 24);
```

> **Note**
>
> 不能将值 0 用于 binlog retention hours。

对于 Aurora MySQL 版本 2.11.0 及更高版本和版本 3 数据库集群，最大 binlog retention hours 值为 2160（90 天）。

在设置保留期后，监视数据库实例的存储用量以确认保留的二进制日志不会占用太多存储空间。

mysql.rds_show_configuration

保留二进制日志的小时数。

语法

```
CALL mysql.rds_show_configuration;
```

使用说明

要验证 Amazon RDS 保留二进制日志的小时数，请使用 mysql.rds_show_configuration 存储过程。

示例

以下示例显示了保留期：

```
call mysql.rds_show_configuration;
                name                          value     description
                binlog retention hours        24        binlog retention hours specifies
 the duration in hours before binary logs are automatically deleted.
```

Aurora MySQL 特定的 information_schema 表

Aurora MySQL 有一些特定于 Aurora 的 `information_schema` 表。

information_schema.aurora_global_db_instance_status

`information_schema.aurora_global_db_instance_status` 表包含有关全局数据库主数据库集群和辅助数据库集群中所有数据库实例状态的信息。下表显示了您可以使用的列。其余列仅供 Aurora 内部使用。

> **Note**
>
> 此信息模式表仅适用于 Aurora MySQL 版本 3.04.0 及更高版本的全局数据库。

列	数据类型	描述
SERVER_ID	varchar(100)	数据库实例的标识符。
SESSION_ID	varchar(100)	当前会话的唯一标识符。`MASTER_SESSION_ID` 的值标识写入器（主要）数据库实例。
AWS_REGION	varchar(100)	运行此全局数据库实例的 AWS 区域。有关区域列表，请参阅区域可用性。
DURABLE_LSN	bigint unsigned	在存储中变得持久的日志序列号（LSN）。日志序列号 (LSN) 是标识数据库事务日志中的记录的唯一序列号。对 LSN 进行排序，以便较大的 LSN 表示较晚的事务。
HIGHEST_LSN_RCVD	bigint unsigned	数据库实例从写入器数据库实例收到的最高 LSN。

列	数据类型	描述
OLDEST_READ_VIEW_TRX_ID	bigint unsigned	写入器数据库实例可以清除到的最早事务的 ID。
OLDEST_READ_VIEW_LSN	bigint unsigned	数据库实例从存储中读取所用的最早 LSN。
VISIBILITY_LAG_IN_MSEC	float(10,0) unsigned	对于主数据库集群中的读取器，此数据库实例滞后于写入器数据库实例的时间（以毫秒为单位）。对于辅助数据库集群中的读取器，此数据库实例滞后于辅助卷的时间（以毫秒为单位）。

information_schema.aurora_global_db_status

`information_schema.aurora_global_db_status` 表包含有关 Aurora 全局数据库滞后各方面的信息，特别是底层 Aurora 存储的滞后（所谓持久性滞后）以及恢复点目标（RPO）之间的滞后。下表显示了您可以使用的列。其余列仅供 Aurora 内部使用。

> **Note**
>
> 此信息模式表仅适用于 Aurora MySQL 版本 3.04.0 及更高版本的全局数据库。

列	数据类型	描述
AWS_REGION	varchar(100)	运行此全局数据库实例的 AWS 区域。有关区域列表，请参阅区域可用性。
HIGHEST_LSN_WRITTEN	bigint unsigned	此数据库集群上当前存在的最高日志序列号（LSN）。日志序列号 (LSN) 是标识数据库事务日志中的记录的唯一序列

列	数据类型	描述
		号。对 LSN 进行排序，以便较大的 LSN 表示较晚的事务。
DURABILITY_LAG_IN_MILLISECONDS	float(10,0) unsigned	辅助数据库集群上的 `HIGHEST_LSN_WRITTEN` 与主数据库集群上的 `HIGHEST_LSN_WRITTEN` 之间的时间戳值差异。在 Aurora 全局数据库的主数据库集群上，该值始终为 0。
RPO_LAG_IN_MILLISECONDS	float(10,0) unsigned	恢复点目标（RPO）滞后。RPO 滞后是最近的用户事务在存储在 Aurora 全局数据库的主数据库集群上之后，执行 COMMIT 操作以便存储在辅助数据库集群上所需的时间。在 Aurora 全局数据库的主数据库集群上，该值始终为 0。 简而言之，该指标计算 Aurora 全局数据库中每个 Aurora MySQL 数据库集群的恢复点目标，也就是说，如果发生中断，可能会丢失多少数据。与滞后一样，RPO 是按时间计量的。
LAST_LAG_CALCULATION_TIMESTAMP	datetime	指定上次为 `DURABILITY_LAG_IN_MILLISECONDS` 和 `RPO_LAG_IN_MILLISECONDS` 计算值的时间的时间戳。时间值（如 `1970-01-01 00:00:00+00` ）表示这是主数据库集群。

列	数据类型	描述
OLDEST_READ_VIEW_TRX_ID	bigint unsigned	写入器数据库实例可以清除到的最早事务的 ID。

information_schema.replica_host_status

`information_schema.replica_host_status` 表包含复制信息。您可以使用的列如下表所示。其余列仅供 Aurora 内部使用。

列	数据类型	描述
CPU	double	副本主机的 CPU 使用百分比。
IS_CURRENT	tinyint	副本是否为最新副本。
LAST_UPDATE_TIMESTAMP	datetime(6)	上次更新发生的时间。用于确定记录是否过时。
REPLICA_LAG_IN_MILLISECONDS	double	副本滞后，以毫秒为单位。
SERVER_ID	varchar(100)	数据库服务器的 ID。
SESSION_ID	varchar(100)	数据库会话的 ID。用于确定数据库实例是写入器实例还是读取器实例。

> **Note**
>
> 当副本实例滞后时，从其 `information_schema.replica_host_status` 表中查询到的信息可能已过时。在这种情况下，我们建议您改为从写入器实例进行查询。
> 虽然 `mysql.ro_replica_status` 表具有相似的信息，但我们不建议使用它。

information_schema.aurora_forwarding_processlist

`information_schema.aurora_forwarding_processlist` 表包含有关写入转发所涉及的进程的信息。

此表的内容仅在开启全局或集群内写入转发的数据库集群的写入器数据库实例上可见。读取器数据库实例上返回一个空的结果集。

字段	数据类型	描述
ID	bigint	写入器数据库实例上连接的标识符。该标识符与 `SHOW PROCESSLIST` 语句的 `Id` 列中显示的值相同，并由线程中的 `CONNECTION_ID()` 函数返回。
USER	varchar(32)	发出语句的 MySQL 用户。
HOST	varchar(255)	发出语句的 MySQL 客户端。对于转发的语句，此字段显示在转发读取器数据库实例上建立连接的应用程序客户端主机地址。
DB	varchar(64)	线程的原定设置数据库。
COMMAND	varchar(16)	线程代表客户端执行的命令的类型，或者，如果会话空闲，则为 `Sleep`。有关线程命令的描述，请参阅 MySQL 文档中有关线程命令值的部分。
TIME	int	线程处于其当前状态的时间（以秒为单位）。
STATE	varchar(64)	指示线程正在做什么的操作、事件或状态。有关状态值的描述，请参阅 MySQL 文档中的常规线程状态。
信息	longtext	线程正在执行的语句，或者，如果线程未执行语句，则为 `NULL`。该语句可能是发送到服务器的语句，或者，如果该语句执行其他语句，则为最内层的语句。
IS_FORWARDED	bigint	表示线程是不是从读取器数据库实例转发的。

字段	数据类型	描述
REPLICA_SESSION_ID	bigint	Aurora 副本上的连接标识符。该标识符与转发 Aurora 读取器数据库实例上 `SHOW PROCESSLIST` 语句的 `Id` 列中显示的值相同。
REPLICA_INSTANCE_IDENTIFIER	varchar(64)	转发线程的数据库实例标识符。
REPLICA_CLUSTER_NAME	varchar(64)	转发线程的数据库集群标识符。对于集群内写入转发，此标识符与写入器数据库实例的数据库集群相同。
REPLICA_REGION	varchar(64)	转发线程源自的 AWS 区域。对于集群内写入转发，此区域与写入器数据库实例的 AWS 区域相同。

Amazon Aurora MySQL 的数据库引擎更新

Amazon Aurora 定期发布更新。更新将在系统维护时段内应用于 Aurora 数据库集群。应用更新的时间取决于数据库集群的区域和维护时段设置以及更新的类型。

Amazon Aurora 版本在多天时间里陆续对所有 AWS 区域发布。某些区域可能会暂时显示另一区域尚不可用的引擎版本。

更新将同时应用于数据库集群中的所有实例。更新需要在数据库集群中的所有实例上重新启动数据库，因此，会出现 20 到 30 秒的停机，之后您可以继续使用数据库集群。您可以从 AWS Management Console查看或更改维护时段设置。

有关 Amazon Aurora 支持的 Aurora MySQL 版本的详细信息，请参阅 发布说明。

接下来，您可以了解如何为集群选择正确的 Aurora MySQL 版本、创建或升级集群时如何指定版本，以及以最小的中断将集群从一个版本升级到另一个版本的步骤。

主题

- 检查 Aurora MySQL 版本号
- Amazon Aurora MySQL 的长期支持（LTS）版本和测试版
- 为 Amazon Aurora MySQL 兼容版的版本 2 终止标准支持做好准备
- 准备终止使用 Amazon Aurora MySQL 兼容版的版本 1
- 升级 Amazon Aurora MySQL 数据库集群
- Amazon Aurora MySQL 的数据库引擎更新和修复

检查 Aurora MySQL 版本号

虽然 Aurora MySQL 兼容版 与 MySQL 数据库引擎兼容，但 Aurora MySQL 包括特定于特定 Aurora MySQL 版本的功能和错误修复。应用程序开发人员可以使用 SQL 检查其应用程序中的 Aurora MySQL 版本。数据库管理员可以在创建或升级 Aurora MySQL 数据库集群和数据库实例时检查和指定 Aurora MySQL 版本。

主题

- 通过AWS检查或指定 Aurora MySQL 引擎版本
- 使用 SQL 检查 Aurora MySQL 版本

通过AWS检查或指定 Aurora MySQL 引擎版本

使用AWS Management Console、AWS CLI 或 RDS API 执行管理任务时，您可以使用描述性字母数字格式指定 Aurora MySQL 版本。

从 Aurora MySQL 版本 2 开始，Aurora 引擎版本具有以下语法。

```
mysql-major-version.mysql_aurora.aurora-mysql-version
```

mysql-major-version- 部分为 5.7 或 8.0。此值表示客户端协议的版本和相应 Aurora MySQL 版本的 MySQL 功能支持的一般级别。

The *aurora-mysql-version* 是一个包含三个部分的点分值：Aurora MySQL 主要版本、Aurora MySQL 次要版本和补丁级别。主要版本为 2 或 3。这些值分别表示与 MySQL 5.7 或 8.0 兼容的 Aurora MySQL。次要版本表示 2.x 或 3.x 系列中的功能版本。对于每个次要版本，补丁级别从 0 开始，表示应用于次要版本的后续错误修复集。有时，新功能会合并到次要版本中，但不会立即显示出来。在这些情况下，该功能会进行微调，并在以后的补丁级别中公开。

所有 2.x Aurora MySQL 引擎版本都与 Community MySQL 5.7.12 或更高版本兼容。所有 3.x Aurora MySQL 引擎版本都与 MySQL 8.0.23 或更高版本兼容。您可以参考特定 3.x 版本的版本注释来了解相应的 MySQL 兼容版本。

例如，Aurora MySQL 3.04.0 和 2.11.2 的引擎版本如下所示。

```
8.0.mysql_aurora.3.04.0
5.7.mysql_aurora.2.11.2
```

> **Note**
>
> 社群 MySQL 版本与 Aurora MySQL 2.x 版本之间没有一一对应关系。对于 Aurora MySQL 版本 3，有更直接的映射。要检查特定 Aurora MySQL 版本中有哪些错误修复和新功能，请参阅《Aurora MySQL 版本注释》中的 Amazon Aurora MySQL 版本 3 的数据库引擎更新和 Amazon Aurora MySQL 版本 2 的数据库引擎更新。有关新功能和版本的时间顺序列表，请参阅文档历史记录。要检查与安全相关的修复所需的最低版本，请参阅《Aurora MySQL 发布说明》中的 Aurora MySQL 中修复的安全漏洞。

您可以在一些 AWS CLI 命令和 RDS API 操作中指定 Aurora MySQL 引擎版本。例如，您可以在运行 `--engine-version` 命令 create-db-cluster 和 modify-db-cluster 时指定 AWS CLI 选项。您可以在运行 RDS API 操作 CreateDBCluster 和 ModifyDBCluster 时指定 `EngineVersion` 参数。

在 Aurora MySQL 版本 2 及更高版本中，AWS Management Console中的引擎版本还包含 Aurora 版本。升级集群将更改显示的值。这种更改可帮助您指定和检查精确的 Aurora MySQL 版本，而无需连接到集群或运行任何 SQL 命令。

> **Tip**
>
> 对于通过 AWS CloudFormation 管理的 Aurora 集群，`EngineVersion` 设置中的此更改可通过 AWS CloudFormation 触发操作。有关 AWS CloudFormation 如何处理对 `EngineVersion` 设置的更改的信息，请参阅 AWS CloudFormation文档。

使用 SQL 检查 Aurora MySQL 版本

您可以使用 SQL 查询在应用程序中检索的 Aurora 版本号使用格式 *<major version>*.*<minor version>*.*<patch version>*。您可以通过查询 AURORA_VERSION 系统变量，获取 Aurora MySQL 集群中任何数据库实例的此版本号。要获取此版本号，请使用以下查询之一。

```
select aurora_version();
select @@aurora_version;
```

这些查询会产生类似于以下内容的输出。

```
mysql> select aurora_version(), @@aurora_version;
+------------------+------------------+
| aurora_version() | @@aurora_version |
+------------------+------------------+
| 3.05.2           | 3.05.2           |
+------------------+------------------+
```

控制台、CLI 和 RDS API 使用 通过AWS检查或指定 Aurora MySQL 引擎版本 中描述的技术返回的版本号通常更具描述性。

Amazon Aurora MySQL 的长期支持（LTS）版本和测试版

Aurora MySQL 为某些 Aurora MySQL 引擎版本提供长期支持（LTS）版本和测试版。

主题

- Aurora MySQL 长期支持 (LTS) 版本
- Aurora MySQL 测试版

Aurora MySQL 长期支持 (LTS) 版本

每个新的 Aurora MySQL 版本在一定时间内保持可用状态，以便在创建或升级数据库集群时使用。在此期间后，您必须升级使用该版本的任何集群。您可以在支持期限结束之前手动升级集群，或者 Aurora 可以在不再支持集群的 Aurora MySQL 版本时自动升级集群。

Aurora 将某些 Aurora MySQL 版本指定为长期支持 (LTS) 版本。与使用非 LTS 版本的数据库集群相比，使用 LTS 版本的集群可以将同一版本保留更长时间，并执行更少的升级周期。Aurora 会在每个 LTS 版本发布后，对该版本提供至少三年的支持。如果需要升级具有 LTS 版本的数据库集群，Aurora 将其升级到下一个 LTS 版本。这样，在长时间内不需要再次升级集群。

在 Aurora MySQL LTS 版本的使用期限内，新的补丁级别引入了重要问题的修复。补丁级别不包括任何新特征。您可以选择是否将此类补丁应用于运行 LTS 版本的数据库集群。对于某些关键修复，Amazon 可能会托管升级到同一 LTS 版本中的补丁级别。此类托管升级是在集群维护窗口内自动执行。

对于大多数 Aurora MySQL 集群，我们建议您升级到最新的版本，而不是使用 LTS 版本。这样做可以利用 Aurora 以作为托管服务，并使您可以获取最新的特征和错误修复。LTS 版本适用于具有以下特征的集群：

- 除了在极少数情况下应用关键补丁以外，您无法承受 Aurora MySQL 应用程序停机。
- 对于 Aurora MySQL 数据库引擎的每次更新，集群和关联的应用程序的测试周期需要很长的时间。
- Aurora MySQL 集群的数据库版本具有应用程序所需的所有数据库引擎特征和错误修复。

Aurora MySQL 的当前 LTS 版本如下：

- Aurora MySQL 版本 3.04.*。有关该 LTS 版本的更多详细信息，请参阅《Aurora MySQL 发布说明》中的 Amazon Aurora MySQL 版本 3 的数据库引擎更新。

> **Note**
>
> 我们建议您禁用 LTS 版本的自动次要版本升级。将 `AutoMinorVersionUpgrade` 参数设置为 `false`，或在 AWS Management Console上清除允许自动次要版本升级复选框。

如果您不禁用该功能，数据库集群可能会升级到非 LTS 版本，例如 3.05.2。

Aurora MySQL 测试版

Aurora MySQL 测试版是在数量有限的 AWS 区域中发布的仅包含安全修复的早期版本。随着下一个补丁发布，这些修复随后将在所有区域更广泛地部署。

测试版的编号与 Aurora MySQL 次要版本类似，但有额外的第四位数字，例如 2.12.0.1 或 3.05.0.1。

有关更多信息，请参阅《Aurora MySQL 发布说明》中的 Amazon Aurora MySQL 版本 2 的数据库引擎更新和 Amazon Aurora MySQL 版本 3 的数据库引擎更新。

为 Amazon Aurora MySQL 兼容版的版本 2 终止标准支持做好准备

Amazon Aurora MySQL 兼容版的版本 2（与 MySQL 5.7 兼容）计划于 2024 年 10 月 31 日终止提供标准支持。我们建议，在 Aurora MySQL 版本 2 达到其标准支持终止期之前，将运行 Aurora MySQL 版本 2 的所有集群升级到默认的 Aurora MySQL 版本 3（与 MySQL 8.0 兼容）或更高版本。2024 年 10 月 31 日，Amazon RDS 会自动将您的数据库注册到 Amazon RDS 扩展支持。如果您在 Aurora Serverless 版本 1 集群中运行 Amazon Aurora MySQL 版本 2（与 MySQL 5.7 兼容），则这对您不适用。如果您要将 Aurora Serverless 版本 1 集群升级到 Aurora MySQL 版本 3，请参阅 Aurora Serverless v1 数据库集群的升级路径。

您可以在 Aurora MySQL 主要版本的发布日历中找到 Aurora MySQL 主要版本即将终止支持的日期。

如果您的集群运行的是 Aurora MySQL 版本 2，则随着其标准支持终止日期的临近，您将会定期收到通知，告知您有关如何进行升级的最新信息。我们将定期更新此页面，为您提供最新信息。

标准支持时间表

1. 现在到 2024 年 10 月 31 日 - 您可以将 Aurora MySQL 版本 2（与 MySQL 5.7 兼容）的集群升级到 Aurora MySQL 版本 3（与 MySQL 8.0 兼容）。
2. 2024 年 10 月 31 日 – 在这一天，Aurora MySQL 版本 2 将达到标准支持终止日期，Amazon RDS 会自动将您的集群注册到 Amazon RDS 扩展支持。

我们将自动为您注册 RDS 扩展支持。有关更多信息，请参阅 Amazon Aurora 的 Amazon RDS 扩展支持。

查找受此终止使用流程影响的集群

要查找受此终止使用流程影响的群集，请使用以下过程。

> **Important**
>
> 请确保在资源所在的每个 AWS 区域中针对每个 AWS 账户按照这些说明操作。

控制台

查找 Aurora MySQL 版本 2 集群

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Databases（数据库）。
3. 在按数据库筛选框中，输入 5.7。
4. 检查引擎列中的 Aurora MySQL。

AWS CLI

要查找受此终止使用流程影响的集群，请使用 AWS CLI，调用 describe-db-clusters 命令。您可以使用以下示例脚本。

Example

```
aws rds describe-db-clusters --include-share --query 'DBClusters[?(Engine==`aurora-
mysql` && contains(EngineVersion,`5.7.mysql_aurora`))].{EngineVersion:EngineVersion,
 DBClusterIdentifier:DBClusterIdentifier, EngineMode:EngineMode}' --output table
 --region us-east-1

        +-----------------------------------------------------------------+
        |                       DescribeDBClusters                        |
        +---------------------+---------------+-------------------------+
        |         DBCI        |      EM       |          EV             |
        +---------------------+---------------+-------------------------+
        |    aurora-mysql2    |  provisioned  | 5.7.mysql_aurora.2.11.3 |
        | aurora-serverlessv1 |   serverless  | 5.7.mysql_aurora.2.11.3 |
        +---------------------+---------------+-------------------------+
```

RDS API

要查找运行 Aurora MySQL 版本 2 的 Aurora MySQL 数据库集群，请将 RDS DescribeDBClusters API 操作与以下所需参数结合使用：

- `DescribeDBClusters`
  - Filters.Filter.N
    - Name
      - engine
    - Values.Value.N
      - ['aurora']

Amazon RDS Extended Support

在支持终止日期（2024 年 10 月 31 日）之前，您可以在社区 MySQL 5.7 上免费使用 Amazon RDS 扩展支持。2024 年 10 月 31 日，Amazon RDS 会自动将您的数据库注册到适用于 Aurora MySQL 版本 2 的 RDS 扩展支持。适用于 Aurora 的 RDS 扩展支持是一项付费服务，它额外提供长达 28 个月的 Aurora MySQL 版本 2 支持，直到 2027 年 2 月 RDS 扩展支持终止。RDS 扩展支持仅适用于 Aurora MySQL 次要版本 2.11 和 2.12。要在标准支持终止后继续使用 Amazon Aurora MySQL 版本 2，请在 2024 年 10 月 31 日之前，安排在其中一个次要版本上运行您的数据库。

有关 RDS 扩展支持的更多信息，例如费用和其它注意事项，请参阅Amazon Aurora 的 Amazon RDS 扩展支持。

执行升级

与次要版本相比，在主要版本之间升级需要更广泛的计划和测试。这个过程可能需要大量时间。我们希望分三个步骤来执行升级过程，包括升级之前、升级和升级之后的活动。

升级之前：

在升级之前，我们建议您针对升级的集群检查应用程序兼容性、性能、维护过程以及类似注意事项，确认升级后您的应用程序将按预期正常运行。以下是五项建议，有助于您获得更好的升级体验。

- 首先，了解Aurora MySQL 主要版本就地升级的工作原理至关重要。
- 接下来，了解当从 Aurora MySQL 版本 2 升级到版本 3 时可采用的升级技术。

- 为了帮助您决定正确的升级时间和方法，您可以通过 比较 Aurora MySQL 版本 2 和 Aurora MySQL 版本 3了解 Aurora MySQL 版本 3 与您的当前环境之间的区别：
- 在确定了方便且效果最佳的选项后，使用为 Aurora MySQL 集群计划主要版本升级尝试在克隆的集群上模拟就地升级。
- 查看 Aurora MySQL 的主要版本升级预检查。升级预检查器可以运行并确定您的数据库是否可以成功升级，升级后是否存在任何应用程序不兼容问题，以及性能、维护过程和类似注意事项。
- 并非所有类型或版本的 Aurora MySQL 集群都可以使用就地升级机制。有关更多信息，请参阅 Aurora MySQL 主要版本升级路径。

如果您有任何问题或疑问，可通过社区论坛和 Premium Support 联系 AWS Support 团队。

执行升级：

您可以使用下面的升级技术之一。系统将经历的停机时间取决于所选择的技术。

- 蓝绿部署 - 对于首要任务是减少应用程序停机时间的情况，您可以使用 Amazon RDS 蓝绿部署在预调配的 Amazon Aurora 数据库集群中执行主要版本升级。蓝绿部署会创建一个复制生产环境的暂存环境。您可以在绿色（暂存）环境中对 Aurora 数据库集群进行特定更改，而不会影响生产工作负载。切换通常需要不到一分钟，不会丢失数据。有关更多信息，请参阅 适用于 Aurora 的 Amazon RDS 蓝绿部署概述。这样可以最大限度地减少停机时间，但需要您在执行升级时运行额外的资源。
- 就地升级 - 您可以执行就地升级，Aurora 将自动为您执行预检查过程，使集群脱机、备份集群、执行升级，然后使集群恢复联机。只需单击几下即可执行就地主要版本升级，并且不涉及其他协调或与其他集群的失效转移，但会涉及到停机。有关更多信息，请参阅 如何执行就地升级
- 快照还原 - 您可以通过从 Aurora MySQL 版本 2 快照还原到 Aurora MySQL 版本 3 集群，来升级 Aurora MySQL 版本 2 集群。为此，您应该按照拍摄快照并从快照还原的过程进行操作。此过程涉及到数据库中断，因为您要从快照执行还原。

升级后：

升级后，您需要密切监视系统（应用程序和数据库），并在必要时进行微调更改。严格遵循升级前的步骤可以最大限度地减少所需的更改。有关更多信息，请参阅 排除 Amazon Aurora MySQL 数据库性能故障。

要了解有关 Aurora MySQL 主要版本升级的方法、规划、测试和故障排除的更多信息，请务必仔细阅读升级 Amazon Aurora MySQL 数据库集群的主要版本，包括Aurora MySQL 就地升级的故障排除。另请注意，Aurora MySQL 版本 3 不支持某些实例类型。有关更多信息，请参阅 Amazon Aurora 数据库实例类。

Aurora Serverless v1 数据库集群的升级路径

与次要版本相比，在主要版本之间升级需要更广泛的计划和测试。这个过程可能需要大量时间。我们希望分三个步骤来执行升级过程，包括升级之前、升级和升级之后的活动。

Aurora MySQL 版本 2（与 MySQL 5.7 兼容）将继续获得 Aurora Serverless v1 集群的标准支持。

如果您想要升级至 Amazon Aurora MySQL 3（与 MySQL 8.0 兼容）并继续运行 Aurora Serverless，您可以使用 Amazon Aurora Serverless v2。要了解 Aurora Serverless v1 和 Aurora Serverless v2 之间的区别，请参阅 比较 Aurora Serverless v2 和 Aurora Serverless v1。

升级到 Aurora Serverless v2：您可以将 Aurora Serverless v1 集群升级到 Aurora Serverless v2。有关更多信息，请参阅 从 Aurora Serverless v1 集群升级到 Aurora Serverless v2。

准备终止使用 Amazon Aurora MySQL 兼容版的版本 1

Amazon Aurora MySQL 兼容版的版本 1（与 MySQL 5.6 兼容）计划于 2023 年 2 月 28 日终止使用。Amazon 建议您将运行 Aurora MySQL 版本 1 的所有集群（预置和 Aurora Serverless）升级到 Aurora MySQL 版本 2（与 MySQL 5.7 兼容）或 Aurora MySQL 版本 3（与 MySQL 8.0 兼容）。在 Aurora MySQL 版本 1 支持期结束之前进行此升级。

对于 Aurora 预置数据库集群，您可以通过多种方法完成从 Aurora MySQL 版本 1 到 Aurora MySQL 版本 2 的升级。您可以在 如何执行就地升级 中找到有关就地升级机制的说明。完成升级的另一种方法是制作 Aurora MySQL 版本 1 集群的快照，并将快照还原到 Aurora MySQL 版本 2 集群。或者，您可以按照多步骤流程操作，并排运行新旧集群。有关每种方法的更多详细信息，请参阅升级 Amazon Aurora MySQL 数据库集群的主要版本。

对于 Aurora Serverless v1 数据库集群，您可以执行从 Aurora MySQL 版本 1 到 Aurora MySQL 版本 2 的就地升级。有关此方法的更多详细信息，请参阅修改 Aurora Serverless v1 数据库集群。

对于 Aurora 预调配数据库集群，您可以通过使用两阶段升级过程，完成从 Aurora MySQL 版本 1 到 Aurora MySQL 版本 3 的升级：

1. 使用上述方法从 Aurora MySQL 版本 1 升级到 Aurora MySQL 版本 2。
2. 使用与从 Aurora MySQL 版本 1 升级到 Aurora MySQL 版本 2 相同的方法，从版本 2 升级到版本 3。有关更多详细信息，请参阅从 Aurora MySQL 版本 2 升级到版本 3。记下 Aurora MySQL 版本 2 和 3 之间的功能区别。

您可以在 Amazon Aurora 版本 中找到 Aurora 主要版本即将终止使用的日期。Amazon 会自动升级您在终止使用日期之前没有自行升级的所有集群。在终止使用日期之后，这些到后续主版本的自动升级将在集群的计划维护时段进行。

以下是升级即将终止使用的 Aurora MySQL 版本 1 集群（预置和 Aurora Serverless）的其他里程碑。对于每个里程碑，开始时间均为 00:00 通用协调时间 (UTC)。

1. 现在到 2023 年 2 月 28 日 - 您可以随时开始将 Aurora MySQL 版本 1（与 MySQL 5.6 兼容）集群升级到 Aurora MySQL 版本 2（与 MySQL 5.7 兼容）。从 Aurora MySQL 版本 2，您可以进一步升级到 Aurora MySQL 版本 3（与 MySQL 8.0 兼容）以支持 Aurora 预置数据库集群。
2. 2023 年 1 月 16 日 - 在此时间之后，您无法从 AWS Management Console 或 AWS Command Line Interface（AWS CLI）创建新的 Aurora MySQL 版本 1 集群或实例。您也无法向 Aurora Global Database 添加新的辅助区域。这可能会影响您从计划外停机中恢复的能力（如 从计划外停机中恢复 Amazon Aurora Global Database 中所述），因为在此时间之后您无法完成步骤 5 和 6。您还将无法新建运行 Aurora MySQL 版本 1 的跨区域只读副本。在 2023 年 2 月 28 日之前，您仍可以对现有的 Aurora MySQL 版本 1 集群执行以下操作：
   - 将为 Aurora MySQL 版本 1 集群制作的快照还原为与原始快照集群相同的版本。
   - 添加只读副本（不适用于 Aurora Serverless 数据库集群）。
   - 更改实例配置。
   - 执行时间点还原。
   - 创建现有版本 1 集群的克隆。
   - 新建运行 Aurora MySQL 版本 2 或更高版本的跨区域只读副本。
3. 2023 年 2 月 28 日 - 在此时间之后，我们计划在随后的计划维护时段内将 Aurora MySQL 版本 1 集群自动升级到默认版本的 Aurora MySQL 2 版本 2。还原 Aurora MySQL 版本 1 数据库快照会导致还原的集群当时自动升级到默认版本的 Aurora MySQL 版本 2。

与次要版本相比，在主要版本之间升级需要更广泛的计划和测试。这个过程可能需要大量时间。

对于首要任务是减少停机时间的情况，您也可以使用蓝绿部署在预调配的 Amazon Aurora 数据库集群中执行主要版本升级。蓝绿部署会创建一个复制生产环境的暂存环境。您可以在绿色（暂存）环境中更改 Aurora 数据库集群，而不会影响生产工作负载。切换通常需要不到一分钟，不会丢失数据，也无需更改应用程序。有关更多信息，请参阅 适用于 Aurora 的 Amazon RDS 蓝绿部署概述。

升级完成后，您可能还有后续工作要执行。例如，您可能需要跟进由于 SQL 兼容性、某些 MySQL 相关功能的工作方式或旧版本与新版本之间的参数设置导致的差异。

要了解有关 Aurora MySQL 主要版本升级的方法、规划、测试和故障排除的更多信息，请务必仔细阅读 升级 Amazon Aurora MySQL 数据库集群的主要版本。

查找受此终止使用流程影响的集群

要查找受此终止使用流程影响的群集，请使用以下过程。

> **Important**
>
> 请确保在资源所在的每个 AWS 区域中针对每个 AWS 账户按照这些说明操作。

控制台

查找 Aurora MySQL 版本 1 集群

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Databases（数据库）。
3. 在 Filter by databases（按数据库筛选）框中，输入 5.6。
4. 检查引擎列中的 Aurora MySQL。

AWS CLI

要查找受此终止使用流程影响的集群，请使用 AWS CLI，调用 describe-db-clusters 命令。您可以使用以下示例脚本。

Example

```
aws rds describe-db-clusters --include-share --query 'DBClusters[?Engine==`aurora`].
{EV:EngineVersion, DBCI:DBClusterIdentifier, EM:EngineMode}' --output table --region
 us-east-1

        +-----------------------------------------+
        |            DescribeDBClusters           |
        +---------------+--------------+----------+
        |     DBCI      |     EM       |    EV    |
        +---------------+--------------+----------+
        |  my-database-1|  serverless  |  5.6.10a |
        +---------------+--------------+----------+
```

RDS API

要查找运行 Aurora MySQL 版本 1 的 Aurora MySQL 数据库集群，请将 RDS DescribeDBClusters API 操作与以下所需参数结合使用：

- `DescribeDBClusters`
  - Filters.Filter.N
    - 名称
      - 引擎
    - Values.Value.N
      - ['aurora']

升级 Amazon Aurora MySQL 数据库集群

您可以升级 Aurora MySQL 数据库集群以获取错误修复、新的 Aurora MySQL 功能或对基础数据库引擎的全新版本的更改。以下各节介绍了操作方法。

> **Note**
>
> 您进行的升级类型取决于您可以为集群承受多少停机时间、计划进行多少验证测试、具体错误修复或新功能对您的使用案例的重要性，以及您是计划进行频繁的小型升级还是偶尔进行升级跳过几次中间版本。对于每次升级，您可以更改集群的主要版本、次要版本和补丁程序级别。如果您不熟悉 Aurora MySQL 主要版本、次要版本和补丁程序级别之间的区别，可以阅读检查 Aurora MySQL 版本号上的背景信息。

> **Tip**
>
> 您可以使用蓝绿部署，最大限度地减少数据库集群升级所需的停机时间。有关更多信息，请参阅 使用 Amazon Aurora 蓝绿部署进行数据库更新。

主题

- 升级 Aurora MySQL 数据库集群的次要版本或补丁程序级别

- 升级 Amazon Aurora MySQL 数据库集群的主要版本

升级 Aurora MySQL 数据库集群的次要版本或补丁程序级别

可以使用以下方法升级数据库集群的次要版本或修补数据库集群：

- 通过修改引擎版本升级 Aurora MySQL（适用于 Aurora MySQL 版本 2 和 3）
- 启用 Aurora MySQL 次要版本之间的自动升级

有关零停机时间修补如何减少升级过程中的中断的信息，请参阅使用零停机时间修补。

有关对 Aurora MySQL 数据库集群执行次要版本升级的信息，请参阅以下主题。

主题

- 在执行次要版本升级之前
- Aurora MySQL 的次要版本升级预检查
- 通过修改引擎版本升级 Aurora MySQL
- 启用 Aurora MySQL 次要版本之间的自动升级
- 使用零停机时间修补
- 备选的蓝绿升级技术

在执行次要版本升级之前

建议您执行以下操作以缩短次要版本升级期间的停机时间：

- 应在流量较低的时段执行 Aurora 数据库集群维护。使用 Performance Insights 来识别这些时间段，以便正确配置维护时段。有关 Performance Insights 的更多信息，请参阅在 Amazon RDS 上使用 Performance Insights 监控数据库负载。有关数据库集群维护时段的更多信息，请参阅调整首选数据库集群维护时段。
- 使用支持指数回退和抖动的 AWS SDK 作为最佳实践。有关更多信息，请参阅 Exponential Backoff And Jitter。

Aurora MySQL 的次要版本升级预检查

当您启动次要版本升级时，Amazon Aurora 会自动运行预检查。

这些预检查是必需的。您不能选择跳过它们。预检查提供以下好处：

- 它们让您可以在升级期间避免出现计划外停机。
- 如果存在不一致项，Amazon Aurora 将阻止升级并提供日志以供您参阅。然后，您可以使用日志来减少不兼容的情况，以准备数据库进行升级。有关消除不兼容情况的详细信息，请参阅 MySQL 文档中的 Preparing your installation for upgrade。

在为了升级而停止数据库实例之前先运行预检查，这意味着它们在运行时不会造成任何停机。如果预检查发现不兼容问题，Aurora 会在停止数据库实例之前自动取消升级。Aurora 还会针对不兼容问题生成事件。有关 Amazon Aurora 事件的更多信息，请参阅使用 Amazon RDS 事件通知。

Aurora 在日志文件 `PrePatchCompatibility.log` 中记录有关每项不兼容性的详细信息。在大部分情况下，日志条目包括用于纠正不兼容性的 MySQL 文档的链接。有关查看日志文件的更多信息，请参阅 查看和列出数据库日志文件。

由于预检查的性质，它们会分析数据库中的对象。此分析会导致资源消耗并增加完成升级的时间。

通过修改引擎版本升级 Aurora MySQL

升级 Aurora MySQL 数据库集群的次要版本会将其它修复和新功能应用于现有集群。

这种升级适用于原始版本和升级的版本都具有相同的 Aurora MySQL 主要版本（无论是 2 还是 3）的 Aurora MySQL 集群。这个过程快速而直接，因为它不涉及 Aurora MySQL 元数据的任何转换或表数据的重组。

您可以通过使用 AWS Management Console、AWS CLI 或 RDS API 修改数据库集群的引擎版本来执行这种升级。例如，如果您的集群正在运行 Aurora MySQL 3.x，请选择更高的 3.x 版本。

如果要对 Aurora Global Database 执行次要升级，请先升级所有辅助集群，然后再升级主集群。

> **Note**
>
> 要执行升级到 Aurora MySQL 版本 3.04.* 或更高版本或版本 2.12.* 的次要版本升级，请按照以下过程：
>
> 1. 从全局集群中删除所有辅助区域。按照从 Amazon Aurora Global Database 删除集群中的步骤操作。
> 2. 将主区域的引擎版本升级到 3.04.* 或更高版本或版本 2.12.*（如果适用）。按照To modify the engine version of a DB cluster中的步骤操作。

3. 向全局集群添加辅助区域。按照将 AWS 区域 添加到 Amazon Aurora Global Database中的步骤操作。

修改数据库集群的引擎版本

- 通过使用控制台 – 修改集群的属性。在 Modify DB cluster (修改数据库集群) 窗口中，更改 DB engine version (数据库引擎版本) 框中的 Aurora MySQL 引擎版本。如果您不熟悉修改集群的一般过程，请按照使用控制台、CLI 和 API 修改数据库集群中的说明操作。
- 通过使用 AWS CLI – 调用 modify-db-cluster AWS CLI 命令，为 `--db-cluster-identifier` 选项指定数据库集群的名称，并为 `--engine-version` 选项指定引擎版本。

  例如，要升级到 Aurora MySQL 版本 3.04.1，请将 `--engine-version` 选项设置为 `8.0.mysql_aurora.3.04.1`。指定 `--apply-immediately` 选项可立即更新数据库集群的引擎版本。
- 通过使用 RDS API – 调用 ModifyDBCluster API 操作，为 `DBClusterIdentifier` 参数指定数据库集群的名称，并为 `EngineVersion` 参数指定引擎版本。将 `ApplyImmediately` 参数设置为 `true` 可立即更新数据库集群的引擎版本。

启用 Aurora MySQL 次要版本之间的自动升级

对于 Amazon Aurora MySQL 数据库集群，您可以指定 Aurora 自动将数据库集群升级到新的次要版本。您可以通过设置数据库集群的 `AutoMinorVersionUpgrade` 属性（在AWS Management Console中启用自动次要版本升级）来实现此目的。

自动升级在维护时段发生。如果数据库集群中的各个数据库实例的维护时段与集群维护时段不同，则集群维护时段优先。

自动次要版本升级不适用于以下类型的 Aurora MySQL 集群：

- 属于 Aurora 全局数据库的集群
- 具有跨区域副本的集群

中断持续时间取决于工作负载、集群大小、二进制日志数据量以及 Aurora 是否可以使用零停机时间修补（ZDP）功能。Aurora 会重新启动数据库集群，因此您可能会在恢复使用集群之前经历短暂的不可用时间。特别是，二进制日志数据量会影响恢复时间。数据库实例在恢复期间处理二进制日志数据。因此，大量的二进制日志数据会增加恢复时间。

> **Note**
>
> 只有在数据库集群中的所有数据库实例中启用 AutoMinorVersionUpgrade 设置，Aurora 才会执行自动升级。有关如何设置该设置及其在集群和实例级别应用时的工作原理，请参阅 Aurora 数据库集群的自动次要版本升级。
> 然后，如果数据库集群的实例存在到 AutoUpgrade 已设置为 true 的次要数据库引擎版本的升级路径，则会进行升级。AutoUpgrade 设置是动态的，由 RDS 进行设置。
> 系统会对默认次要版本执行自动次要版本升级。

您可以使用如下 CLI 命令来检查 Aurora MySQL 集群中所有数据库实例的 AutoMinorVersionUpgrade 设置的状态。

```
aws rds describe-db-instances \
  --query '*[].
{DBClusterIdentifier:DBClusterIdentifier,DBInstanceIdentifier:DBInstanceIdentifier,AutoMinorVer
```

该命令产生的输出类似于以下内容：

```
[
  {
      "DBInstanceIdentifier": "db-t2-medium-instance",
      "DBClusterIdentifier": "cluster-57-2020-06-03-6411",
      "AutoMinorVersionUpgrade": true
  },
  {
      "DBInstanceIdentifier": "db-t2-small-original-size",
      "DBClusterIdentifier": "cluster-57-2020-06-03-6411",
      "AutoMinorVersionUpgrade": false
  },
  {
      "DBInstanceIdentifier": "instance-2020-05-01-2332",
      "DBClusterIdentifier": "cluster-57-2020-05-01-4615",
      "AutoMinorVersionUpgrade": true
  },
... output omitted ...
```

在此示例中，数据库集群 cluster-57-2020-06-03-6411 的允许自动次要版本升级处于关闭状态，因为对于集群中的其中一个数据库实例关闭了此功能。

使用零停机时间修补

对 Aurora MySQL 数据库集群执行升级时可能会在数据库关闭和升级期间发生中断。默认情况下，如果在数据库运行时开始升级，则会丢失数据库集群正在处理的所有连接和事务。如果等到数据库处于空闲状态再执行升级，则可能需要等待很长时间。

零停机时间修补 (ZDP) 功能尝试在 Aurora MySQL 升级期间尽力保留客户端连接。如果 ZDP 成功完成，则应用程序会话将保留，并且数据库引擎将在升级过程中重新启动。数据库引擎重新启动可能会导致吞吐量下降，持续时间从几秒钟到大约一分钟不等。

ZDP 不适用于以下情况：

- 操作系统（OS）补丁和升级
- 主要版本升级

ZDP 适用于所有支持的 Aurora MySQL 版本和数据库实例类。

Aurora Serverless v1 或 Aurora 全球数据库不支持 ZDP。

> **Note**
>
> 建议仅将 T 数据库实例类用于开发和测试服务器，或其他非生产服务器。有关 T 实例类的更多详细信息，请参阅使用 T 实例类进行开发和测试。

您可以在 MySQL 错误日志中查看 ZDP 期间重要属性的指标。您还可以在 AWS Management Console中的 Events（事件）页面上查看有关 Aurora MySQL 何时使用 ZDP、何时选择不使用 ZDP 的信息。

在 Aurora MySQL 中，Aurora 可以不停机执行修补，不管是否启用二进制日志复制。如果启用二进制日志复制，则在 ZDP 操作期间，Aurora MySQL 将自动断开与二进制日志目标的连接。Aurora MySQL 会自动重新连接到二进制日志目标，并在重启完成后恢复复制。

ZDP 还可与 Aurora MySQL 中的重启增强功能结合使用。修补写入器数据库实例会同时自动修补读取器。执行修补后，Aurora 将恢复写入器和读取器数据库实例上的连接。

在以下条件下，ZDP 可能无法成功完成：

- 正在执行长时间运行的查询或事务。如果 Aurora 在此情况下可以执行 ZDP，则将取消所有开放事务，但会保留其连接。
- 临时表、用户锁定或表锁定正在使用中，例如，在数据定义语言（DDL）语句运行期间。Aurora 会断开这些连接。

- 存在待处理的参数更改。

如果因以下一个或多个条件导致没有适合执行 ZDP 的时间范围，则修补将恢复为标准行为。

尽管成功执行 ZDP 操作后连接保持不变，但一些变量和功能会重新初始化。零停机修补重启后，以下类型的信息将不会保留：

- 全局变量。Aurora 将恢复会话变量，但重启后不会恢复全局变量。
- 状态变量。特别是，引擎状态报告的正常运行时间值会在使用 ZDR 或 ZDP 机制重启后重置。
- LAST_INSERT_ID.
- 表的内存中 auto_increment 状态。重新初始化内存中的自动增量状态。有关自动增量值的更多信息，请参阅 MySQL 参考手册。
- 来自 INFORMATION_SCHEMA 和 PERFORMANCE_SCHEMA 表的诊断信息。这些诊断信息也会显示在 SHOW PROFILE 和 SHOW PROFILES 等命令的输出中。

Events (事件) 页面上报告了以下与零停机时间重启相关的活动：

- 尝试在零停机的情况下升级数据库。
- 尝试在零停机的情况下升级数据库已完成。事件会报告该过程花费的时间。事件还会报告在重启期间保留的连接数量以及断开的连接数量。您可以查看数据库错误日志，了解有关重启期间所发生情况的更多详细信息。

备选的蓝绿升级技术

在某些情况下，您的首要任务是立即从旧集群切换到升级后的集群。在此类情况下，您可以使用多步骤流程，并排运行新旧集群。此处，您可以将数据从旧集群复制到新集群，直到您准备好接管新集群。有关详细信息，请参阅使用 Amazon Aurora 蓝绿部署进行数据库更新。

升级 Amazon Aurora MySQL 数据库集群的主要版本

在诸如 3.04.1 的 Aurora MySQL 版本号中，3 表示主要版本。Aurora MySQL 版本 2 与 MySQL 5.7 兼容。Aurora MySQL 版本 3 与 MySQL 8.0 兼容。

与次要版本相比，在主要版本之间升级需要更广泛的计划和测试。这个过程可能需要大量时间。升级完成后，您可能还有后续工作要执行。例如，出现这种情况可能是由于 SQL 兼容性或某些 MySQL 相关功能的工作方式不同。或者可能是由于旧版本和新版本之间的参数设置不同而导致的。

目录

- 从 Aurora MySQL 版本 2 升级到版本 3
- Aurora MySQL 主要版本升级路径
- Aurora MySQL 主要版本就地升级的工作原理
- 为 Aurora MySQL 集群计划主要版本升级
  - 通过克隆数据库集群来模拟升级
  - 蓝绿部署
- Aurora MySQL 的主要版本升级预检查
  - Aurora MySQL 的预检查过程
  - Aurora MySQL 的预检查日志格式
  - Aurora MySQL 的预检查日志输出示例
  - 预检查 Aurora MySQL 的性能
  - 社区 MySQL 升级预检查摘要
  - Aurora MySQL 升级预检查摘要
  - Aurora MySQL 的预检查描述参考
    - 错误
      - 报告错误的 MySQL 预检查
      - 报告错误的 Aurora MySQL 预检查
    - 警告
      - 报告警告的 MySQL 预检查
      - 报告警告的 Aurora MySQL 预检查
    - 版权声明
    - 错误、警告或通知
- 如何执行就地升级
  - 就地升级如何影响集群的参数组
  - Aurora MySQL 版本之间的集群属性更改
  - 全局数据库的就地主要版本升级
  - 回溯注意事项
- Aurora MySQL 就地升级教程
- 查找 Aurora MySQL 主要版本升级失败的原因

- Aurora MySQL 就地升级的故障排除
- Aurora MySQL 版本 3 的升级后清理
  - 空间索引

从 Aurora MySQL 版本 2 升级到版本 3

如果您有一个与 MySQL 5.7 兼容的集群，并希望将其升级到与 MySQL 8.0 兼容的集群，可以通过在集群本身上运行升级过程来实现。这种升级是就地升级，与您通过创建新集群进行的升级形成鲜明对比。此技术会保留集群的相同终端节点和其他特征。升级速度相对较快，因为它不需要将所有数据复制到新的集群卷。此稳定性有助于显著减少应用程序中的任何配置更改。它还有助于减少升级后集群的测试量。这是因为数据库实例的数量及其实例类都保持不变。

就地升级机制会在进行操作的过程中关闭您的数据库集群。Aurora 将执行正常关机，并完成进行中的操作（例如事务回滚和撤消清除）。有关更多信息，请参阅 Aurora MySQL 主要版本就地升级的工作原理。

就地升级方法非常方便，因为它执行过程简单，并且可以显著减少对关联应用程序的配置更改。例如，就地升级会为集群保留终端节点和数据库实例集。但是，就地升级所需的时间可能会有所不同，具体取决于架构的属性和集群的繁忙程度。这样，根据集群的需求，您可以在多种升级技术间进行选择：

- 就地升级
- 蓝绿部署
- 快照还原

> **Note**
>
> 如果您使用 AWS CLI 或 RDS API 作为快照还原升级方法，则必须运行后续操作才能在还原的数据库集群中创建写入器数据库实例。

有关 Aurora MySQL 版本 3 及其新功能的一般信息，请参阅 与 MySQL 8.0 兼容的 Aurora MySQL 版本 3。

有关计划升级的详细信息，请参阅 为 Aurora MySQL 集群计划主要版本升级 和 如何执行就地升级。

Aurora MySQL 主要版本升级路径

并非所有类型或版本的 Aurora MySQL 集群都可以使用就地升级机制。通过查阅下表，您可以了解每个 Aurora MySQL 集群的适当升级路径。

Aurora MySQL 数据库集群类型	它可以使用就地升级吗？	操作
Aurora MySQL 预置集群，版本 2	是	与 MySQL 5.7 兼容的 Aurora MySQL 集群支持就地升级。 有关升级到 Aurora MySQL 版本 3 的信息，请参阅 为 Aurora MySQL 集群计划主要版本升级 和 如何执行就地升级。
Aurora MySQL 预置集群，版本 3	不适用	使用次要版本升级过程在 Aurora MySQL 版本 3 的各版本之间进行升级。
Aurora Serverless v1 集群	不适用	只有 Aurora MySQL 版本 2 才支持 Aurora Serverless v1。
Aurora Serverless v2 集群	不适用	只有 Aurora MySQL 版本 3 才支持 Aurora Serverless v2。
Aurora Global Database 中的集群	是	要将 Aurora MySQL 从版本 2 升级到版本 3，请按照对 Aurora Global Database 中的集群进行就地升级的过程操作。在全局集群上执行升级。Aurora 会同时升级全局数据库中的主集群以及所有辅助集群。 如果您使用 AWS CLI 或 RDS API，请调用 `modify-global-cluster` 命令或 `ModifyGlobalCluster` 操作，而不是 `modify-db-cluster` 或 `ModifyDBCluster` 。 只有当您将 `lower_case_table_names` 参数设置为默认值并且重启全球数据库之后，才能执行从 Aurora MySQL 版本 2 到版本 3 的就地升级。有关更多信息，请参阅 主要版本升级。
并行查询集群	是	您可以执行就地升级。

Aurora MySQL 数据库集群类型	它可以使用就地升级吗？	操作
二进制日志复制的目标集群	也许	如果二进制日志复制来自 Aurora MySQL 集群，您可以执行就地升级。如果二进制日志复制来自 RDS for MySQL 或本地 MySQL 数据库实例，则无法执行升级。在这种情况下，您可以使用快照还原机制进行升级。
具有零数据库实例的集群	否	使用 AWS CLI 或 RDS API，您可以在没有任何附加的数据库实例的情况下创建 Aurora MySQL 集群。同样，您还可以从 Aurora MySQL 集群中删除所有数据库实例，同时保持集群卷中的数据不变。虽然集群的数据库实例为零，但您无法执行就地升级。 升级机制要求集群中的写入器实例对系统表、数据文件等执行转换。在这种情况下，请使用 AWS CLI 或 RDS API 为集群创建写入器实例。然后你可以执行就地升级。
启用了回溯的集群	是	可以对使用回溯功能的 Aurora MySQL 集群执行就地升级。但是，升级后，您无法将集群回溯到升级之前的时间。

Aurora MySQL 主要版本就地升级的工作原理

Aurora MySQL 将主要版本升级作为多节点过程执行。您可以查看升级的当前状态。有些升级步骤还提供进度信息。每个阶段开始时，Aurora MySQL 会记录一个事件。您可以在 RDS 控制台的 Events（事件）页面上检查事件发生的事件。有关使用事件的更多信息，请参阅使用 Amazon RDS 事件通知。

> **Important**
>
> 一旦该过程开始，它就会运行，直到升级成功或失败。您不能在升级进行中取消升级。如果升级失败，Aurora 将回滚所有更改，并且集群具有与以前相同的引擎版本、元数据等。

升级过程包括三个阶段：

1. Aurora 在开始升级过程之前执行一系列预检查。在 Aurora 进行这些检查的同时，您的集群会保持运行。例如，集群不能有任何处于准备状态的 XA 事务，也不能处理任何数据定义语言 (DDL) 语句。例如，您可能需要关闭提交某些类型的 SQL 语句的应用程序。或者，您只需等待某些长期运行的语句完成。然后再次尝试升级。有些检查会测试不会阻止升级但可能使升级花费很长时间的条件。

   如果 Aurora 检测到任何必需条件未满足，请修改事件详细信息中识别的条件。按照Aurora MySQL 就地升级的故障排除中的指导进行操作。如果 Aurora 检测到可能导致升级缓慢的情况，请计划在一段较长的时间内监控升级。

2. Aurora 使您的集群离线。然后，Aurora 执行与上一阶段类似的一组测试，以确认关机过程中没有产生新问题。如果此时 Aurora 检测到任何会阻止升级的情况，Aurora 会取消升级并使集群恢复联机。在这种情况下，请确认条件何时不再适用，然后再次开始升级。

3. Aurora 创建集群卷的快照。假设您在升级完成后兼容性或其他类型的问题。或者假设您想同时使用原始集群和升级后的集群执行测试。在这种情况下，您可以从此快照中还原，以使用原始引擎版本和原始数据创建新集群。

   > **Tip**
   >
   > 此快照是手动快照。但是，即使您已经达到手动快照的配额，Aurora 也可以创建手动快照并继续升级过程。此快照将永久保留（如果需要），直到您将其删除。完成所有升级后测试后，您可以删除此快照以显著减少存储费用。

4. Aurora 克隆集群卷。克隆是一项快速的操作，不涉及复制实际的表数据。如果 Aurora 在升级过程中遇到问题，它将从克隆的集群卷恢复为原始数据，然后使集群重新联机。升级期间的临时克隆卷不受单个集群卷的克隆数量的通常限制。

5. Aurora 对写入器数据库实例执行清理关机。在清理关机期间，以下操作的进度事件每 15 分钟记录一次。您可以在 RDS 控制台的 Events（事件）页面上检查事件发生的事件。

   - Aurora 清除旧版本行的撤销记录。
   - Aurora 回滚任何未提交的事务。

6. Aurora 升级写入器数据库实例上的引擎版本：

   - Aurora 在写入器数据库实例上安装新引擎版本的二进制文件。
   - Aurora 使用写入器数据库实例将数据升级为兼容 MySQL 5.7 的格式。在此阶段，Aurora 修改系统表并执行影响集群卷中的数据的其他转换。特别是，Aurora 升级系统表中的分区元数据以与 MySQL 5.7 分区格式兼容。如果集群中的表有大量分区，则此阶段可能需要花费很长时间。

如果在此阶段发生任何错误，您可以在 MySQL 错误日志中找到详细信息。此阶段开始后，如果升级过程因任何原因失败，Aurora 将从克隆的集群卷中恢复原始数据。

7. Aurora 升级读取器数据库实例上的引擎版本。
8. 升级过程已完成。Aurora 记录最后一个事件以表示升级过程已成功完成。现在，您的数据库集群正在运行新的主要版本。

为 Aurora MySQL 集群计划主要版本升级

为了帮助您决定正确的升级时间和方法，您可以了解 Aurora MySQL 版本 3 与当前环境之间的区别：

- 如果您要从 RDS for MySQL 8.0 或 MySQL 8.0 社区版进行转换，请参阅比较 Aurora MySQL 版本 3 和 MySQL 8.0 社群版。
- 如果您要从 Aurora MySQL 版本 2、RDS for MySQL 5.7 或社区 MySQL 5.7 升级，请参阅比较 Aurora MySQL 版本 2 和 Aurora MySQL 版本 3。
- 为任何自定义参数组创建新的 MySQL 8.0 兼容版本。将所有必要的自定义参数值应用于新参数组。咨询 Aurora MySQL 版本 3 的参数更改 以了解参数更改。
- 请查看您的 Aurora MySQL 版本 2 数据库架构和对象定义，以了解 MySQL 8.0 社区版中引入的新保留关键字的使用情况。请在升级之前执行此操作。有关更多信息，请参阅 MySQL 文档中的 MySQL 8.0 新关键字和保留关键字。

您还可以在 MySQL 参考手册的 MySQL 8.0 中的变化中找到更多特定于 MySQL 的升级注意事项和提示。例如，您可以使用命令 `mysqlcheck --check-upgrade` 来分析现有的 Aurora MySQL 数据库并确定潜在的升级问题。

> **Note**
>
> 在使用就地升级或快照还原技术升级到 Aurora MySQL 版本 3 时，我们建议使用更大的数据库实例类。例如，db.r5.24xlarge 和 db.r6g.16xlarge。这样，通过使用数据库实例上的绝大部分可用 CPU 容量，有助于更快地完成升级过程。在主要版本升级完成后，您可以更改为所需的数据库实例类。

完成升级后，您可以按照 Aurora MySQL 版本 3 的升级后清理 中的升级后步骤进行操作。最后，测试应用程序的功能和性能。

如果您要从 MySQL 或社区 MySQL 进行 RDS 转换，请按照将数据迁移到 Amazon Aurora MySQL 数据库集群中所述的迁移过程进行操作。在某些情况下，作为迁移的一部分，您可以使用二进制日志复制将数据与 Aurora MySQL 版本 3 集群同步数据。如果是这样，源系统必须运行与目标数据库集群兼容的版本。

为了确保在主要版本之间升级集群后，您的应用程序和管理过程能够顺利运行，应进行一些预先计划和准备。要查看可为您的 AWS CLI 脚本或基于 RDS API 的应用程序更新哪些类型的管理代码，请参阅就地升级如何影响集群的参数组。另请参阅Aurora MySQL 版本之间的集群属性更改。

要了解在升级期间可能遇到的问题，请参阅Aurora MySQL 就地升级的故障排除。对于可能导致升级需要很长时间的问题，您可以提前测试这些条件并进行更正。

> **Note**
>
> 就地升级会在进行操作的过程中关闭您的数据库集群。Aurora MySQL 将执行正常关机，并完成进行中的操作（例如撤消清除）。如果要清除的撤消记录很多，则升级可能需要很长时间。我们建议仅在历史记录列表长度（HLL）较小之后才执行升级。HLL 的一般可接受值为 100000 或更小。有关更多信息，请参阅此博客文章。

通过克隆数据库集群来模拟升级

您可以检查升级后集群的应用程序兼容性、性能、维护过程以及类似注意事项。为此，您可以在真正进行升级之前执行升级模拟。此技术对生产集群尤其有用。在这里，重要的是要尽量减少停机时间，并在升级完成后立即让升级后的集群准备就绪。

使用以下步骤：

1. 创建原始集群的克隆。按照克隆 Amazon Aurora 数据库集群卷中过程操作。
2. 设置与原始集群中类似的写入器和读取器数据库实例集。
3. 对克隆的集群执行就地升级。按照如何执行就地升级中过程操作。

   创建克隆后立即开始升级。这样，集群卷仍然与原始集群的状态相同。如果克隆在升级之前处于空闲状态，则 Aurora 会在后台执行数据库清理过程。在这种情况下，克隆的升级并不是原始集群升级的准确模拟。
4. 使用克隆的集群测试应用程序兼容性、性能、管理过程等。
5. 如果遇到任何问题，请调整升级计划以解决这些问题。例如，调整任何应用程序代码以与更高版本的功能集兼容。根据集群中的数据量估计升级可能需要多长时间。您也可以选择将升级安排在集群不忙的时候。

6. 在您对应用程序和工作负载在测试集群中正常运行感到满意后，您可以为生产集群执行就地升级。
7. 努力显著减少集群在主要版本升级期间的总停机时间。为此，请确保升级时集群上的工作负载较低或为零。特别是确保在启动升级时没有长期运行的事务在进行。

蓝绿部署

在某些情况下，您的首要任务是立即从旧集群切换到升级后的集群。在此类情况下，您可以使用多步骤流程，并排运行新旧集群。此处，您可以将数据从旧集群复制到新集群，直到您准备好接管新集群。有关详细信息，请参阅使用 Amazon Aurora 蓝绿部署进行数据库更新。

Aurora MySQL 的主要版本升级预检查

将 MySQL 从一个主要版本升级到另一个主要版本，例如从 MySQL 5.7 升级到 MySQL 8.0，需要进行一些重大的架构更改，这要求进行仔细的规划和准备。与次要版本升级（重点主要是更新数据库引擎软件，在某些情况下还包括系统表）不同，主要 MySQL 升级通常会对数据库存储和管理其元数据的方式进行根本性的更改。

为协助您确定此类不兼容性，当从 Aurora MySQL 版本 2 升级到版本 3 时，Aurora 会自动运行升级兼容性检查（预检查），来检查数据库集群中的对象并确定可能阻碍升级继续进行的已知不兼容性。有关 Aurora MySQL 预检查的详细信息，请参阅 Aurora MySQL 的预检查描述参考。除了社区 MySQL upgrade checker utility 运行的预检查之外，还会运行 Aurora 预检查。

这些预检查是必需的。您不能选择跳过它们。预检查提供以下好处：

- 它们可以减少遇到升级失败的可能性，而升级失败可能导致停机时间延长。
- 如果存在不一致性，Amazon Aurora 将阻止继续升级，并提供日志以供您参阅。然后，您可以使用日志，通过解决不一致性来准备数据库以升级到版本 3。有关解决不兼容性的详细信息，请参阅 MySQL 文档中的 Preparing your installation for upgrade 和 MySQL Server 博客上的 Upgrading to MySQL 8.0? Here is what you need to know...。

  有关升级到 MySQL 8.0 的更多信息，请参阅 MySQL 文档中的升级 MySQL。

预检查在数据库集群变为脱机状态来进行主版本升级之前运行。如果预检查发现不兼容问题，Aurora 会在停止数据库实例之前自动取消升级。Aurora 还会针对不兼容问题生成事件。有关 Amazon Aurora 事件的更多信息，请参阅使用 Amazon RDS 事件通知。

在完成预检查后，Aurora 在 `upgrade-prechecks.log` 文件中记录有关每项不兼容性的详细信息。在大部分情况下，日志条目包括用于纠正不兼容性的 MySQL 文档的链接。有关查看日志文件的更多信息，请参阅 查看和列出数据库日志文件。

> **Note**
>
> 由于预检查的性质，它们会分析数据库中的对象。此分析会导致资源消耗并增加完成升级的时间。有关预检查性能注意事项的更多信息，请参阅 Aurora MySQL 的预检查过程。

目录

- Aurora MySQL 的预检查过程
- Aurora MySQL 的预检查日志格式

- Aurora MySQL 的预检查日志输出示例
- 预检查 Aurora MySQL 的性能
- 社区 MySQL 升级预检查摘要
- Aurora MySQL 升级预检查摘要
- Aurora MySQL 的预检查描述参考
  - 错误
    - 报告错误的 MySQL 预检查
    - 报告错误的 Aurora MySQL 预检查
  - 警告
    - 报告警告的 MySQL 预检查
    - 报告警告的 Aurora MySQL 预检查
  - 版权声明
  - 错误、警告或通知

Aurora MySQL 的预检查过程

如前所述，Aurora MySQL 升级过程涉及到在进行主要版本升级之前对数据库运行兼容性检查（预检查）。

对于就地升级，预检查会在写入器数据库实例处于联机状态时对该实例运行。如果预检查成功，升级将继续。如果发现错误，则会将错误记录在 `upgrade-prechecks.log` 文件中并取消升级。在再次尝试升级之前，请先解决在 `upgrade-prechecks.log` 文件中返回的所有错误。

对于快照还原升级，预检查将在还原过程中运行。如果成功，数据库将升级到新的 Aurora MySQL 版本。如果发现错误，则会将错误记录在 `upgrade-prechecks.log` 文件中并取消升级。在再次尝试升级之前，请先解决在 `upgrade-prechecks.log` 文件中返回的所有错误。

有关更多信息，请参阅查找 Aurora MySQL 主要版本升级失败的原因 和Aurora MySQL 的预检查描述参考。

要监控预检查状态，可以查看数据库集群上的以下事件。

预检查状态	事件消息	操作
Started	正在进行升级准备：正在启动在线升级预检查。	无

预检查状态	事件消息	操作
失败	数据库集群处于无法升级的状态：升级预检查失败。有关更多详细信息，请参阅 upgrade-prechecks.log 文件。 有关排查升级失败原因的更多信息，请参阅 https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/AuroraMySQL.Upgrading.Troubleshooting.html	查看 upgrade-prechecks.log 是否有错误。 纠正错误。 重试升级。
成功	正在进行升级准备：已完成在线升级预检查。	预检查成功，未返回任何错误。 查看 upgrade-prechecks.log 是否有警告和通知。

有关查看事件的更多信息，请参阅查看 Amazon RDS 事件。

Aurora MySQL 的预检查日志格式

升级兼容性检查（预检查）完成后，可以查看 upgrade-prechecks.log 文件。日志文件包含每次预检查的结果、受影响对象和纠正信息。

错误会阻止升级。在重试升级之前，必须先解决这些问题。

警告和通知不那么重要，但我们仍建议您仔细查看它们，以确保应用程序工作负载不存在兼容性问题。尽快解决所有已发现的问题。

日志文件采用以下格式：

- `targetVersion`：Aurora MySQL 升级的 MySQL 兼容版本。
- `auroraServerVersion`：对其运行预检查的 Aurora MySQL 版本。
- `auroraTargetVersion`：要升级到的 Aurora MySQL 版本。

- checksPerformed：包含已执行的预检查列表。
- id：正在运行的预检查的名称。
- title：对正在运行的预检查的描述。
- status：这并不指示预检查成功还是失败，但显示预检查查询的状态：
  - OK：预检查查询运行并成功完成。
  - ERROR：预检查查询运行失败。出现这种情况的原因可能是资源限制、实例意外重新启动或兼容性预检查查询被中断等问题。

    有关更多信息，请参阅此示例。
- description：不兼容性的一般描述，以及如何纠正该问题。
- documentationLink：如果适用，此处会注明指向相关 Aurora MySQL 或 MySQL 文档的链接。有关更多信息，请参阅 Aurora MySQL 的预检查描述参考。
- detectedProblems：如果预检查返回错误、警告或通知，这会显示不兼容性的详细信息以及不兼容的对象（如果适用）：
  - level：预检查检测到的不兼容性级别。有效级别如下所示：
    - Error：在您解决不兼容性之前，升级无法继续。
    - Warning：升级可以继续，但检测到已弃用的对象、语法或配置。请仔细查看警告，并尽快予以解决，以免在将来的版本中出现问题。
    - Notice：升级可以继续，但检测到已弃用的对象、语法或配置。请仔细查看通知，并尽快予以解决，以免在将来的版本中出现问题。
  - dbObject：在其中检测出不兼容性的数据库对象的名称。
  - description：有关不兼容性的详细描述，以及如何纠正该问题。
- errorCount：检测到的不兼容性错误的数量。这些错误会阻止升级。
- warningCount：检测到的不兼容性警告的数量。这些警告不会阻止升级，但应尽快予以解决，以免在将来的版本中出现问题。
- noticeCount：检测到的不兼容通知的数量。这些通知不会阻止升级，但应尽快予以解决，以免在将来的版本中出现问题。
- Summary：预检查兼容性错误、警告和通知计数的摘要。

Aurora MySQL 的预检查日志输出示例

下面的示例显示了您可能看到的预检查日志输出。有关运行的预检查的详细信息，请参阅 Aurora MySQL 的预检查描述参考。

预检查状态为 OK，未检测到不兼容性

预检查查询成功完成。未检测到任何不兼容性。

```
{
  "id": "auroraUpgradeCheckIndexLengthLimitOnTinytext",
  "title": "Check for the tables with indexes defined with prefix length greater
 than 255 bytes on tiny text columns",
  "status": "OK",
  "detectedProblems": []
},
```

预检查状态为 OK，检测到错误

预检查查询成功完成。检测到一个错误。

```
{
  "id": "auroraUpgradeCheckForPrefixIndexOnGeometryColumns",
  "title": "Check for geometry columns on prefix indexes",
  "status": "OK",
  "description": "Consider dropping the prefix indexes of geometry columns and
 restart the upgrade.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test25.sbtest1",
        "description": "Table `test25`.`sbtest1` has an index `idx_t1` on geometry
 column/s. Mysql 8.0 does not support this type of index on a geometry column
 https://dev.mysql.com/worklog/task/?id=11808. To upgrade to MySQL 8.0, Run 'DROP
 INDEX `idx_t1` ON `test25`.`sbtest1`;"
      },
 }
```

预检查状态为 OK，检测到警告

当预检查成功或失败时，可能会返回警告。

此处预检查查询成功完成。检测到两个警告。

```
{
  "id": "zeroDatesCheck",
  "title": "Zero Date, Datetime, and Timestamp values",
```

```
  "status": "OK",
  "description": "Warning: By default zero date/datetime/timestamp values are no
 longer allowed in MySQL, as of 5.7.8 NO_ZERO_IN_DATE and NO_ZERO_DATE are included
 in SQL_MODE by default. These modes should be used with strict mode as they will be
 merged with strict mode in a future release. If you do not include these modes in
 your SQL_MODE setting, you are able to insert date/datetime/timestamp values that
 contain zeros. It is strongly advised to replace zero values with valid ones, as
 they may not work correctly in the future.",
  "documentationLink": "https://lefred.be/content/mysql-8-0-and-wrong-dates/",
  "detectedProblems": [
      {
        "level": "Warning",
        "dbObject": "global.sql_mode",
        "description": "does not contain either NO_ZERO_DATE or NO_ZERO_IN_DATE
 which allows insertion of zero dates"
      },
      {
        "level": "Warning",
        "dbObject": "session.sql_mode",
        "description": " of 10 session(s) does not contain either NO_ZERO_DATE or
 NO_ZERO_IN_DATE which allows insertion of zero dates"
      }
    ]
}
```

预检查状态为 ERROR，未报告任何不兼容性

预检查查询失败，出现一个错误，因此无法验证不兼容性。

```
{
  "id": "auroraUpgradeCheckForDatafilePathInconsistency",
  "title": "Check for inconsistency related to ibd file path.",
  "status": "ERROR",
  "description": "Can't connect to MySQL server on 'localhost:3306' (111) at
 13/08/2024 12:22:20 UTC. This failure can occur due to low memory available on the
 instance for executing upgrade prechecks. Please check 'FreeableMemory' Cloudwatch
 metric to verify the available memory on the instance while executing prechecks. If
 instance ran out of memory, we recommend to retry the upgrade on a higher instance
 class."
}
```

这一失败的原因可能是实例意外重新启动，或数据库上正在运行的兼容性预检查查询被中断。例如，在较小的数据库实例类上，当实例上的可用内存不足时，可能会遇到这种情况。

在运行预检查时，可以使用 FreeableMemory Amazon CloudWatch 指标来验证实例上的可用内存。如果实例内存不足，建议在更大的数据库实例类上重试升级。在某些情况下，可以使用蓝绿部署。这支持在独立于生产工作负载的“绿色”数据库集群上运行预检查和升级，但这也会消耗系统资源。

有关更多信息，请参阅排查 Aurora MySQL 数据库的内存使用问题。

预检查摘要，检测到一个错误和三个警告

兼容性预检查还包含有关源 Aurora MySQL 版本和目标 Aurora MySQL 版本的信息，以及位于预检查输出末尾的错误、警告和通知计数摘要。

例如，以下输出显示有人尝试从 Aurora MySQL 2.11.6 升级到 Aurora MySQL 3.07.1。此升级返回了一个错误、三个警告，但没有返回任何通知。由于返回错误时无法继续升级，因此必须解决 routineSyntaxCheck 兼容性问题并重试升级。

```
{
  "serverAddress": "/tmp%2Fmysql.sock",
  "serverVersion": "5.7.12 - MySQL Community Server (GPL)",
  "targetVersion": "8.0.36",
  "auroraServerVersion": "2.11.6",
  "auroraTargetVersion": "3.07.1",
  "outfilePath": "/rdsdbdata/tmp/PreChecker.log",
  "checksPerformed": [{
      ... output for each individual precheck ...
      .
      .
      {
        "id": "oldTemporalCheck",
        "title": "Usage of old temporal type",
        "status": "OK",
          "detectedProblems": []
      },
      {
        "id": "routinesSyntaxCheck",
        "title": "MySQL 8.0 syntax check for routine-like objects",
        "status": "OK",
        "description": "The following objects did not pass a syntax check with
 the latest MySQL 8.0 grammar. A common reason is that they reference names that
 conflict with new reserved keywords. You must update these routine definitions and
 `quote` any such references before upgrading.",
        "documentationLink": "https://dev.mysql.com/doc/refman/en/keywords.html",
        "detectedProblems": [{
```

```
            "level": "Error",
            "dbObject": "test.select_res_word",
            "description": "at line 2,18: unexpected token 'except'"
        }]
      },
      .
      .
      .
      {
        "id": "zeroDatesCheck",
        "title": "Zero Date, Datetime, and Timestamp values",
        "status": "OK",
        "description": "Warning: By default zero date/datetime/timestamp values
 are no longer allowed in MySQL, as of 5.7.8 NO_ZERO_IN_DATE and NO_ZERO_DATE are
 included in SQL_MODE by default. These modes should be used with strict mode as
 they will be merged with strict mode in a future release. If you do not include
 these modes in your SQL_MODE setting, you are able to insert date/datetime/
timestamp values that contain zeros. It is strongly advised to replace zero values
 with valid ones, as they may not work correctly in the future.",
        "documentationLink": "https://lefred.be/content/mysql-8-0-and-wrong-dates/",
        "detectedProblems": [{
            "level": "Warning",
            "dbObject": "global.sql_mode",
            "description": "does not contain either NO_ZERO_DATE or NO_ZERO_IN_DATE
 which allows insertion of zero dates"
            },
            {
            "level": "Warning",
            "dbObject": "session.sql_mode",
            "description": " of 8 session(s) does not contain either NO_ZERO_DATE or
 NO_ZERO_IN_DATE which allows insertion of zero dates"
            }
          ]
       },
       .
       .
       .
  }],
  "errorCount": 1,
  "warningCount": 3,
  "noticeCount": 0,
  "Summary": "1 errors were found. Please correct these issues before upgrading to
 avoid compatibility issues."
```

```
}
```

预检查 Aurora MySQL 的性能

兼容性预检查在数据库实例进入脱机状态来进行升级之前运行，因此在正常情况下，它们不会在运行时导致数据库实例停机。但是，它们可能会影响写入器数据库实例上运行的应用程序工作负载。预检查通过 information_schema 表访问数据字典，如果数据库对象很多，这可能会很慢。请考虑以下因素：

- 预检查持续时间因数据库对象（例如表、列、例程和约束）的数量而异。包含大量对象的数据库集群可能需要更长的时间来运行预检查。

  例如，removedFunctionsCheck 可能需要更长的时间，并且会根据 stored objects 的数量使用更多的资源。
- 对于就地升级，使用更大的数据库实例类（例如 db.r5.24xlarge 或 db.r6g.16xlarge），有助于通过使用更多 CPU 来更快地完成升级。可以在升级后缩小规模。
- 跨多个数据库对 `information_schema` 进行的查询可能会很慢，尤其是对于许多对象和较小的数据库实例。在这类情况下，可以考虑使用克隆、快照还原或蓝绿部署来进行升级。
- 预检查资源使用量（CPU、内存）会随着对象增加而增加，从而导致较小数据库实例上的运行时间更长。在这类情况下，可以考虑使用克隆、快照还原或蓝绿部署来进行升级测试。

  如果由于资源不足而导致预检查失败，则可以使用状态输出在预检查日志中检测到此问题：

```
"status": "ERROR",
```

有关更多信息，请参阅Aurora MySQL 主要版本就地升级的工作原理 和为 Aurora MySQL 集群计划主要版本升级。

社区 MySQL 升级预检查摘要

以下是 MySQL 5.7 和 8.0 之间不兼容性的一般列表：

- 与 MySQL 5.7 兼容的数据库集群不得使用 MySQL 8.0 不支持的功能。

  有关更多信息，请参阅 MySQL 文档中的 MySQL 8.0 中删除的功能。
- 不得出现关键字或保留关键字违规情况。MySQL 8.0 中可能会保留一些以前未保留的关键字。

  有关更多信息，请参阅 MySQL 文档中的关键字和保留关键字。

- 要获得改进的 Unicode 支持，请考虑将使用 utf8mb3 字符集的对象转换为使用 utf8mb4 字符集。utf8mb3 字符集已弃用。此外，请考虑对字符集引用使用 utf8mb4 而不是 utf8，因为 utf8 当前是 utf8mb3 字符集的别名。

  有关更多信息，请参阅 MySQL 文档中的 utf8mb3 字符集（3 字节 UTF-8 Unicode 编码）。
- 不得有非默认行格式的 InnoDB 表。
- 不得有 ZEROFILL 或 display 长度类型属性。
- 不得有使用不支持本机分区的存储引擎的分区表。
- MySQL 5.7 mysql 系统数据库中不得有与 MySQL 8.0 数据字典使用的表同名的表。
- 不得有使用过时的数据类型或函数的表。
- 不得有超过 64 个字符的外键约束名称。
- sql_mode 系统变量设置中不得定义过时的 SQL 模式。
- 不得有包含长度超过 255 个字符的单个 ENUM 或 SET 列元素的表或存储过程。
- 不得有驻留在共享 InnoDB 表空间中的表分区。
- 表空间数据文件路径中不得有循环引用。
- 不得有对 GROUP BY 子句使用 ASC 或 DESC 限定符的查询和存储程序定义。
- 不得有任何已移除的系统变量，并且系统变量必须使用适用于 MySQL 8.0 的新默认值。
- 不得有零（0）日期、日期时间或时间戳值。
- 不得有由于文件移除或损坏而导致的架构不一致。
- 不得有包含 FTS 字符串的表名称。
- 不得有属于不同引擎的 InnoDB 表。
- 不得有对于 MySQL 5.7 无效的表或架构名称。

有关运行的预检查的详细信息，请参阅 Aurora MySQL 的预检查描述参考。

有关升级到 MySQL 8.0 的更多信息，请参阅 MySQL 文档中的升级 MySQL。有关 MySQL 8.0 中变化的一般描述，请参阅 MySQL 文档中的 What is new in MySQL 8.0。

Aurora MySQL 升级预检查摘要

Aurora MySQL 在从版本 2 升级到版本 3 时有自己的特定要求，包括以下各项：

- 视图、例程、触发器和事件中不得有弃用的 SQL 语法，例如 SQL_CACHE、SQL_NO_CACHE 和 QUERY_CACHE。

- 没有 FTS 索引的任何表上都不得有 FTS_DOC_ID 列。
- InnoDB 数据字典和实际表定义之间不得有列定义不匹配。
- 将 lower_case_table_names 参数设置为 1 时，所有数据库和表名都必须为小写。
- 事件和触发器不得有缺失的或空的定义程序或无效的创建上下文。
- 数据库中的所有触发器名称都必须是唯一的。
- Aurora MySQL 版本 3 不支持 DDL 恢复和快速 DDL。数据库中不得有与这些功能相关的构件。
- 采用 REDUNDANT 或 COMPACT 行格式的表的索引不能大于 767 字节。
- 在 tiny 文本列上定义的索引的前缀长度不能超过 255 字节。对于 utf8mb4 字符集，这会将支持的前缀长度限制为 63 个字符。

  在 MySQL 5.7 中，使用 innodb_large_prefix 参数可支持更大的前缀长度。MySQL 8.0 中已弃用此参数。
- mysql.host 表中不得有 InnoDB 元数据不一致。
- 系统表中不得有列数据类型不匹配。
- 不得有 XA 事务处于 prepared 状态。
- 视图中的列名称不能超过 64 个字符。
- 存储过程中的特殊字符不能不一致。
- 表不能有数据文件路径不一致。

有关运行的预检查的详细信息，请参阅 Aurora MySQL 的预检查描述参考。

Aurora MySQL 的预检查描述参考

此处详细描述了 Aurora MySQL 的升级预检查。

目录

- 错误
  - 报告错误的 MySQL 预检查
  - 报告错误的 Aurora MySQL 预检查
- 警告
  - 报告警告的 MySQL 预检查
  - 报告警告的 Aurora MySQL 预检查
- 版权声明

- 错误、警告或通知

错误

以下预检查失败时会生成错误，升级无法继续。

主题

- 报告错误的 MySQL 预检查
- 报告错误的 Aurora MySQL 预检查

报告错误的 MySQL 预检查

以下预检查来自社区 MySQL：

- checkTableMysqlSchema
- circularDirectoryCheck
- columnsWhichCannotHaveDefaultsCheck
- depreciatedSyntaxCheck
- engineMixupCheck
- enumSetElementLengthCheck
- foreignKeyLengthCheck
- getDuplicateTriggers
- getEventsWithNullDefiner
- getMismatchedMetadata
- getTriggersWithNullDefiner
- getValueOfVariablelower_case_table_names
- groupByAscSyntaxCheck
- mysqlEmptyDotTableSyntaxCheck
- mysqlIndexTooLargeCheck
- mysqlInvalid57NamesCheck
- mysqlOrphanedRoutinesCheck
- mysqlSchemaCheck

- nonNativePartitioningCheck
- oldTemporalCheck
- partitionedTablesInSharedTablespaceCheck
- removedFunctionsCheck
- routineSyntaxCheck
- schemaInconsistencyCheck

checkTableMysqlSchema

预检查级别：错误

`check table x for upgrade` 命令针对 `mysql` 架构报告的问题

在开始升级到 Aurora MySQL 版本 3 之前，将对数据库实例的 `mysql` 架构中的每个表运行 `check table for upgrade`。`check table for upgrade` 命令检查表中是否存在升级到较新 MySQL 版本期间可能出现的任何潜在问题。在尝试升级之前运行此命令，有助于提前确定和解决任何不兼容性，从而使实际升级过程更加顺畅。

此命令对每个表执行各种检查，例如：

- 验证表结构和元数据是否与目标 MySQL 版本兼容
- 检查表是否使用了任何已弃用或已移除的功能
- 确保表可以正确地升级而不会丢失数据

有关更多信息，请参阅 MySQL 文档中的 CHECK TABLE statement。

输出示例：

```
{
  "id": "checkTableMysqlSchema",
  "title": "Issues reported by 'check table x for upgrade' command for mysql
 schema.",
  "status": "OK",
  "detectedProblems": []
}
```

此项预检查的输出取决于遇到的错误以及遇到错误的时间，因为 `check table for upgrade` 会执行多项检查。

如果您在此项预检查中遇到任何错误，请向 AWS Support 提交案例，来请求解决元数据不一致性。或者，可以通过执行逻辑转储，然后还原到新的 Aurora MySQL 版本 3 数据库集群来重试升级。

circularDirectoryCheck

预检查级别：错误

表空间数据文件路径中的循环目录引用

从 MySQL 8.0.17 开始，`CREATE TABLESPACE ... ADD DATAFILE` 子句不再允许循环目录引用。为避免出现升级问题，请在升级到 Aurora MySQL 版本 3 之前，从表空间数据文件路径中移除所有循环目录引用。

输出示例：

```
{
  "id": "circularDirectory",
  "title": "Circular directory references in tablespace data file paths",
  "status": "OK",
  "description": "Error: Following tablespaces contain circular directory references
 (e.g. the reference '/../') in data file paths which as of MySQL 8.0.17 are not
 permitted by the CREATE TABLESPACE ... ADD DATAFILE clause. An exception to the
 restriction exists on Linux, where a circular directory reference is permitted if
 the preceding directory is a symbolic link. To avoid upgrade issues, remove any
 circular directory references from tablespace data file paths before upgrading.",
  "documentationLink": "https://dev.mysql.com/doc/refman/8.0/en/upgrading-from-
previous-series.html#upgrade-innodb-changes",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "ts2",
        "description": "circular reference in datafile path: '/home/ec2-user/dbdata/
mysql_5_7_44/../ts2.ibd'",
        "dbObjectType": "Tablespace"
      }
  ]
}
```

如果您收到此错误，请使用 file-per-table tablespace 来重建表。对所有表空间和表定义使用默认文件路径。

> **Note**
>
> Aurora MySQL 不支持一般表空间或 CREATE TABLESPACE 命令。
> 在重建表空间之前，请参阅 MySQL 文档中的 Online DDL operations，来了解锁定和数据移动对前台事务的影响。

重建后，预检查通过，支持升级继续进行。

```
{
  "id": "circularDirectoryCheck",
  "title": "Circular directory references in tablespace data file paths",
  "status": "OK",
  "detectedProblems": []
},
```

columnsWhichCannotHaveDefaultsCheck

预检查级别：错误

不能有默认值的列

在 MySQL 8.0.13 之前，BLOB、TEXT、GEOMETRY 和 JSON 列不能有 default values。在升级到 Aurora MySQL 版本 3 之前，请移除这些列上的所有默认子句。有关这些数据类型的默认处理方式的变化的更多信息，请参阅 MySQL 文档中的 Data type default values。

输出示例：

```
{
  "id": "columnsWhichCannotHaveDefaultsCheck",
  "title": "Columns which cannot have default values",
  "status": "OK",
  "description": "Error: The following columns are defined as either BLOB, TEXT,
 GEOMETRY or JSON and have a default value set. These data types cannot have default
 values in MySQL versions prior to 8.0.13, while starting with 8.0.13, the default
 value must be specified as an expression. In order to fix this issue, please use
 the ALTER TABLE ... ALTER COLUMN ... DROP DEFAULT statement.",
  "documentationLink": "https://dev.mysql.com/doc/refman/8.0/en/data-type-
defaults.html#data-type-defaults-explicit",
  "detectedProblems": [
      {
```

```
        "level": "Error",
        "dbObject": "test.test_blob_default.geo_col",
        "description": "geometry"
      }
  ]
},
```

预检查会返回错误，因为 test.test_blob_default 表中的 geo_col 列使用指定了默认值的 BLOB、TEXT、GEOMETRY 或 JSON 数据类型。

查看表定义，我们可以看到 geo_col 列定义为 geo_col geometry NOT NULL default ''。

```
mysql> show create table test_blob_default\G
*************************** 1. row ***************************
       Table: test_blob_default
Create Table: CREATE TABLE `test_blob_default` (
  `geo_col` geometry NOT NULL DEFAULT ''
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

移除此默认子句可支持预检查通过。

> **Note**
>
> 在运行 ALTER TABLE 语句或重建表空间之前，请参阅 MySQL 文档中的 Online DDL operations，来了解锁定和数据移动对前台事务的影响。

```
mysql> ALTER TABLE test_blob_default modify COLUMN geo_col geometry NOT NULL;
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> show create table test_blob_default\G
*************************** 1. row ***************************
       Table: test_blob_default
Create Table: CREATE TABLE `test_blob_default` (
  `geo_col` geometry NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
```

预检查通过，您可以重试升级。

```
{
  "id": "columnsWhichCannotHaveDefaultsCheck",
  "title": "Columns which cannot have default values",
  "status": "OK",
  "detectedProblems": []
},
```

depreciatedSyntaxCheck

预检查级别：错误

在定义中使用弃用的关键词

MySQL 8.0 移除了 query cache。因此，移除了一些特定于查询缓存的 SQL 语法。如果任何数据库对象包含 QUERY CACHE、SQL_CACHE 或 SQL_NO_CACHE 关键字，则会返回预检查错误。要解决此问题，请重新创建这些对象，并移除提到的关键字。

输出示例：

```
{
  "id": "depreciatedSyntaxCheck",
  "title": "Usage of depreciated keywords in definition",
  "status": "OK",
  "description": "Error: The following DB objects contain keywords like 'QUERY
 CACHE', 'SQL_CACHE', 'SQL_NO_CACHE' which are not supported in major version 8.0.
 It is recommended to drop these DB objects or rebuild without any of the above
 keywords before upgrade.",
  "detectedProblems": [
      {
"level": "Error",
"dbObject": "test.no_query_cache_check",
"description": "PROCEDURE uses depreciated words in definition"
      }
  ]
}
```

预检查报告 test.no_query_cache_check 存储过程正在使用其中一个已移除的关键字。查看过程定义，可以看到它使用了 SQL_NO_CACHE。

```
mysql> show create procedure test.no_query_cache_check\G
*************************** 1. row ***************************
```

```
           Procedure: no_query_cache_check
            sql_mode:
    Create Procedure: CREATE DEFINER=`reinvent`@`%` PROCEDURE
 `no_query_cache_check`()
BEGIN
    SELECT SQL_NO_CACHE k from sbtest1 where id > 10 and id < 20 group by k asc;
END
character_set_client: utf8mb4
collation_connection: utf8mb4_0900_ai_ci
  Database Collation: latin1_swedish_ci
1 row in set (0.00 sec)
```

移除该关键字。

```
mysql> drop procedure test.no_query_cache_check;
Query OK, 0 rows affected (0.01 sec)

mysql> delimiter //

mysql> CREATE DEFINER=`reinvent`@`%` PROCEDURE `no_query_cache_check`() BEGIN
 SELECT k from sbtest1 where id > 10 and id < 20 group by k asc; END//
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;
```

移除该关键字后，预检查成功完成。

```
{
  "id": "depreciatedSyntaxCheck",
  "title": "Usage of depreciated keywords in definition",
  "status": "OK",
  "detectedProblems": []
}
```

engineMixupCheck

预检查级别：错误

InnoDB 识别的属于不同引擎的表

与 schemaInconsistencyCheck 类似，此项预检查在继续升级之前验证 MySQL 中的表元数据是否一致。

如果您在此项预检查中遇到任何错误，请向 AWS Support 提交案例，来请求解决元数据不一致性。或者，可以通过执行逻辑转储，然后还原到新的 Aurora MySQL 版本 3 数据库集群来重试升级。

输出示例：

```
{
  "id": "engineMixupCheck",
  "title": "Tables recognized by InnoDB that belong to a different engine",
  "status": "OK",
  "description": "Error: Following tables are recognized by InnoDB engine while the
 SQL layer believes they belong to a different engine. Such situation may happen
 when one removes InnoDB table files manually from the disk and creates e.g. a
 MyISAM table with the same name.\n\nA possible way to solve this situation is to
 e.g. in case of MyISAM table:\n\n1. Rename the MyISAM table to a temporary name
 (RENAME TABLE).\n2. Create some dummy InnoDB table (its definition does not need to
 match), then copy (copy, not move) and rename the dummy .frm and .ibd files to the
 orphan name using OS file commands.\n3. The orphan table can be then dropped (DROP
 TABLE), as well as the dummy table.\n4. Finally the MyISAM table can be renamed
 back to its original name.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "mysql.general_log_backup",
        "description": "recognized by the InnoDB engine but belongs to CSV"
      }
  ]
}
```

enumSetElementLengthCheck

预检查级别：错误

ENUM 和 **SET** 列定义包含长度超过 255 个字符的元素

表和存储过程的 ENUM 或 SET 列元素不得超过 255 个字符或 1020 个字节。在 MySQL 8.0 之前，最大组合长度为 64 K，但 8.0 将单个元素限制为 255 个字符或 1020 字节（支持多字节）。如果针对 enumSetElementLengthCheck 的预检查失败，请在重试升级之前修改任何超出这些新限制的元素。

输出示例：

```
{
```

```
  "id": "enumSetElementLengthCheck",
  "title": "ENUM/SET column definitions containing elements longer than 255
 characters",
  "status": "OK",
  "description": "Error: The following columns are defined as either ENUM or SET and
 contain at least one element longer that 255 characters. They need to be altered so
 that all elements fit into the 255 characters limit.",
  "documentationLink": "https://dev.mysql.com/doc/refman/8.0/en/string-type-
overview.html",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.large_set.s",
        "description": "SET contains element longer than 255 characters"
      }
  ]
},
```

预检查会报告错误，因为 `test.large_set` 表中的列 `s` 包含一个大于 255 个字符的 SET 元素。

在减小此列的 SET 大小后，预检查通过，支持升级继续进行。

```
{
  "id": "enumSetElementLenghtCheck",
  "title": "ENUM/SET column definitions containing elements longer than 255
 characters",
  "status": "OK",
  "detectedProblems": []
},
```

foreignKeyLengthCheck

预检查级别：错误

外键约束名称长度超过 64 个字符

在 MySQL 中，标识符的长度限制为 64 个字符，如 MySQL documentation 中所述。由于发现了外键长度可能等于或超过该值的 issues，导致升级失败，因此实施了这项预检查。如果您在此项预检查中遇到错误，则应 alter or rename 约束，使其少于 64 个字符，然后重试升级。

输出示例：

```
{
```

```
  "id": "foreignKeyLength",
  "title": "Foreign key constraint names longer than 64 characters",
  "status": "OK",
  "detectedProblems": []
}
```

getDuplicateTriggers

预检查级别：错误

数据库中的所有触发器名称都必须是唯一的。

由于数据字典实现的变化，MySQL 8.0 不支持数据库中区分大小写的触发器。此项预检查可验证数据库集群中没有一个或多个包含重复触发器的数据库。有关更多信息，请参阅 MySQL 文档中的 Identifier case sensitivity。

输出示例：

```
{
  "id": "getDuplicateTriggers",
  "title": "MySQL pre-checks that all trigger names in a database are unique or
 not.",
  "status": "OK",
  "description": "Error: You have one or more database containing duplicate
 triggers. Mysql 8.0 does not support case sensitive triggers within a database
 https://dev.mysql.com/doc/refman/8.0/en/identifier-case-sensitivity.html. To
 upgrade to MySQL 8.0, drop the triggers with case-insensitive duplicate names and
 recreate with distinct names.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test",
        "description": "before_insert_product"
      },
      {
        "level": "Error",
        "dbObject": "test",
        "description": "before_insert_PRODUCT"
      }
  ]
}
```

此项预检查报告了一个错误，即数据库集群有两个同名但使用不同大小写的触发器：test.before_insert_product 和 test.before_insert_PRODUCT。

在升级之前，请重命名这些触发器，或删除它们并使用新名称重新创建。

将 test.before_insert_PRODUCT 重命名为 test.before_insert_product_2 后，预检查成功。

```
{
  "id": "getDuplicateTriggers",
  "title": "MySQL pre-checks that all trigger names in a database are unique or
 not.",
  "status": "OK",
  "detectedProblems": []
}
```

getEventsWithNullDefiner

预检查级别：错误

mysql.event 的定义者列不能为 null 或为空。

DEFINER 属性指定拥有存储对象定义（例如触发器、存储过程或事件）的 MySQL 账户。在您想要控制存储对象在其中运行的安全上下文的情况下，此属性特别有用。创建存储对象时，如果未指定 DEFINER，则默认为创建了该对象的用户。

升级到 MySQL 8.0 时，MySQL 数据字典中不能有任何具有 null 或空定义者的存储对象。如果有这样的存储对象，则会引发预检查错误。必须先修复此问题，然后才能继续升级。

错误示例：

```
{
  "id": "getEventsWithNullDefiner",
  "title": "The definer column for mysql.event cannot be null or blank.",
  "status": "OK",
  "description": "Error: Set definer column in mysql.event to a valid non-null
 definer.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.get_version",
```

```
        "description": "Set definer for event get_version in Schema test"
      }
  ]
}
```

由于 `test.get_version` event 具有 `null` 定义者，因此预检查会返回一个错误。

要解决这个问题，可以检查事件定义。如您所见，定义者为 `null` 或空。

```
mysql> select db,name,definer from mysql.event where name='get_version';
+------+-------------+---------+
| db   | name        | definer |
+------+-------------+---------+
| test | get_version |         |
+------+-------------+---------+
1 row in set (0.00 sec)
```

删除该事件，或使用有效的定义者重新创建该事件。

> **Note**
>
> 在删除或重新定义 DEFINER 之前，请仔细查看并检查您的应用程序和权限要求。有关更多信息，请参阅 MySQL 文档中的 Stored object access control。

```
mysql> drop event test.get_version;
Query OK, 0 rows affected (0.00 sec)

mysql> DELIMITER ;
mysql> delimiter $$
mysql> CREATE EVENT get_version
    ->     ON SCHEDULE
    ->       EVERY 1 DAY
    ->     DO
    ->      ///DO SOMETHING //
    -> $$
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;

mysql> select db,name,definer from mysql.event where name='get_version';
```

```
+------+-------------+------------+
| db   | name        | definer    |
+------+-------------+------------+
| test | get_version | reinvent@% |
+------+-------------+------------+
1 row in set (0.00 sec)
```

现在，预检查通过。

```
{
  "id": "getEventsWithNullDefiner",
  "title": "The definer column for mysql.event cannot be null or blank.",
  "status": "OK",
  "detectedProblems": []},
```

getMismatchedMetadata

预检查级别：错误

InnoDB 数据字典和实际表定义之间的列定义不匹配

与 schemaInconsistencyCheck 类似，此项预检查在继续升级之前验证 MySQL 中的表元数据是否一致。在这种情况下，预检查会验证 InnoDB 数据字典和 MySQL 表定义之间的列定义是否匹配。如果检测到不匹配，则升级将无法继续。

如果您在此项预检查中遇到任何错误，请向 AWS Support 提交案例，来请求解决元数据不一致性。或者，可以通过执行逻辑转储，然后还原到新的 Aurora MySQL 版本 3 数据库集群来重试升级。

输出示例：

```
{
  "id": "getMismatchedMetadata",
  "title": "Column definition mismatch between InnoDB Data Dictionary and actual
 table definition.",
  "status": "OK",
  "description": "Error: Your database has mismatched metadata. The upgrade to mysql
 8.0 will not succeed until this is fixed.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.mismatchTable",
```

```
        "description": "Table `test/mismatchTable` column names mismatch with InnoDb
 dictionary column names: iD <> id"
      }
  ]
}
```

预检查报告 `test.mismatchTable` 表中 `id` 列的元数据不匹配。具体而言，MySQL 元数据的列名称为 `iD`，而 InnoDB 的列名称为 `id`。

getTriggersWithNullDefiner

预检查级别：错误

`information_schema.triggers` 的定义者列不能为 `null` 或为空。

此项预检查验证您的数据库中没有使用 `null` 或空定义者定义的触发器。有关存储对象的定义者要求的更多信息，请参阅 getEventsWithNullDefiner。

输出示例：

```
{
  "id": "getTriggersWithNullDefiner",
  "title": "The definer column for information_schema.triggers cannot be null or
 blank.",
  "status": "OK",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.example_trigger",
        "description": "Set definer for trigger example_trigger in Schema test"
      }
  ]
}
```

由于 `test` 架构中的 `example_trigger` 触发器具有 `null` 定义者，因此预检查会返回错误。要更正此问题，请使用有效的用户重新创建触发器来修复定义者，或者删除此触发器。有关更多信息，请参阅 getEventsWithNullDefiner 中的示例。

> **Note**
>
> 在删除或重新定义 `DEFINER` 之前，请仔细查看并检查您的应用程序和权限要求。有关更多信息，请参阅 MySQL 文档中的 Stored object access control。

getValueOfVariablelower_case_table_names

预检查级别：错误

将 **lower_case_table_names** 参数设置为 **1** 时，所有数据库和表名称都必须为小写。

在 MySQL 8.0 之前，数据库名称、表名称和其它对象对应于数据目录中的文件，例如基于文件的元数据 (.frm)。lower_case_table_names 系统变量可让用户控制服务器如何处理数据库对象的标识符区分大小写，以及此类元数据对象的存储。重启后，可以在已经初始化的服务器上更改此参数。

但是，在 MySQL 8.0 中，虽然此参数仍然控制服务器如何处理标识符区分大小写，但在数据字典初始化后无法对其进行更改。升级或创建 MySQL 8.0 数据库时，首次在 MySQL 上启动数据字典时为 `lower_case_table_names` 设置的值将用于该数据库的生命周期。此限制是作为 Atomic Data Dictionary 实现的一部分而实施的，其中数据库对象从基于文件的元数据迁移到 `mysql` 架构中的内部 InnoDB 表。

有关更多信息，请参阅 MySQL 文档中的 Data dictionary changes。

为了避免在升级过程中将基于文件的元数据更新到新的原子数据字典时出现问题，此项预检查验证当 `lower_case_table_names = 1` 时，所有表都以小写存储在磁盘上。如果不是，则会返回预检查错误，您必须在继续升级之前更正元数据。

输出示例：

```
{
  "id": "getValueOfVariablelower_case_table_names",
  "title": "MySQL pre-checks that all database or table names are lowercase when the
 lower_case_table_names parameter is set to 1.",
  "status": "OK",
  "description": "Error: You have one or more databases or tables with uppercase
 letters in the names, but the lower_case_table_names parameter is set to 1. To
 upgrade to MySQL 8.0, either change all database or table names to lowercase, or
 set the parameter to 0.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.TEST",
        "description": "Table test.TEST contains one or more capital letters in name
 while lower_case_table_names = 1"
      }
  ]
}
```

由于表 `test.TEST` 包含大写字母，但 `lower_case_table_names` 设置为 `1`，因此返回错误。

要解决此问题，可以将表重命名为使用小写，或在开始升级之前修改数据库集群上的 `lower_case_table_names` 参数。

> **Note**
>
> 请仔细测试和查看有关 MySQL 中 case sensitivity 的文档，以及任何此类更改可能会如何影响您的应用程序。
> 另请查看 MySQL 8.0 文档，了解在 MySQL 8.0 中如何以不同的方式处理 lower_case_table_names。

groupByAscSyntaxCheck

预检查级别：错误

使用已移除的 **`GROUP BY ASC/DESC`** 语法

从 MySQL 8.0.13 开始，`GROUP BY` 子句的已弃用 `ASC` 或 `DESC` 语法已被移除。依赖于 `GROUP BY` 排序的查询现在可能产生不同的结果。要获得特定的排序顺序，请改用 `ORDER BY` 子句。如果数据库中存在使用此语法的任何对象，则在重试升级之前，必须使用 `ORDER BY` 子句重新创建这些对象。有关更多信息，请参阅 MySQL 文档中的 SQL changes。

输出示例：

```
{
  "id": "groupbyAscSyntaxCheck",
  "title": "Usage of removed GROUP BY ASC/DESC syntax",
  "status": "OK",
  "description": "Error: The following DB objects use removed GROUP BY ASC/DESC
 syntax. They need to be altered so that ASC/DESC keyword is removed from GROUP BY
 clause and placed in appropriate ORDER BY clause.",
  "documentationLink": "https://dev.mysql.com/doc/relnotes/mysql/8.0/en/
news-8-0-13.html#mysqld-8-0-13-sql-syntax",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.groupbyasc",
        "description": "PROCEDURE uses removed GROUP BY ASC syntax",
        "dbObjectType": "Routine"
```

```
      }
  ]
}
```

mysqlEmptyDotTableSyntaxCheck

预检查级别：错误

检查例程中是否使用了弃用的 **.<table>** 语法。

在 MySQL 8.0 中，例程不能再包含已弃用的标识符语法 (\".<table>\")。如果任何存储例程或触发器包含此类标识符，则升级将失败。例如，不再允许使用以下 .dot_table 引用：

```
mysql> show create procedure incorrect_procedure\G
*************************** 1. row ***************************
           Procedure: incorrect_procedure
            sql_mode:
    Create Procedure: CREATE DEFINER=`reinvent`@`%` PROCEDURE
 `incorrect_procedure`()
BEGIN delete FROM .dot_table; select * from .dot_table where 1=1; END
character_set_client: utf8mb4
collation_connection: utf8mb4_0900_ai_ci
  Database Collation: latin1_swedish_ci
1 row in set (0.00 sec)
```

重新创建例程和触发器以使用正确的标识符语法和转义后，预检查通过，升级就可以继续了。有关标识符的更多信息，请参阅 MySQL 文档中的 Schema object names。

输出示例：

```
{
  "id": "mysqlEmptyDotTableSyntaxCheck",
  "title": "Check for deprecated '.<table>' syntax used in routines.",
  "status": "OK",
  "description": "Error: The following routines contain identifiers in deprecated
 identifier syntax (\".<table>\"), and should be corrected before upgrade:\n",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.incorrect_procedure",
        "description": " routine body contains deprecated identifiers."
      }
```

```
  ]
}
```

预检查会返回 test 数据库中 incorrect_procedure 例程的错误，因为它包含已弃用的语法。

更正例程后，预检查成功，您可以重试升级。

mysqlIndexTooLargeCheck

预检查级别：错误

检查索引是否过大，无法在高于 5.7 的 MySQL 版本上发挥作用

对于紧凑或冗余的行格式，应无法创建前缀大于 767 字节的索引。但是，在 MySQL 版本 5.7.35 之前，这是可能的。有关更多信息，请参阅 MySQL 5.7.35 release notes。

升级到 MySQL 8.0 后，任何受此错误影响的索引都将变得无法访问。此项预检查可识别在可以升级继续之前必须重建的违规索引。

```
{
 "id": "mysqlIndexTooLargeCheck",
 "title": "Check for indexes that are too large to work on higher versions of MySQL
Server than 5.7",
 "status": "OK",
 "description": "Error: The following indexes ware made too large for their format
in an older version of MySQL (older than 5.7.34). Normally those indexes within
tables with compact or redundant row formats shouldn't be larger than 767 bytes. To
fix this problem those indexes should be dropped before upgrading or those tables
will be inaccessible.",
 "detectedProblems": [
     {
       "level": "Error",
       "dbObject": "test.table_with_large_idx",
       "description": "IDX_2"
     }
 ]
}
```

预检查会返回错误，因为 test.table_with_large_idx 表包含对于某个表的索引，此索引使用压缩或冗余行格式且大于 767 字节。升级到 MySQL 8.0 后，这些表将变得无法访问。在继续升级之前，请执行以下操作之一：

- 删除预检查中提到的索引。

- 添加预检查中提到的索引。
- 更改表使用的行格式。

在此，我们重建表来解决预检查失败问题。在重建表之前，请确保 innodb_file_format 设置为 Barracuda，innodb_default_row_format 设置为 dynamic。这些是 MySQL 5.7 中的默认设置。有关更多信息，请参阅 MySQL 文档中的 InnoDB row formats 和 InnoDB file-format management。

> **Note**
>
> 在重建表空间之前，请参阅 MySQL 文档中的 Online DDL operations，来了解锁定和数据移动对前台事务的影响。

```
mysql > select @@innodb_file_format,@@innodb_default_row_format;
+----------------------+-----------------------------+
| @@innodb_file_format | @@innodb_default_row_format |
+----------------------+-----------------------------+
| Barracuda            | dynamic                     |
+----------------------+-----------------------------+
1 row in set (0.00 sec)

mysql> optimize table table_with_large_idx;
+---------------------------+----------+----------
+-------------------------------------------------------------------------+
| Table                     | Op       | Msg_type | Msg_text
                                |
+---------------------------+----------+----------
+-------------------------------------------------------------------------+
| test.table_with_large_idx | optimize | note     | Table does not support optimize,
 doing recreate + analyze instead |
| test.table_with_large_idx | optimize | status   | OK
                                |
+---------------------------+----------+----------
+-------------------------------------------------------------------------+
2 rows in set (0.02 sec)

# Verify FILE_FORMAT and ROW_FORMAT
mysql>  select * from information_schema.innodb_sys_tables where name like 'test/
table_with_large_idx';
+----------+---------------------------+------+--------+-------+-------------
+------------+---------------+------------+
```

```
| TABLE_ID | NAME                          | FLAG | N_COLS | SPACE | FILE_FORMAT |
 ROW_FORMAT | ZIP_PAGE_SIZE | SPACE_TYPE |
+----------+-------------------------------+------+--------+-------+-------------
+------------+---------------+------------+
|       43 | test/table_with_large_idx |   33 |      4 |    26 | Barracuda   |
 Dynamic    |             0 | Single     |
+----------+-------------------------------+------+--------+-------+-------------
+------------+---------------+------------+
1 row in set (0.00 sec)
```

重建表后，预检查通过，升级可以继续进行。

```
{
  "id": "mysqlIndexTooLargeCheck",
  "title": "Check for indexes that are too large to work on higher versions of MySQL
 Server than 5.7",
  "status": "OK",
  "detectedProblems": []
},
```

mysqlInvalid57NamesCheck

预检查级别：错误

检查 MySQL 5.7 中使用的无效表名称和架构名称

迁移到 MySQL 8.0 中的新数据字典时，数据库实例不能包含前缀为 `#mysql50#` 的架构或表。如果存在任何此类对象，则升级将失败。要解决此问题，请对返回的架构和表运行 mysqlcheck。

> **Note**
>
> 请务必使用 MySQL 5.7 version 的 `mysqlcheck`，因为 --fix-db-names 和 --fix-table-names 已从 MySQL 8.0 中移除。

输出示例：

```
{
  "id": "mysqlInvalid57NamesCheck",
  "title": "Check for invalid table names and schema names used in 5.7",
  "status": "OK",
```

```
  "description": "The following tables and/or schemas have invalid names. In order
 to fix them use the mysqlcheck utility as follows:\n\n  $ mysqlcheck --check-
upgrade --all-databases\n  $ mysqlcheck --fix-db-names --fix-table-names --all-
databases\n\nOR via mysql client, for eg:\n\n  ALTER DATABASE `#mysql50#lost+found`
 UPGRADE DATA DIRECTORY NAME;",
  "documentationLink": "https://dev.mysql.com/doc/refman/5.7/en/identifier-
mapping.html https://dev.mysql.com/doc/refman/5.7/en/alter-database.html https://
dev.mysql.com/doc/refman/8.0/en/mysql-nutshell.html#mysql-nutshell-removals",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "#mysql50#fix_db_names",
        "description": "Schema name"
      }
  ]
}
```

预检查报告架构 #mysql50#fix_db_names 的名称无效。

修复架构名称后，预检查通过，支持升级继续进行。

```
{
  "id": "mysqlInvalid57NamesCheck",
  "title": "Check for invalid table names and schema names used in 5.7",
  "status": "OK",
  "detectedProblems": []
},
```

mysqlOrphanedRoutinesCheck

预检查级别：错误

检查 5.7 中的孤立例程

在 MySQL 8.0 中迁移到新的数据字典时，如果数据库中有任何存储过程不再存在架构，则升级失败。此项预检查可验证数据库实例上的存储过程中引用的所有架构是否仍存在。要让升级继续进行，请删除这些存储过程。

输出示例：

```
{
  "id": "mysqlOrphanedRoutinesCheck",
  "title": "Check for orphaned routines in 5.7",
```

```
  "status": "OK",
  "description": "Error: The following routines have been orphaned. Schemas that
 they are referencing no longer exists.\nThey have to be cleaned up or the upgrade
 will fail.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "dropped_db.get_version",
        "description": "is orphaned"
      }
  ]
},
```

预检查报告 dropped_db 数据库中的 get_version 存储过程是孤立的。

要清理此过程，可以重新创建缺失的架构。

```
mysql> create database dropped_db;
Query OK, 1 row affected (0.01 sec)
```

重新创建架构后，可以删除该过程来让升级继续进行。

```
{
  "id": "mysqlOrphanedRoutinesCheck",
  "title": "Check for orphaned routines in 5.7",
  "status": "OK",
  "detectedProblems": []
},
```

mysqlSchemaCheck

预检查级别：错误

mysql 架构中的表名称与 MySQL 8.0 中的新表冲突

MySQL 8.0 中引入的新 Atomic Data Dictionary 将所有元数据存储在 mysql 架构的一组内部 InnoDB 表中。在升级期间，将在 mysql 架构中创建新的 internal data dictionary tables。为避免产生命名冲突而导致升级失败，预检查会检查 mysql 架构中的所有表名称，来确保没有任何新的表名称已被使用。如果是这样，则会返回错误，并且不支持继续升级。

输出示例：

```
{
```

```
  "id": "mysqlSchema",
  "title": "Table names in the mysql schema conflicting with new tables in the
 latest MySQL.",
  "status": "OK",
  "description": "Error: The following tables in mysql schema have names that will
 conflict with the ones introduced in the latest version. They must be renamed or
 removed before upgrading (use RENAME TABLE command). This may also entail changes
 to applications that use the affected tables.",
  "documentationLink": "https://dev.mysql.com/doc/refman/8.0/en/upgrade-before-you-
begin.html",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "mysql.tablespaces",
        "description": "Table name used in mysql schema.",
        "dbObjectType": "Table"
      }
  ]
}
```

返回错误是因为 `mysql` 架构中有一个名为 `tablespaces` 的表。这是 MySQL 8.0 中新的内部数据字典表名称之一。在升级之前，必须使用 `RENAME TABLE` 命令重命名任何此类表，或移除它们。

nonNativePartitioningCheck

预检查级别：错误

使用具有非原生分区的引擎对表进行分区

根据 MySQL 8.0 documentation，两个存储引擎目前提供原生分区支持：InnoDB 和 NDB。其中，Aurora MySQL 版本 3 只支持 InnoDB，它与 MySQL 8.0 兼容。使用任何其它存储引擎在 MySQL 8.0 中创建分区表的任何尝试都会失败。此项预检查会查找数据库集群中使用非原生分区的表。如果返回任何表，则必须移除分区或将存储引擎转换为 InnoDB。

输出示例：

```
{
  "id": "nonNativePartitioning",
  "title": "Partitioned tables using engines with non native partitioning",
  "status": "OK",
  "description": "Error: In the latest MySQL storage engine is responsible for
 providing its own partitioning handler, and the MySQL server no longer provides
 generic partitioning support. InnoDB and NDB are the only storage engines that
 provide a native partitioning handler that is supported in the latest MySQL. A
```

```
 partitioned table using any other storage engine must be altered—either to convert
 it to InnoDB or NDB, or to remove its partitioning—before upgrading the server,
 else it cannot be used afterwards.",
  "documentationLink": "https://dev.mysql.com/doc/refman/8.0/en/upgrading-from-
previous-series.html#upgrade-configuration-changes",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.partMyisamTable",
         "description": "MyISAM engine does not support native partitioning",
         "dbObjectType": "Table"
      }
  ]
}
```

在此处，MyISAM 表正在使用分区，需要先执行操作才能继续升级。

oldTemporalCheck

预检查级别：错误

使用旧的时间类型

“旧时间”是指在 MySQL 版本 5.5 及更低版本中创建的时间类型列（例如 TIMESTAMP 和 DATETIME）。在 MySQL 8.0 中，移除了对这些旧的时间数据类型的支持，这意味着，如果数据库包含这些旧的时间类型，则无法从 MySQL 5.7 就地升级到 8.0。要修复此问题，在继续升级之前，必须 rebuild 包含此类旧时间数据类型的所有表。

有关在 MySQL 5.7 中弃用旧的时间数据类型的更多信息，请参阅此 blog。有关在 MySQL 8.0 中移除旧的时间数据类型的更多信息，请参阅此 blog。

> **Note**
>
> 在重建表空间之前，请参阅 MySQL 文档中的 Online DDL operations，来了解锁定和数据移动对前台事务的影响。

输出示例：

```
{
  "id": "oldTemporalCheck",
  "title": "Usage of old temporal type",
  "status": "OK",
```

```
  "description": "Error: Following table columns use a deprecated and no longer
 supported temporal disk storage format. They must be converted to the new format
 before upgrading. It can by done by rebuilding the table using 'ALTER TABLE
 <table_name> FORCE' command",
  "documentationLink": "https://dev.mysql.com/blog-archive/mysql-8-0-removing-
support-for-old-temporal-datatypes/",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.55_temporal_table.timestamp_column",
        "description": "timestamp /* 5.5 binary format */",
        "dbObjectType": "Column"
      }
  ]
},
```

表 `test.55_temporal_table` 中的 `timestamp_column` 列报告了一个错误，因为它使用了不再支持的旧时间磁盘存储格式。

要解决此问题并让升级继续进行，请重建表，来将旧时间磁盘存储格式转换为 MySQL 5.6 中引入的新格式。有关在执行此操作之前的更多信息和先决条件，请参阅 MySQL 文档中的 Converting between 3-byte and 4-byte Unicode character sets。

运行以下命令来重建此项预检查中提到的表，可将旧的时间类型转换为具有小数秒精度的较新格式。

```
ALTER TABLE ... ENGINE=InnoDB;
```

有关重建表的更多信息，请参阅 MySQL 文档中的 ALTER TABLE statement。

重建相关表并重新启动升级后，兼容性检查通过，并支持升级继续进行。

```
{
  "id": "oldTemporalCheck",
  "title": "Usage of old temporal type",
  "status": "OK",
  "detectedProblems": []
}
```

partitionedTablesInSharedTablespaceCheck

预检查级别：错误

在共享表空间中使用了分区表

从 MySQL 8.0.13 开始，不再支持在共享表空间中放置表分区。升级之前，请将任何此类表从共享表空间移至 file-per-table 表空间。

> **Note**
>
> 在重建表空间之前，请参阅 MySQL 文档中的 Partitioning operations，来了解锁定和数据移动对前台事务的影响。

输出示例：

```
{
  "id": "partitionedTablesInSharedTablespaceCheck",
  "title": "Usage of partitioned tables in shared tablespaces",
  "status": "OK",
  "description": "Error: The following tables have partitions in shared tablespaces.
 They need to be moved to file-per-table tablespace before upgrading. You can do
 this by running query like 'ALTER TABLE table_name REORGANIZE PARTITION X INTO
 (PARTITION X VALUES LESS THAN (30) TABLESPACE=innodb_file_per_table);'",
  "documentationLink": "https://dev.mysql.com/doc/refman/8.0/en/mysql-
nutshell.html#mysql-nutshell-removals",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.partInnoDBTable",
        "description": "Partition p1 is in shared tablespace innodb",
        "dbObjectType": "Table"
      }
  ]
}
```

预检查失败，因为表 `test.partInnoDBTable` 中的分区 p1 位于系统表空间中。

要解决此问题，请重建 `test.partInnodbTable` 表，将违规分区 p1 放在 file-per-table 表空间中。

```
mysql > ALTER TABLE partInnodbTable REORGANIZE PARTITION p1
    ->   INTO (PARTITION p1 VALUES LESS THAN ('2014-01-01')
 TABLESPACE=innodb_file_per_table);
Query OK, 0 rows affected, 1 warning (0.02 sec)
```

```
Records: 0  Duplicates: 0  Warnings: 0
```

完成后，预检查通过。

```
{
  "id": "partitionedTablesInSharedTablespaceCheck",
  "title": "Usage of partitioned tables in shared tablespaces",
  "status": "OK",
  "detectedProblems": []
}
```

removedFunctionsCheck

预检查级别：错误

使用了从最新 MySQL 版本中移除的函数

在 MySQL 8.0 中，已移除了许多内置函数。此项预检查会检查数据库中是否存在可能使用这些函数的对象。如果找到此类对象，则将返回错误。在重试升级之前，必须先解决这些问题。

移除的大多数函数是 spatial functions，这些函数已被等效的 ST_* 函数所取代。在这些情况下，可以修改数据库对象来使用新的过程命名。有关更多信息，请参阅 MySQL 文档中的 MySQL 8.0 中删除的功能。

输出示例：

```
{
  "id": "removedFunctionsCheck",
  "title": "Usage of removed functions",
  "status": "OK",
  "description": "Error: The following DB objects use functions that were removed
 in the latest MySQL version. Please make sure to update them to use supported
 alternatives before upgrade.",
  "documentationLink": "https://dev.mysql.com/doc/refman/8.0/en/mysql-
nutshell.html#mysql-nutshell-removals",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.GetLocationsInPolygon",
        "description": "PROCEDURE uses removed function POLYGONFROMTEXT (consider
 using ST_POLYGONFROMTEXT instead)",
        "dbObjectType": "Routine"
```

```
      },
      {
        "level": "Error",
        "dbObject": "test.InsertLocation",
        "description": "PROCEDURE uses removed function POINTFROMTEXT (consider
 using ST_POINTFROMTEXT instead)",
        "dbObjectType": "Routine"
      }
  ]
},
```

预检查报告 `test.GetLocationsInPolygon` 存储过程正在使用两个已移除的函数：POLYGONFROMTEXT 和 POINTFROMTEXT。它还建议您使用新的 ST_POLYGONFROMTEXT 和 ST_POINTFROMTEXT 作为替换函数。使用建议重新创建该过程后，预检查成功完成。

```
{
  "id": "removedFunctionsCheck",
  "title": "Usage of removed functions",
  "status": "OK",
  "detectedProblems": []
},
```

> **Note**
>
> 虽然在大多数情况下，弃用的函数有直接的替换函数，但请务必测试您的应用程序并查看文档，来了解由于更改而导致的任何行为变化。

routineSyntaxCheck

预检查级别：错误

MySQL 语法检查是否有类似于例程的对象

MySQL 8.0 引入了以前未保留的 reserved keywords。升级预检查工具将评估保留关键字在数据库对象名称及其定义和正文中的使用情况。如果它检测到数据库对象（例如存储过程、函数、事件和触发器）中正在使用保留关键字，则升级将失败并向 `upgrade-prechecks.log` 文件中发布一个错误。要解决此问题，必须在升级之前，更新这些对象定义并将所有此类引用括在单引号（'）中。有关在 MySQL 中对保留字进行转义的更多信息，请参阅 MySQL 文档中的 String literals。

或者，可以将名称更改为其它名称，这可能需要更改应用程序。

输出示例：

```
{
  "id": "routineSyntaxCheck",
  "title": "MySQL syntax check for routine-like objects",
  "status": "OK",
  "description": "The following objects did not pass a syntax check with the latest
 MySQL grammar. A common reason is that they reference names that conflict with new
 reserved keywords. You must update these routine definitions and `quote` any such
 references before upgrading.",
  "documentationLink": "https://dev.mysql.com/doc/refman/en/keywords.html",
  "detectedProblems": [
      {
         "level": "Error",
         "dbObject": "test.select_res_word",
         "description": "at line 2,18: unexpected token 'except'",
         "dbObjectType": "Routine"
      }
  ]
}
```

要解决此问题，请检查例程定义。

```
SHOW CREATE PROCEDURE test.select_res_word\G

*************************** 1. row ***************************
           Procedure: select_res_word
            sql_mode:
 ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_Z
    Create Procedure: CREATE PROCEDURE 'select_res_word'()
BEGIN
    SELECT * FROM except;
END
character_set_client: utf8
collation_connection: utf8_general_ci
  Database Collation: latin1_swedish_ci
1 row in set (0.00 sec)
```

该过程使用名为 except 的表，这是 MySQL 8.0 中的一个新关键字。通过对字符串文本进行转义来重新创建该过程。

```
> drop procedure if exists select_res_word;
Query OK, 0 rows affected (0.00 sec)

> DELIMITER $$
 > CREATE PROCEDURE select_res_word()
    -> BEGIN
    ->     SELECT * FROM 'except';
    -> END$$
Query OK, 0 rows affected (0.00 sec)

 > DELIMITER ;
```

预检查现已通过。

```
{
  "id": "routineSyntaxCheck",
  "title": "MySQL syntax check for routine-like objects",
  "status": "OK",
  "detectedProblems": []
}
```

schemaInconsistencyCheck

预检查级别：错误

文件移除或损坏导致的架构不一致

如前所述，MySQL 8.0 引入了 Atomic Data Dictionary，它将所有元数据存储在 `mysql` 架构内的一组内部 InnoDB 表中。这种新架构提供了一种事务性、符合 ACID 的方法来管理数据库元数据，解决了基于文件的旧方法中的“原子 DDL”问题。在 MySQL 8.0 之前，如果 DDL 操作意外中断，架构对象可能会变成孤立对象。在升级期间将基于文件的元数据迁移到新的原子数据字典表，可确保数据库实例中没有此类孤立的架构对象。如果遇到任何孤立对象，则升级将失败。

为了有助于在启动升级之前检测这些孤立对象，将运行 `schemaInconsistencyCheck` 预检查，来确保所有数据字典元数据对象都处于同步状态。如果检测到任何孤立的元数据对象，则升级不会继续。要继续升级，请清理这些孤立的元数据对象。

如果您在此项预检查中遇到任何错误，请向 AWS Support 提交案例，来请求解决元数据不一致性。或者，可以通过执行逻辑转储，然后还原到新的 Aurora MySQL 版本 3 数据库集群来重试升级。

输出示例：

```
{
  "id": "schemaInconsistencyCheck",
  "title": "Schema inconsistencies resulting from file removal or corruption",
  "status": "OK",
  "description": "Error: Following tables show signs that either table datadir
 directory or frm file was removed/corrupted. Please check server logs, examine
 datadir to detect the issue and fix it before upgrade",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.schemaInconsistencyCheck_failure",
        "description": "present in INFORMATION_SCHEMA's INNODB_SYS_TABLES table but
 missing from TABLES table"
      }
  ]
}
```

预检查报告 `test.schemaInconsistencyCheck_failure` 表的元数据不一致。在这种情况下，表存在于 InnoDB 存储引擎元数据 (`information_schema.INNODB_SYS_TABLES`) 中，但不存在于 MySQL 元数据 (`information_schema.TABLES`) 中。

报告错误的 Aurora MySQL 预检查

以下预检查特定于 Aurora MySQL：

- auroraCheckDDLRecovery
- auroraCheckRdsUpgradePrechecksTable
- auroraFODUpgradeCheck
- auroraGetDanglingFulltextIndex
- auroraUpgradeCheckForDatafilePathInconsistency
- auroraUpgradeCheckForFtsSpaceIdZero
- auroraUpgradeCheckForIncompleteXATransactions
- auroraUpgradeCheckForInstanceLimit
- auroraUpgradeCheckForInternalUsers
- auroraUpgradeCheckForMasterUser

- auroraUpgradeCheckForPrefixIndexOnGeometryColumns
- auroraUpgradeCheckForSpecialCharactersInProcedures
- auroraUpgradeCheckForSysSchemaObjectTypeMismatch
- auroraUpgradeCheckForViewColumnNameLength
- auroraUpgradeCheckIndexLengthLimitOnTinyColumns
- auroraUpgradeCheckMissingInnodbMetadataForMysqlHostTable
- auroraUpgradeCheckForInvalidUtf8mb3ColumnComments

auroraCheckDDLRecovery

预检查级别：错误

检查是否存在与 Aurora DDL 恢复功能相关的构件

作为 Aurora MySQL 中数据定义语言（DDL）恢复实现的一部分，正在运行的 DDL 语句的元数据保留在 `mysql` 架构内的 `ddl_log_md_table` 和 `ddl_log_table` 表中。Aurora 对于 DDL 恢复的实现在版本 3 及后续版本中不受支持，因为该功能是 MySQL 8.0 中新的 Atomic Data Dictionary 实现的一部分。如果在兼容性检查期间有任何 DDL 语句正在运行，则此项预检查可能会失败。我们建议您在没有任何 DDL 语句正在运行的情况下尝试升级。

如果在没有任何 DDL 语句正在运行的情况下，此项预检查失败，请向 AWS Support 提交案例，来请求解决元数据不一致问题。或者，可以通过执行逻辑转储，然后还原到新的 Aurora MySQL 版本 3 数据库集群来重试升级。

如果有任何 DDL 语句正在运行，则预检查输出将显示以下消息：

```
“There are DDL statements in process. Please allow DDL statements to finish before
 upgrading.”
```

输出示例：

```
{
  "id": "auroraCheckDDLRecovery",
  "title": "Check for artifacts related to Aurora DDL recovery feature",
  "status": "OK",
  "description": "Aurora implementation of DDL recovery is not supported from 3.x
 onwards. This check verifies that the database do not have artifacts realted to the
 feature",
```

```
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "mysql.ddl_log_md_table",
        "description": "Table mysql.ddl_log_md_table is not empty. Your database has
 pending DDL recovery operations. Reachout to AWS support for assistance."
      },
      {
        "level": "Error",
        "dbObject": "mysql.ddl_log_table",
        "description": "Table mysql.ddl_log_table is not empty. Your database has
 pending DDL recovery operations. Reachout to AWS support for assistance."
      },
      {
        "level": "Error",
        "dbObject": "information_schema.processlist",
        "description": "There are DDL statements in process. Please allow DDL
 statements to finish before upgrading."
      }
  ]
}
```

由于正在运行的 DDL 与兼容性检查同时运行，因此预检查返回了一个错误。我们建议您在没有任何 DDL 正在运行的情况下重试升级。

auroraCheckRdsUpgradePrechecksTable

预检查级别：错误

检查 **mysql.rds_upgrade_prechecks** 表是否存在

这是 RDS 服务执行的仅限内部的预检查。任何错误都将在升级时自动得以处理，可以安全地忽略。

如果您在此项预检查中遇到任何错误，请向 AWS Support 提交案例，来请求解决元数据不一致性。或者，可以通过执行逻辑转储，然后还原到新的 Aurora MySQL 版本 3 数据库集群来重试升级。

```
{
  "id": "auroraCheckRdsUpgradePrechecksTable",
  "title": "Check existence of mysql.rds_upgrade_prechecks table",
  "status": "OK",
  "detectedProblems": []
```

```
}
```

auroraFODUpgradeCheck

预检查级别：错误

检查是否存在与 Aurora 快速 DDL 功能相关的构件

Aurora MySQL 版本 2 在实验室模式下引入了快速 DDL 优化，来提高某些 DDL 操作的效率。在 Aurora MySQL 版本 3 中，实验室模式已被移除，快速 DDL 实现已被名为 Instant DDL 的 MySQL 8.0 功能所取代。

在升级到 Aurora MySQL 版本 3 之前，任何在实验室模式下使用快速 DDL 的表都必须通过运行 `OPTIMIZE TABLE` 或 `ALTER TABLE ... ENGINE=InnoDB` 命令进行重建，以确保与 Aurora MySQL 版本 3 兼容。

此项预检查将返回任何此类表的列表。重建返回的表后，可以重试升级。

输出示例：

```
{
  "id": "auroraFODUpgradeCheck",
  "title": "Check for artifacts related to Aurora fast DDL feature",
  "status": "OK",
  "description": "Aurora fast DDL is not supported from 3.x onwards. This check
 verifies that the database does not have artifacts related to the feature",
  "documentationLink": "https://docs.aws.amazon.com/AmazonRDS/latest/
AuroraUserGuide/AuroraMySQL.Managing.FastDDL.html#AuroraMySQL.Managing.FastDDL-v2",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.test",
        "description": "Your table has pending Aurora fast DDL operations. Run
 'OPTIMIZE TABLE <table name>' for the table to apply all the pending DDL updates.
 Then try the upgrade again."
      }
  ]
}
```

预检查报告表 `test.test` 具有待处理的快速 DDL 操作。

要让升级继续进行，可以重建表，然后重试升级。

> **Note**
>
> 在重建表空间之前，请参阅 MySQL 文档中的 Online DDL operations，来了解锁定和数据移动对前台事务的影响。

```
mysql> optimize table test.test;
+-----------+----------+----------
+-------------------------------------------------------------------+
| Table     | Op       | Msg_type | Msg_text
                  |
+-----------+----------+----------
+-------------------------------------------------------------------+
| test.test | optimize | note     | Table does not support optimize, doing recreate
 + analyze instead |
| test.test | optimize | status   | OK
                  |
+-----------+----------+----------
+-------------------------------------------------------------------+
2 rows in set (0.04 sec)
```

重建表后，预检查成功。

```
{
  "id": "auroraFODUpgradeCheck",
  "title": "Check for artifacts related to Aurora fast DDL feature",
  "status": "OK",
  "detectedProblems": []
}
```

auroraGetDanglingFulltextIndex

预检查级别：错误

带有悬挂的 **FULLTEXT** 索引引用的表

在 MySQL 5.6.26 之前，删除全文搜索索引后，隐藏的 FTS_DOC_ID 和 FTS_DOC_ID_INDEX 列可能会变成孤立的列。有关更多信息，请参阅 Bug #76012。

如果您在早期版本的 MySQL 上创建了任何出现这种情况的表，则可能会导致升级到 Aurora MySQL 版本 3 失败。此项预检查可验证在升级到 MySQL 8.0 之前，数据库集群上不存在此类孤立或“悬挂”的全文索引。如果此项预检查失败，请重建任何包含此类悬挂全文索引的表。

输出示例：

```
{
  "id": "auroraGetDanglingFulltextIndex",
  "title": "Tables with dangling FULLTEXT index reference",
  "status": "OK",
  "description": "Error: The following tables contain dangling FULLTEXT index which
 is not supported. It is recommended to rebuild the table before upgrade.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.table_with_fts_index",
        "description": "Table `test.table_with_fts_index` contains dangling FULLTEXT
 index. Kindly recreate the table before upgrade."
      }
  ]
},
```

预检查报告 `test.table_with_fts_index` 表存在错误，因为它包含悬挂的全文索引。要让升级继续进行，请重建表来清理全文索引辅助表。使用 `OPTIMIZE TABLE test.table_with_fts_index` 或者 `ALTER TABLE test.table_with_fts_index, ENGINE=INNODB`。

重建表后，预检查通过。

```
{
  "id": "auroraGetDanglingFulltextIndex",
  "title": "Tables with dangling FULLTEXT index reference",
  "status": "OK",
  "detectedProblems": []
},
```

auroraUpgradeCheckForDatafilePathInconsistency

预检查级别：错误

检查与 **ibd** 文件路径相关的不一致性

此项预检查仅适用于 Aurora MySQL 版本 3.03.0 及更低版本。如果您在此项预检查中遇到错误，请升级到 Aurora MySQL 版本 3.04 或更高版本。

输出示例：

```
{
  "id": "auroraUpgradeCheckForDatafilePathInconsistency",
  "title": "Check for inconsistency related to ibd file path.",
  "status": "OK",
  "detectedProblems": []
}
```

auroraUpgradeCheckForFtsSpaceIdZero

预检查级别：错误

检查是否有空间 ID 为零的全文索引

在 MySQL 中，当您向 InnoDB 表添加 full-text index 时，会创建许多辅助索引表空间。由于 MySQL 的早期版本中存在一个 bug（在 5.6.20 版本中已修复），这些辅助索引表可能是在 system tablespace 中创建的，而不是在它们自己的 InnoDB 表空间中创建的。

如果存在任何此类辅助表空间，则升级将失败。重新创建在预检查错误中提到的全文索引，然后重试升级。

输出示例：

```
{
  "id": "auroraUpgradeCheckForFtsSpaceIdZero",
  "title": "Check for fulltext index with space id as zero",
  "status": "OK",
  "description": "The auxiliary tables of FTS indexes on the table are created
 in system table-space. Due to this DDL queries executed on MySQL8.0 shall cause
 database unavailability. To avoid that, drop and recreate all the FTS indexes on
 the table or rebuild the table using ALTER TABLE query before the upgrade.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.fts_space_zero_check",
        "description": " The auxiliary tables of FTS indexes on the table
 'test.fts_space_zero_check' are created in system table-space due to https://
bugs.mysql.com/bug.php?id=72132. In MySQL8.0, DDL queries executed on this table
 shall cause database unavailability. To avoid that, drop and recreate all the
 FTS indexes on the table or rebuild the table using ALTER TABLE query before the
 upgrade."
      }
  ]
```

```
},
```

预检查报告 `test.fts_space_zero_check` 表的一个错误，因为该表在系统表空间中有辅助全文搜索（FTS）表。

删除并重新创建与此表关联的 FTS 索引后，预检查成功。

> **Note**
>
> 在重建表空间之前，请参阅 MySQL 文档中的 Partitioning operations，来了解锁定和数据移动对前台事务的影响。

```
{
 "id": "auroraUpgradeCheckForFtsSpaceIdZero",
 "title": "Check for fulltext index with space id as zero",
 "status": "OK",
 "detectedProblems": []
}
```

auroraUpgradeCheckForIncompleteXATransactions

预检查级别：错误

检查 XA 事务是否处于已准备状态

在运行主要版本升级过程时，务必对 Aurora MySQL 版本 2 数据库实例执行 clean shutdown。这可以确保所有事务都已提交或回滚，并且 InnoDB 已清除所有撤消日志记录。由于事务回滚是必需的，因此，如果数据库中有任何 XA transactions 处于已准备状态，则可能会阻止继续进行干净关闭。因此，如果检测到任何已准备的 XA 事务，则在您采取措施来提交或回滚这些事务之前，升级将无法继续。

有关使用 `XA RECOVER` 查找处于已准备状态的 XA 事务的更多信息，请参阅 MySQL 文档中的 XA transaction SQL statements。有关 XA 事务状态的更多信息，请参阅 MySQL 文档中的 XA transaction states。

输出示例：

```
{
  "id": "auroraUpgradeCheckForIncompleteXATransactions",
```

```
  "title": "Pre-checks for XA Transactions in prepared state.",
  "status": "OK",
  "description": "Your cluster currently has XA transactions in the prepared state.
 To proceed with the upgrade, commit or rollback these transactions.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "all",
        "description": "Your cluster currently has XA transactions in the prepared
 state. To proceed with the upgrade, commit or rollback these transactions."
      }
  ]
}
```

此项预检查报告一个错误，因为有处于已准备状态的事务应提交或回滚。

登录数据库后，可以检查 information_schema.innodb_trx 表和 XA RECOVER 输出来获取更多信息。

> **Important**
>
> 在提交或回滚事务之前，建议您查看 MySQL documentation 和您的应用程序要求。

```
mysql> select trx_started,
    trx_mysql_thread_id,
    trx_id,trx_state,
    trx_operation_state,
    trx_rows_modified,
    trx_rows_locked
from
    information_schema.innodb_trx;
+---------------------+--------------------+---------+-----------
+---------------------+------------------+-----------------+
| trx_started         | trx_mysql_thread_id | trx_id  | trx_state |
 trx_operation_state | trx_rows_modified | trx_rows_locked |
+---------------------+--------------------+---------+-----------
+---------------------+------------------+-----------------+
| 2024-08-12 01:09:39 |                  0 | 2849470 | RUNNING   | NULL
    |                 1 |               0 |
+---------------------+--------------------+---------+-----------
+---------------------+------------------+-----------------+
```

```
1 row in set (0.00 sec)

mysql> xa recover;
+----------+--------------+--------------+--------+
| formatID | gtrid_length | bqual_length | data   |
+----------+--------------+--------------+--------+
|        1 |            6 |            0 | xatest |
+----------+--------------+--------------+--------+
1 row in set (0.00 sec)
```

在这种情况下，我们会回滚已准备的事务。

```
mysql> XA ROLLBACK 'xatest';
Query OK, 0 rows affected (0.00 sec)
v
mysql> xa recover;
Empty set (0.00 sec)
```

XA 事务回滚后，预检查成功。

```
{
  "id": "auroraUpgradeCheckForIncompleteXATransactions",
  "title": "Pre-checks for XA Transactions in prepared state.",
  "status": "OK",
  "detectedProblems": []
}
```

auroraUpgradeCheckForInstanceLimit

预检查级别：错误

检查当前实例类是否支持升级

目前不支持从 Aurora MySQL 版本 2.12.0 或 2.12.1（其中写入器数据库实例类为 db.r6i.32xlarge）运行就地升级。在这种情况下，预检查会返回错误。要让升级继续进行，可以将数据库实例类更改为 db.r6i.24xlarge 或更小。或者，可以升级到 Aurora MySQL 版本 2.12.2 或更高版本，其中 db.r6i.32xlarge 支持就地升级到 Aurora MySQL 版本 3。

输出示例：

```
{
```

```
  "id": "auroraUpgradeCheckForInstanceLimit",
  "title": "Checks if upgrade is supported on the current instance class",
  "status": "OK",
  "description": "Upgrade from Aurora Version 2.12.0 and 2.12.1 may fail for 32.xl
 and above instance class.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "all",
        "description": "Upgrade is not supported on this instance size for Aurora
 MySql Version 2.12.1. Before upgrading to Aurora MySql 3, please consider either:
 1. Changing the instance class to 24.xl or lower. -or- 2. Upgrading to patch
 version 2.12.2 or higher."
      }
  ]
},
```

预检查会返回错误，因为写入器数据库实例使用的是 db.r6i.32xlarge 实例类，并且在 Aurora MySQL 版本 2.12.1 上运行。

auroraUpgradeCheckForInternalUsers

预检查级别：错误

检查是否有 8.0 内部用户

此项预检查仅适用于 Aurora MySQL 版本 3.03.0 及更低版本。如果您在此项预检查中遇到错误，请升级到 Aurora MySQL 版本 3.04 或更高版本。

输出示例：

```
{
  "id": "auroraUpgradeCheckForInternalUsers",
  "title": "Check for 8.0 internal users.",
  "status": "OK",
  "detectedProblems": []
}
```

auroraUpgradeCheckForMasterUser

预检查级别：错误

检查 RDS 主用户是否存在

MySQL 8 添加了支持 role 和 dynamic privileges 的新权限模型，使权限管理变得更简单、更精细。作为此次变更的一部分，Aurora MySQL 引入了新的 `rds_superuser_role`，从 Aurora MySQL 版本 2 升级到版本 3 时，会自动向数据库的主用户授予此角色。

有关 Aurora MySQL 中分配给主用户的角色和权限的更多信息，请参阅主用户账户权限。有关 Aurora MySQL 版本 3 中基于角色的权限模型的更多信息，请参阅基于角色的权限模型。

此项预检查验证数据库中是否存在主用户。如果主用户不存在，则预检查失败。要让升级继续进行，请通过重置主用户密码或手动创建用户来重新创建主用户。然后重试升级。有关重置主用户密码的更多信息，请参阅更改数据库主用户的密码。

输出示例：

```
{
  "id": "auroraUpgradeCheckForMasterUser",
  "title": "Check if master user exists",
  "status": "OK",
  "description": "Throws error if master user has been dropped!",
  "documentationLink": "https://docs.aws.amazon.com/AmazonRDS/latest/
AuroraUserGuide/UsingWithRDS.MasterAccounts.html",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "all",
        "description": "Your Master User on host '%' has been dropped. To proceed
 with the upgrade, recreate the master user `reinvent` on default host '%'"
      }
  ]
}
```

重置主用户密码后，预检查将通过，可以重试升级。

以下示例使用 AWS CLI 来重置密码。密码更改会立即应用。

```
aws rds modify-db-cluster \
    --db-cluster-identifier my-db-cluster \
    --master-user-password my-new-password
```

然后，预检查成功。

```
{
```

```
  "id": "auroraUpgradeCheckForMasterUser",
  title": "Check if master user exists",
  "status": "OK",
  "detectedProblems": []
}
```

auroraUpgradeCheckForPrefixIndexOnGeometryColumns

预检查级别：错误

检查前缀索引上是否有几何列

从 MySQL 8.0.12 开始，无法再使用 GEOMETRY 数据类型在列上创建 prefixed 索引。有关更多信息，请参阅 WL#11808。

如果存在任何此类索引，则升级将失败。要解决此问题，请删除预检查失败中提及的表上的带前缀 GEOMETRY 索引。

输出示例：

```
{
  "id": "auroraUpgradeCheckForPrefixIndexOnGeometryColumns",
  "title": "Check for geometry columns on prefix indexs",
  "status": "OK",
  "description": "Consider dropping the prefix Indexes of geometry columns and
 restart the upgrade.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.geom_index_prefix",
        "description": "Table `test`.`geom_index_prefix` has an index `LatLon` on
 geometry column/s. Mysql 8.0 does not support this type of index on a geometry
 column https://dev.mysql.com/worklog/task/?id=11808. To upgrade to MySQL 8.0, Run
 'DROP INDEX `LatLon` ON `test`.`geom_index_prefix`;"
      }
  ]
}
```

预检查会报告错误，因为 test.geom_index_prefix 表中包含的索引在 GEOMETRY 列上带有前缀。

删除此索引后，预检查成功。

```
{
  "id": "auroraUpgradeCheckForPrefixIndexOnGeometryColumns",
  "title": "Check for geometry columns on prefix indexs",
  "status": "OK",
  "detectedProblems": []
}
```

auroraUpgradeCheckForSpecialCharactersInProcedures

预检查级别：错误

检查存储过程中是否存在与特殊字符相关的不一致性

在 MySQL 8.0 之前，数据库名称、表名称和其它对象对应于数据目录中的文件，即基于文件的元数据。作为升级到 MySQL 8.0 的一部分，所有数据库对象都迁移到存储在 `mysql` 架构中的新的内部数据字典表中，以支持新实现的 Atomic Data Dictionary。作为迁移存储过程的一部分，在将每个过程的过程定义和主体摄取到新的数据字典中时，都会对其进行验证。

在 MySQL 8 之前，在某些情况下，可以创建存储例程，或将包含特殊字符的过程直接插入 `mysql.proc` 表中。例如，可以创建一个存储过程，其中包含带有不合规、不换行空格字符 `\xa0` 的注释。如果遇到任何此类过程，则升级失败。

此项预检查可验证存储过程的主体和定义中是否不包含任何此类字符。要让升级继续进行，请重新创建这些存储过程，而不包含任何隐藏或特殊字符。

输出示例：

```
{
  "id": "auroraUpgradeCheckForSpecialCharactersInProcedures",
  "title": "Check for inconsistency related to special characters in stored
 procedures.",
  "status": "OK",
  "description": "Following procedure(s) has special characters inconsistency.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "information_schema.routines",
        "description": "Data Dictionary Metadata is inconsistent for the procedure
 `get_version_proc` in the database `test` due to usage of special characters in
 procedure body. To avoid that, drop and recreate the procedure without any special
 characters before proceeding with the Upgrade."
      }
```

```
  ]
}
```

预检查报告数据库集群在 `test` 数据库包含一个名为 `get_version_proc` 的过程，该过程在过程主体中包含特殊字符。

删除并重新创建该存储过程后，预检查成功，并支持升级继续进行。

```
{
  "id": "auroraUpgradeCheckForSpecialCharactersInProcedures",
  "title": "Check for inconsistency related to special characters in stored
 procedures.",
  "status": "OK",
  "detectedProblems": []
},
```

auroraUpgradeCheckForSysSchemaObjectTypeMismatch

预检查级别：错误

检查 **sys** 架构的对象类型不匹配

sys schema 是 MySQL 数据库中的一组对象和视图，有助于用户对其数据库实例进行故障排除、优化和监控。当执行从 Aurora MySQL 版本 2 到版本 3 的主要版本升级时，将重新创建 `sys` 架构视图，并将其更新为新的 Aurora MySQL 版本 3 定义。

作为升级的一部分，如果 `sys` 架构中的任何对象是使用存储引擎（INFORMATION_SCHEMA.TABLES 中的 `sys_config/BASE TABLE`）而不是视图定义的，则升级将失败。这类表可以在 `information_schema.tables` 表中找到。这不是预期的行为，但在某些情况下可能是由于用户错误造成的。

此项预检查会验证所有 `sys` 架构对象，以确保它们使用正确的表定义，并且视图不会被错误地定义为 InnoDB 或 MyISAM 表。要解决此问题，请通过重命名或删除所有返回的对象来手动修复它们。然后重试升级。

输出示例：

```
{
  "id": "auroraUpgradeCheckForSysSchemaObjectTypeMismatch",
  "title": "Check object type mismatch for sys schema.",
  "status": "OK",
  "description": "Database contains objects with type mismatch for sys schema.",
```

```
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "sys.waits_global_by_latency",
        "description": "Your object sys.waits_global_by_latency has a type mismatch.
 To fix the inconsistency we recommend to rename or remove the object before
 upgrading (use RENAME TABLE command). "
      }
  ]
}
```

预检查报告 sys 架构中的 sys.waits_global_by_latency 视图具有类型不匹配，导致无法继续升级。

登录数据库实例后，可以看到此对象被定义为 InnoDB 表，而它应该是一个视图。

```
mysql> show create table sys.waits_global_by_latency\G
*************************** 1. row ***************************
       Table: waits_global_by_latency
Create Table: CREATE TABLE `waits_global_by_latency` (
  `events` varchar(128) DEFAULT NULL,
  `total` bigint(20) unsigned DEFAULT NULL,
  `total_latency` text,
  `avg_latency` text,
  `max_latency` text
) ENGINE=InnoDB DEFAULT CHARSET=utf8
1 row in set (0.00 sec)
```

要解决这个问题，可以删除该视图并使用 correct definition 重新创建该视图，也可以对其进行重命名。在升级过程中，将使用正确的表定义自动创建该视图。

```
mysql> RENAME TABLE sys.waits_global_by_latency to sys.waits_global_by_latency_old;
Query OK, 0 rows affected (0.01 sec)
```

完成此操作后，预检查通过。

```
{
  "id": "auroraUpgradeCheckForSysSchemaObjectTypeMismatch",
  "title": "Check object type mismatch for sys schema.",
  "status": "OK",
  "detectedProblems": []
}
```

auroraUpgradeCheckForViewColumnNameLength

预检查级别：错误

检查视图中列名称的上限

MySQL 中的 maximum permitted length of a column name 为 64 个字符。在 MySQL 8.0 之前，在某些情况下，可以创建列名称大于 64 个字符的视图。如果数据库实例上存在任何此类视图，则会返回预检查错误，升级将失败。要让升级继续进行，必须重新创建有问题的视图，确保其列长度小于 64 个字符。然后重试升级。

输出示例：

```
{
  "id": "auroraUpgradeCheckForViewColumnNameLength",
  "title": "Check for upperbound limit related to column name in view.",
  "status": "OK",
  "description": "Following view(s) has column(s) with length greater than 64.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject":
 "test.colname_view_test.col2_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pa
        "description": "View `test`.`colname_view_test`has column
 `col2_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad` with
 invalid column name length. To avoid Upgrade errors, view should be altered by
 renaming the column name so that its length is not 0 and doesn't exceed 64."
      }
  ]
}
```

预检查报告 `test.colname_view_test` 视图包含的列 `col2_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad` 超过了支持的最大列长度 64 个字符。

通过查看视图定义，我们可以看到违规的列。

```
mysql> desc `test`.`colname_view_test`;
+-------------------------------------------------------------------+-------------
+------+-----+---------+-------+
| Field                                                             | Type        |
 Null | Key | Default | Extra |
```

```
+-------------------------------------------------------------------+-------------
+------+-----+---------+-------+
| col1                                                              | varchar(20) |
 YES  |     | NULL    |       |
| col2_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad | int(11)     |
 YES  |     | NULL    |       |
+-------------------------------------------------------------------+-------------
+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

要让升级继续进行，请重新创建该视图，确保列长度不超过 64 个字符。

```
mysql> drop view `test`.`colname_view_test`;
Query OK, 0 rows affected (0.01 sec)

mysql> create view `test`.`colname_view_test`(col1, col2_nopad) as select inf,
 fodcol from test;
Query OK, 0 rows affected (0.01 sec)

mysql> desc `test`.`colname_view_test`;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| col1       | varchar(20) | YES  |     | NULL    |       |
| col2_nopad | int(11)     | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

完成此操作后，预检查成功。

```
{
  "id": "auroraUpgradeCheckForViewColumnNameLength",
  "title": "Check for upperbound limit related to column name in view.",
  "status": "OK",
  "detectedProblems": []
}
```

auroraUpgradeCheckIndexLengthLimitOnTinyColumns

预检查级别：错误

检查是否有些表在微小列上定义了前缀长度大于 255 字节的索引

在 MySQL 中使用 binary data type 对某个列创建索引时，必须在索引定义中添加 prefix 长度。在 MySQL 8.0 之前，在某些情况下，可以指定比此类数据类型支持的最大大小更大的前缀长度。例如 TINYTEXT 和 TINYBLOB 列，其中允许的最大数据大小为 255 字节，但允许使用大于此值的索引前缀。有关更多信息，请参阅 MySQL 文档中的 InnoDB limits。

如果此项预检查失败，请删除违规索引，或将索引的 TINYTEXT 和 TINYBLOB 列的前缀长度减少到小于 255 字节。然后重试升级。

输出示例：

```
{
  "id": "auroraUpgradeCheckIndexLengthLimitOnTinyColumns",
  "title": "Check for the tables with indexes defined with prefix length greater
 than 255 bytes on tiny columns",
  "status": "OK",
  "description": "Prefix length of the indexes defined on tiny columns cannot
 exceed 255 bytes. With utf8mb4 char set, this limits the prefix length supported
 upto 63 characters only. A larger prefix length was allowed in MySQL5.7 using
 innodb_large_prefix parameter. This parameter is deprecated in MySQL 8.0.",
  "documentationLink": "https://dev.mysql.com/doc/refman/8.0/en/innodb-limits.html,
 https://dev.mysql.com/doc/refman/8.0/en/storage-requirements.html",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.tintxt_prefixed_index.col1",
        "description": "Index 'PRIMARY' on tinytext/tinyblob column `col1` of table
 `test.tintxt_prefixed_index` is defined with prefix length exceeding 255 bytes.
 Reduce the prefix length to <=255 bytes depending on character set used. For
 utf8mb4, it should be <=63."
      }
  ]
}
```

预检查报告表 test.tintxt_prefixed_index 存在错误，因为它的索引 PRIMARY 在 TINYTEXT 或 TINYBLOB 列上的前缀大于 255 字节。

查看此表的定义，可以看到主键在 TINYTEXT 列 col1 上的前缀为 65。由于表是使用 utf8mb4 字符集定义的，该字符集对于每个字符存储 4 个字节，因此前缀超过了 255 字节的限制。

```
mysql> show create table `test`.`tintxt_prefixed_index`\G
*************************** 1. row ***************************
       Table: tintxt_prefixed_index
```

```
Create Table: CREATE TABLE `tintxt_prefixed_index` (
  `col1` tinytext NOT NULL,
  `col2` tinytext,
  `col_id` tinytext,
  PRIMARY KEY (`col1`(65))
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 ROW_FORMAT=DYNAMIC
1 row in set (0.00 sec)
```

在使用 utf8mb4 字符集时将索引前缀修改为 63，可让升级继续进行。

```
mysql> alter table `test`.`tintxt_prefixed_index` drop PRIMARY KEY, ADD  PRIMARY KEY
 (`col1`(63));
Query OK, 0 rows affected (0.04 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

完成此操作后，预检查成功。

```
{
  "id": "auroraUpgradeCheckIndexLengthLimitOnTinyColumns",
  "title": "Check for the tables with indexes defined with prefix length greater
 than 255 bytes on tiny columns",
  "status": "OK",
  "detectedProblems": []
}
```

auroraUpgradeCheckMissingInnodbMetadataForMysqlHostTable

预检查级别：错误

检查 **mysql.host** 表缺少的 InnoDB 元数据不一致性

这是 RDS 服务执行的仅限内部的预检查。任何错误都将在升级时自动得以处理，可以安全地忽略。

如果您在此项预检查中遇到任何错误，请向 AWS Support 提交案例，来请求解决元数据不一致性。或者，可以通过执行逻辑转储，然后还原到新的 Aurora MySQL 版本 3 数据库集群来重试升级。

auroraUpgradeCheckForInvalidUtf8mb3ColumnComments

预检查级别：错误

检查是否存在无效的 utf8mb3 列注释

此项预检查识别出的表所包含的列注释具有无效的 utf8mb3 字符编码。在 MySQL 8.0 中，对元数据（包括列注释）中的字符编码应用了更严格的验证。如果任何列注释包含在 utf8mb3 字符集中无效的字符，则升级将失败。

此问题最常见的原因是列注释中存在非 BMP（基本多语言平面）字符。非 BMP 字符需要 4 字节的 UTF-8 编码（utf8mb4），但 utf8mb3 仅支持 3 字节序列，而无法表示这些字符。

要解决此问题，在尝试升级之前，必须修改列注释以移除或替换任何非 BMP 字符。您可以使用 ALTER TABLE 语句来更新列注释。

输出示例：

```
{
  "id": "auroraUpgradeCheckForInvalidUtf8mb3ColumnComments",
  "title": "Check for invalid utf8mb3 column comments.",
  "status": "OK",
  "description": "Following table(s) has/have invalid utf8mb3 comments on the
 column/columns.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.t2",
        "description": "Table test.t2 has invalid utf8mb3 comments in it's column/
columns. This is due to non-BMP characters in the comment field. To fix the
 inconsistency, we recommend you to modify comment fields to not use non-BMP
 characters and try the upgrade again."
      }
  ]
}
```

此项预检查报告 test.t2 表在一个或多个列注释中包含无效 utf8mb3 字符，特别是由于存在非 BMP 字符。

要解决此问题，您可以识别有问题的列并更新其注释。首先，检查表结构以识别带有注释的列：

```
mysql> SHOW CREATE TABLE test.t2\G
```

识别出带有问题注释的列后，使用 ALTER TABLE 语句对其进行更新。例如：

```
mysql> ALTER TABLE test.t2 MODIFY COLUMN column_name data_type COMMENT 'Updated
 comment without non-BMP characters';
```

或者，您可以完全移除注释：

```
mysql> ALTER TABLE test.t2 MODIFY COLUMN column_name data_type COMMENT '';
```

更新所有有问题的列注释后，预检查将通过，升级可以继续进行：

```
{
  "id": "auroraUpgradeCheckForInvalidUtf8mb3ColumnComments",
  "title": "Check for invalid utf8mb3 column comments.",
  "status": "OK",
  "detectedProblems": []
}
```

> **Note**
>
> 在修改列注释之前，请确保依赖于这些注释的任何应用程序代码或文档都已相应更新。如果应用程序需要非 BMP 字符，可以考虑迁移到 utf8mb4 字符集以获得更好的 Unicode 支持。

警告

以下预检查失败时会生成警告，但升级可以继续。

主题

- 报告警告的 MySQL 预检查
- 报告警告的 Aurora MySQL 预检查

报告警告的 MySQL 预检查

以下预检查来自社区 MySQL：

- defaultAuthenticationPlugin
- maxdbFlagCheck
- mysqlDollarSignNameCheck
- reservedKeywordsCheck
- utf8mb3Check

- zeroDatesCheck

defaultAuthenticationPlugin

预检查级别：警告

新的默认身份验证插件注意事项

在 MySQL 8.0 中，引入了 caching_sha2_password 身份验证插件，与已弃用的 mysql_native_password 插件相比，它提供了更安全的密码加密和更好的性能。对于 Aurora MySQL 版本 3，用于数据库用户的默认身份验证插件是 mysql_native_password 插件。

此项预检查警告称，在将来的主要版本中，此插件将被移除并会更改默认插件。请考虑在此项更改之前评估应用程序客户端和用户的兼容性。

有关更多信息，请参阅 MySQL 文档中的 caching_sha2_password compatibility issues and solutions。

输出示例：

```
{
  "id": "defaultAuthenticationPlugin",
  "title": "New default authentication plugin considerations",
  "description": "Warning: The new default authentication plugin
 'caching_sha2_password' offers more secure password hashing than previously
 used 'mysql_native_password' (and consequent improved client connection
 authentication). However, it also has compatibility implications that
 may affect existing MySQL installations. If your MySQL installation
 must serve pre-8.0 clients and you encounter compatibility issues after
 upgrading, the simplest way to address those issues is to reconfigure
 the server to revert to the previous default authentication plugin
 (mysql_native_password). For example, use these lines in the server option file:
\n\n[mysqld]\ndefault_authentication_plugin=mysql_native_password\n\nHowever, the
 setting should be viewed as temporary, not as a long term or permanent solution,
 because it causes new accounts created with the setting in effect to forego the
 improved authentication security.\nIf you are using replication please take time to
 understand how the authentication plugin changes may impact you.",
  "documentationLink": "https://dev.mysql.com/doc/refman/8.0/en/upgrading-from-
previous-series.html#upgrade-caching-sha2-password-compatibility-issues\nhttps://
dev.mysql.com/doc/refman/8.0/en/upgrading-from-previous-series.html#upgrade-caching-
sha2-password-replication"
},
```

maxdbFlagCheck

预检查级别：警告

使用过时的 **MAXDB sql_mode** 标志

在 MySQL 8.0 中，removed 了许多已弃用的 sql_mode 系统变量选项，其中一个是 MAXDB。此项预检查会检查所有当前连接的会话，以及例程和触发器，来确保没有任何会话将 sql_mode 设置为任何包含 MAXDB 的组合。

输出示例：

```
{
  "id": "maxdbFlagCheck",
  "title": "Usage of obsolete MAXDB sql_mode flag",
  "status": "OK",
  "description": "Warning: The following DB objects have the obsolete MAXDB
 option persisted for sql_mode, which will be cleared during the upgrade. It can
 potentially change the datatype DATETIME into TIMESTAMP if it is used inside
 object's definition, and this in turn can change the behavior in case of dates
 earlier than 1970 or later than 2037. If this is a concern, please redefine these
 objects so that they do not rely on the MAXDB flag before running the upgrade.",
  "documentationLink": "https://dev.mysql.com/doc/refman/8.0/en/mysql-
nutshell.html#mysql-nutshell-removals",
  "detectedProblems": [
      {
        "level": "Warning",
        "dbObject": "test.maxdb_stored_routine",
        "description": "PROCEDURE uses obsolete MAXDB sql_mode",
        "dbObjectType": "Routine"
      }
  ]
}
```

预检查报告 test.maxdb_stored_routine 例程包含不受支持的 sql_mode 选项。

登录数据库后，可以在例程定义中看到 sql_mode 包含 MAXDB。

```
 > SHOW CREATE PROCEDURE test.maxdb_stored_routine\G

*************************** 1. row ***************************
           Procedure: maxdb_stored_routine
```

```
            sql_mode:
 PIPES_AS_CONCAT,ANSI_QUOTES,IGNORE_SPACE,MAXDB,NO_KEY_OPTIONS,NO_TABLE_OPTIONS,NO_FIELD_OPT
    Create Procedure: CREATE DEFINER="msandbox"@"localhost" PROCEDURE
 "maxdb_stored_routine"()
BEGIN
    SELECT * FROM test;
END
character_set_client: utf8
collation_connection: utf8_general_ci
  Database Collation: latin1_swedish_ci
1 row in set (0.00 sec)
```

要解决此问题，请在客户端上设置正确的 sql_mode 后重新创建该过程。

> **Note**
>
> 根据 MySQL documentation，MySQL 存储在创建或更改例程时生效的 sql_mode 设置。它始终使用此设置运行例程，而不管例程启动时 sql_mode 设置如何。
> 在更改 sql_mode 之前，请参阅 MySQL 文档中的 Server SQL modes。仔细评估这样做对应用程序产生的任何潜在影响。

在不使用不受支持的 sql_mode 的情况下重新创建该过程。

```
mysql > set sql_mode='PIPES_AS_CONCAT,ANSI_QUOTES,IGNORE_SPACE';
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql > DROP PROCEDURE test.maxdb_stored_routine\G
Query OK, 0 rows affected (0.00 sec)

mysql >
mysql > DELIMITER $$
mysql >
mysql > CREATE PROCEDURE test.maxdb_stored_routine()
    -> SQL SECURITY DEFINER
    -> BEGIN
    ->     SELECT * FROM test;
    -> END$$
Query OK, 0 rows affected (0.00 sec)

mysql >
mysql > DELIMITER ;
```

```
mysql > show create procedure test.maxdb_stored_routine\G
*************************** 1. row ***************************
           Procedure: maxdb_stored_routine
            sql_mode: PIPES_AS_CONCAT,ANSI_QUOTES,IGNORE_SPACE
    Create Procedure: CREATE DEFINER="msandbox"@"localhost" PROCEDURE
 "maxdb_stored_routine"()
BEGIN
    SELECT * FROM test;
END
character_set_client: utf8
collation_connection: utf8_general_ci
  Database Collation: latin1_swedish_ci
1 row in set (0.00 sec)
```

预检查成功。

```
{
  "id": "maxdbFlagCheck",
  "title": "Usage of obsolete MAXDB sql_mode flag",
  "status": "OK",
  "detectedProblems": []
}
```

mysqlDollarSignNameCheck

预检查级别：警告

检查对象名称中是否已弃用单美元符号

从 MySQL 8.0.32 开始，弃用将美元符号 ($) 作为未加引号的标识符的第一个字符。如果有任何以 $ 为第一个字符的架构、表、视图、列或例程，则此项预检查会返回警告。虽然此警告不会阻止升级继续进行，但我们建议您尽快采取措施来解决此问题。从 MySQL 8.4 开始，任何此类标识符都将返回语法错误而不是警告。

输出示例：

```
{
  "id": "mysqlDollarSignNameCheck",
  "title": "Check for deprecated usage of single dollar signs in object names",
  "status": "OK",
  "description": "Warning: The following objects have names with deprecated usage
 of dollar sign ($) at the begining of the identifier. To correct this warning,
```

```
 ensure, that names starting with dollar sign, also end with it, similary to quotes
 ($example$). ",
  "detectedProblems": [
      {
        "level": "Warning",
        "dbObject": "test.$deprecated_syntx",
        "description": " name starts with $ sign."
      }
  ]
},
```

预检查会报告警告，因为 `test` 架构中的 `$deprecated_syntx` 表包含 `$` 作为第一个字符。

reservedKeywordsCheck

预检查级别：警告

使用名称与新的保留关键字冲突的数据库对象

此检查与 routineSyntaxCheck 类似，因为它会检查是否使用了名称与新的保留关键字冲突的数据库对象。虽然它们不会阻止升级，但您需要仔细评估警告。

输出示例：

将前面的示例用于名为 `except` 的表，预检查返回警告：

```
{
  "id": "reservedKeywordsCheck",
  "title": "Usage of db objects with names conflicting with new reserved keywords",
  "status": "OK",
  "description": "Warning: The following objects have names that conflict with
 new reserved keywords. Ensure queries sent by your applications use `quotes` when
 referring to them or they will result in errors.",
  "documentationLink": "https://dev.mysql.com/doc/refman/en/keywords.html",
  "detectedProblems": [
      {
        "level": "Warning",
        "dbObject": "test.except",
        "description": "Table name",
        "dbObjectType": "Table"
      }
  ]
}
```

此警告让您了解可能需要查看一些应用程序查询。如果应用程序查询未正确地 escaping string literals，则在升级到 MySQL 8.0 之后，这些查询可能会遇到错误。检查应用程序以进行确认，并针对在版本 3 上运行的 Aurora MySQL 数据库集群的克隆或快照进行测试。

升级后将失败的未转义的应用程序查询示例：

```
SELECT * FROM escape;
```

升级后将成功执行的经正确转义的应用程序查询示例：

```
SELECT * FROM 'escape';
```

utf8mb3Check

预检查级别：警告

使用 **`utf8mb3`** 字符集

在 MySQL 8.0 中，`utf8mb3` 字符集已弃用，并会在将来的 MySQL 主要版本中移除。实现此项预检查是为了在检测到任何使用此字符集的数据库对象时发出警告。虽然这不会阻止升级继续进行，但我们强烈建议您考虑将表迁移到 `utf8mb4` 字符集，这是 MySQL 8.0 中的默认字符集。有关 utf8mb3 和 utf8mb4 的更多信息，请参阅 MySQL 文档中的 Converting between 3-byte and 4-byte Unicode character sets。

输出示例：

```
{
  "id": "utf8mb3",
  "title": "Usage of utf8mb3 charset",
  "status": "OK",
  "description": "Warning: The following objects use the deprecated utf8mb3
 character set. It is recommended to convert them to use utf8mb4 instead, for
 improved Unicode support. The utf8mb3 character is subject to removal in the
 future.",
  "documentationLink": "https://dev.mysql.com/doc/refman/8.0/en/charset-unicode-
utf8mb3.html",
  "detectedProblems": [
      {
        "level": "Warning",
        "dbObject": "test.t1.col1",
        "description": "column's default character set: utf8",
        "dbObjectType": "Column"
```

```
      },
      {
        "level": "Warning",
        "dbObject": "test.t1.col2",
        "description": "column's default character set: utf8",
        "dbObjectType": "Column"
      }
  ]
}
```

要解决此问题，您需要重建引用的对象和表。有关在执行此操作之前的更多信息和先决条件，请参阅 MySQL 文档中的 Converting between 3-byte and 4-byte Unicode character sets。

zeroDatesCheck

预检查级别：警告

日期、日期时间和时间戳值为零

MySQL 现在对在日期、日期时间和时间戳列中使用零值实施更严格的规则。我们建议您将 NO_ZERO_IN_DATE 和 NO_ZERO_DATE SQL 模式与 strict 模式结合使用，因为在将来的 MySQL 版本中，它们将与 strict 模式合并。

如果在运行预检查时，任何数据库连接的 sql_mode 设置不包括这些模式，则会在预检查中发出警告。用户可能仍然能够插入包含零值的日期、日期时间和时间戳值。但是，我们强烈建议将任何零值替换为有效值，因为其行为将来可能会发生变化，并且这些零值可能无法正常发挥作用。由于这是一个警告，因此不会阻止升级，但我们建议您开始计划采取措施。

输出示例：

```
{
  "id": "zeroDatesCheck",
  "title": "Zero Date, Datetime, and Timestamp values",
  "status": "OK",
  "description": "Warning: By default zero date/datetime/timestamp values are no
 longer allowed in MySQL, as of 5.7.8 NO_ZERO_IN_DATE and NO_ZERO_DATE are included
 in SQL_MODE by default. These modes should be used with strict mode as they will be
 merged with strict mode in a future release. If you do not include these modes in
 your SQL_MODE setting, you are able to insert date/datetime/timestamp values that
 contain zeros. It is strongly advised to replace zero values with valid ones, as
 they may not work correctly in the future.",
  "documentationLink": "https://lefred.be/content/mysql-8-0-and-wrong-dates/",
  "detectedProblems": [
```

```
      {
        "level": "Warning",
        "dbObject": "global.sql_mode",
        "description": "does not contain either NO_ZERO_DATE or NO_ZERO_IN_DATE
 which allows insertion of zero dates"
      },
      {
        "level": "Warning",
        "dbObject": "session.sql_mode",
        "description": " of 10 session(s) does not contain either NO_ZERO_DATE or
 NO_ZERO_IN_DATE which allows insertion of zero dates"
      }
  ]
}
```

报告警告的 Aurora MySQL 预检查

以下预检查特定于 Aurora MySQL：

- auroraUpgradeCheckForRollbackSegmentHistoryLength
- auroraUpgradeCheckForUncommittedRowModifications

auroraUpgradeCheckForRollbackSegmentHistoryLength

预检查级别：警告

检查集群的回滚段历史记录列表长度是否很高

如在 auroraUpgradeCheckForIncompleteXATransactions 中所提到的那样，在运行主要版本升级过程时，务必对 Aurora MySQL 版本 2 数据库实例执行 clean shutdown。这可以确保所有事务都已提交或回滚，并且 InnoDB 已清除所有撤消日志记录。

如果数据库集群具有较高的回滚段历史记录列表长度（HLL），则可能会延长 InnoDB 完成其清除撤消日志记录所需的时间量，从而导致主要版本升级过程中的停机时间延长。如果预检查检测到数据库集群上的 HLL 很高，则会发出警告。虽然这不会阻止升级继续进行，但建议您密切监控数据库集群上的 HLL。将其保持在较低水平，可以减少主要版本升级期间所需的停机时间。有关监控 HLL 的更多信息，请参阅 InnoDB 历史记录列表长度显著增加。

输出示例：

```
{
```

```
  "id": "auroraUpgradeCheckForRollbackSegmentHistoryLength",
  "title": "Checks if the rollback segment history length for the cluster is high",
  "status": "OK",
  "description": "Rollback Segment History length is greater than 1M. Upgrade may
 take longer time.",
  "detectedProblems": [
      {
        "level": "Warning",
        "dbObject": "information_schema.innodb_metrics",
        "description": "The InnoDB undo history list length('trx_rseg_history_len')
 is 82989114. Upgrade may take longer due to purging of undo information for old row
 versions."
      }
  ]
}
```

预检查返回警告，因为它检测到数据库集群上的 InnoDB 撤消 HLL 很高（82989114）。尽管升级继续进行，但根据要清除的撤消量，可能会延长升级过程中所需的停机时间。

我们建议您在生产环境中运行升级之前，先调查数据库集群上的未结事务，以确保 HLL 保持为可管理的大小。

auroraUpgradeCheckForUncommittedRowModifications

预检查级别：警告

检查是否有许多未提交的行修改

如在 auroraUpgradeCheckForIncompleteXATransactions 中所提到的那样，在运行主要版本升级过程时，务必对 Aurora MySQL 版本 2 数据库实例执行 clean shutdown。这可以确保所有事务都已提交或回滚，并且 InnoDB 已清除所有撤消日志记录。

如果数据库集群具有修改了大量行的事务，则它可能会延长 InnoDB 完成回滚此事务（作为干净关闭过程的一部分）所需的时间量。如果预检查在数据库集群上发现长时间运行的事务且具有大量的已修改行，则会发出警告。虽然这不会阻止升级继续进行，但建议您密切监控数据库集群上活动事务的大小。将其保持在较低水平，可以减少主要版本升级期间所需的停机时间。

输出示例：

```
{
  "id": "auroraUpgradeCheckForUncommittedRowModifications",
  "title": "Checks if there are many uncommitted modifications to rows",
  "status": "OK",
```

```
  "description": "Database contains uncommitted row changes greater than 10M.
 Upgrade may take longer time.",
  "detectedProblems": [
      {
        "level": "Warning",
        "dbObject": "information_schema.innodb_trx",
        "description": "The database contains 11000000 uncommitted row change(s) in
 1 transaction(s). Upgrade may take longer due to transaction rollback."
      }
  ]
},
```

预检查报告称，数据库集群包含一个事务，其中有 11000000 个未提交的行更改将需要在干净关闭过程中回滚。升级将继续进行，但为了减少升级过程中的停机时间，建议您在生产集群上运行升级之前对此进行监控和调查。

要查看写入器数据库实例上的活动事务，可以使用 information_schema.innodb_trx 表。以下针对写入器数据库实例的查询显示了数据库集群的当前事务、运行时间、状态和修改的行。

```
# Example of uncommitted transaction
mysql> SELECT trx_started,
       TIME_TO_SEC(TIMEDIFF(now(), trx_started)) AS seconds_trx_has_been_running,
       trx_mysql_thread_id AS show_processlist_connection_id,
       trx_id,
       trx_state,
       trx_rows_modified AS rows_modified
FROM information_schema.innodb_trx;
+---------------------+-----------------------------
+---------------------------------+----------+-----------+---------------+
| trx_started         | seconds_trx_has_been_running |
 show_processlist_connection_id | trx_id   | trx_state | rows_modified |
+---------------------+-----------------------------
+---------------------------------+----------+-----------+---------------+
| 2024-08-12 18:32:52 |                           1592 |
 20041 | 52866130 | RUNNING   |      11000000 |
+---------------------+-----------------------------
+---------------------------------+----------+-----------+---------------+
1 row in set (0.01 sec)

# Example of transaction rolling back
mysql> SELECT trx_started,
       TIME_TO_SEC(TIMEDIFF(now(), trx_started)) AS seconds_trx_has_been_running,
       trx_mysql_thread_id AS show_processlist_connection_id,
```

```
       trx_id,
       trx_state,
       trx_rows_modified AS rows_modified
FROM information_schema.innodb_trx;
+---------------------+-----------------------------
+---------------------------------+----------+--------------+---------------+
| trx_started         | seconds_trx_has_been_running |
 show_processlist_connection_id | trx_id   | trx_state    | rows_modified |
+---------------------+-----------------------------
+---------------------------------+----------+--------------+---------------+
| 2024-08-12 18:32:52 |                          1719 |
 20041 | 52866130 | ROLLING BACK |      10680479 |
+---------------------+-----------------------------
+---------------------------------+----------+--------------+---------------+
1 row in set (0.01 sec)
```

事务提交或回滚后，预检查将不再返回警告。在回滚任何大型事务之前，请查阅 MySQL 文档和咨询应用程序团队，因为回滚可能需要一些时间才能完成，具体取决于事务大小。

```
{
  "id": "auroraUpgradeCheckForUncommittedRowModifications",
  "title": "Checks if there are many uncommitted modifications to rows",
  "status": "OK",
  "detectedProblems": []
},
```

有关优化 InnoDB 事务管理以及在 MySQL 数据库实例上运行和回滚大型事务的潜在影响的更多信息，请参阅 MySQL 文档中的 Optimizing InnoDB transaction management。

版权声明

以下预检查失败时会生成通知，但升级可以继续。

sqlModeFlagCheck

预检查级别：通知

使用过时的 **sql_mode** 标志

除了 MAXDB 外，还 removed 许多其它 sql_mode 选项：DB2、MSSQL、MYSQL323、MYSQL40、ORACLE、POSTGRESQL、NO_FIELD_OPTIONS、NO_KEY_O
和 NO_TABLE_OPTIONS。从 MySQL 8.0 开始，这些值都不能分配给 sql_mode 系统变量。如

果此项预检查发现任何使用这些 sql_mode 设置的处于打开状态的会话，请确保更新您的数据库实例和数据库集群参数组以及客户端应用程序和配置来将其禁用。有关更多信息，请参阅 MySQL documentation。

输出示例：

```
{
  "id": "sqlModeFlagCheck",
  "title": "Usage of obsolete sql_mode flags",
  "status": "OK",
  "detectedProblems": []
}
```

要解决其中任何预检查失败，请参阅 maxdbFlagCheck。

错误、警告或通知

根据预检查输出，以下预检查可能会返回错误、警告或通知。

checkTableOutput

预检查级别：错误、警告或通知

check table x for upgrade 命令报告的问题

在开始升级到 Aurora MySQL 版本 3 之前，check table for upgrade 会在数据库集群中用户架构的每个表上运行。此项预检查与 checkTableMysqlSchema 不一样。

check table for upgrade 命令检查表中是否存在升级到较新 MySQL 版本期间可能出现的任何潜在问题。在尝试升级之前运行此命令，有助于提前确定和解决任何不兼容性，从而使实际升级过程更加顺畅。

此命令对每个表执行各种检查，例如：

- 验证表结构和元数据是否与目标 MySQL 版本兼容
- 检查表是否使用了任何已弃用或已移除的功能
- 确保表可以正确地升级而不会丢失数据

与其它预检查不同，此项预检查可以根据 check table 输出返回错误、警告或通知。如果此项预检查返回了任何表，请在启动升级之前，仔细查看这些表以及返回代码和消息。有关更多信息，请参阅 MySQL 文档中的 CHECK TABLE statement。

在此，我们提供了一个错误示例和一个警告示例。

错误示例：

```
{
  "id": "checkTableOutput",
  "title": "Issues reported by 'check table x for upgrade' command",
  "status": "OK",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.parent",
        "description": "Table 'test.parent' doesn't exist"
      }
  ]
},
```

预检查报告了一个错误，即 test.parent 表不存在。

写入器数据库实例的 mysql-error.log 文件显示存在外键错误。

```
2024-08-13T15:32:10.676893Z 62 [Warning] InnoDB: Load table `test`.`parent` failed,
 the table has missing foreign key indexes. Turn off 'foreign_key_checks' and try
 again.
2024-08-13T15:32:10.676905Z 62 [Warning] InnoDB: Cannot open table test/parent from
 the internal data dictionary of InnoDB though the .frm file for the table exists.
Please refer to http://dev.mysql.com/doc/refman/5.7/en/innodb-troubleshooting.html
 for how to resolve the issue.
```

登录到写入器数据库实例并运行 show engine innodb status\G，来获取有关外键错误的更多信息。

```
mysql> show engine innodb status\G
*************************** 1. row ***************************
  Type: InnoDB
  Name:
Status:
=====================================
2024-08-13 15:33:33 0x14ef7b8a1700 INNODB MONITOR OUTPUT
=====================================
.
.
```

```
.
------------------------
LATEST FOREIGN KEY ERROR
------------------------
2024-08-13 15:32:10 0x14ef6dbbb700 Error in foreign key constraint of table test/
child:
there is no index in referenced table which would contain
the columns as the first columns, or the data types in the
referenced table do not match the ones in table. Constraint:
,
  CONSTRAINT `fk_pname` FOREIGN KEY (`p_name`) REFERENCES `parent` (`name`)
The index in the foreign key in table is p_name_idx
Please refer to http://dev.mysql.com/doc/refman/5.7/en/innodb-foreign-key-
constraints.html for correct foreign key definition.
.
.
```

LATEST FOREIGN KEY ERROR 消息报告称，test.child 表中引用 test.parent 表的 fk_pname 外键约束缺少索引或数据类型不匹配。有关 foreign key constraints 的 MySQL 文档指出，外键中引用的列必须具有关联的索引，并且父/子列必须使用相同的数据类型。

要验证这是否与缺失索引或数据类型不匹配有关，请登录数据库，并通过暂时禁用会话变量 foreign_key_checks 来检查表定义。这样做之后，可以看到这个有问题的子约束 (fk_pname) 使用 p_name varchar(20) CHARACTER SET latin1 DEFAULT NULL 来引用父表 name varchar(20) NOT NULL。父表使用 DEFAULT CHARSET=utf8，但子表的 p_name 列使用 latin1，因此引发数据类型不匹配错误。

```
mysql> show create table parent\G
ERROR 1146 (42S02): Table 'test.parent' doesn't exist

mysql> show create table child\G
*************************** 1. row ***************************
       Table: child
Create Table: CREATE TABLE `child` (
  `id` int(11) NOT NULL,
  `p_name` varchar(20) CHARACTER SET latin1 DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `p_name_idx` (`p_name`),
  CONSTRAINT `fk_pname` FOREIGN KEY (`p_name`) REFERENCES `parent` (`name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
1 row in set (0.00 sec)
```

```
mysql> set foreign_key_checks=0;
Query OK, 0 rows affected (0.00 sec)

mysql> show create table parent\G
*************************** 1. row ***************************
       Table: parent
Create Table: CREATE TABLE `parent` (
  `name` varchar(20) NOT NULL,
  PRIMARY KEY (`name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
1 row in set (0.00 sec)

mysql> show create table child\G
*************************** 1. row ***************************
       Table: child
Create Table: CREATE TABLE `child` (
  `id` int(11) NOT NULL,
  `p_name` varchar(20) CHARACTER SET latin1 DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `p_name_idx` (`p_name`),
  CONSTRAINT `fk_pname` FOREIGN KEY (`p_name`) REFERENCES `parent` (`name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
1 row in set (0.00 sec)
```

要解决这个问题，可以将子表更改为与父表使用相同的字符集，也可以将父表更改为与子表使用相同的字符集。在这里，由于子表在 p_name 列定义中显式使用 latin1，因此我们运行 ALTER TABLE 来将字符集修改为 utf8。

```
mysql> alter table child modify p_name varchar(20) character set utf8 DEFAULT NULL;
Query OK, 0 rows affected (0.06 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> flush tables;
Query OK, 0 rows affected (0.01 sec)
```

完成此操作后，预检查通过，升级可以继续进行。

警告示例：

```
{
  "id": "checkTableOutput",
  "title": "Issues reported by 'check table x for upgrade' command",
```

```
  "status": "OK",
  "detectedProblems": [
      {
        "level": "Warning",
        "dbObject": "test.orders",
        "description": "Trigger test.orders.delete_audit_trigg does not have CREATED
 attribute."
      }
  ]
}
```

预检查对于 `test.orders` 表上的 `delete_audit_trigg` 触发器报告了警告，因为该触发器没有 CREATED 属性。根据 MySQL 文档中的 Checking version compatibility，此消息是信息性的，是针对在 MySQL 5.7.2 之前创建的触发器显示的。

因为这是一个警告，所以它不会阻止升级继续进行。但是，如果您希望解决此问题，可以重新创建这个有问题的触发器，预检查将成功而没有警告。

```
{
  "id": "checkTableOutput",
  "title": "Issues reported by 'check table x for upgrade' command",
  "status": "OK",
  "detectedProblems": []
},
```

如何执行就地升级

我们建议您查看 Aurora MySQL 主要版本就地升级的工作原理中的背景材料。

按为 Aurora MySQL 集群计划主要版本升级中所述执行任何升级前的计划和测试。

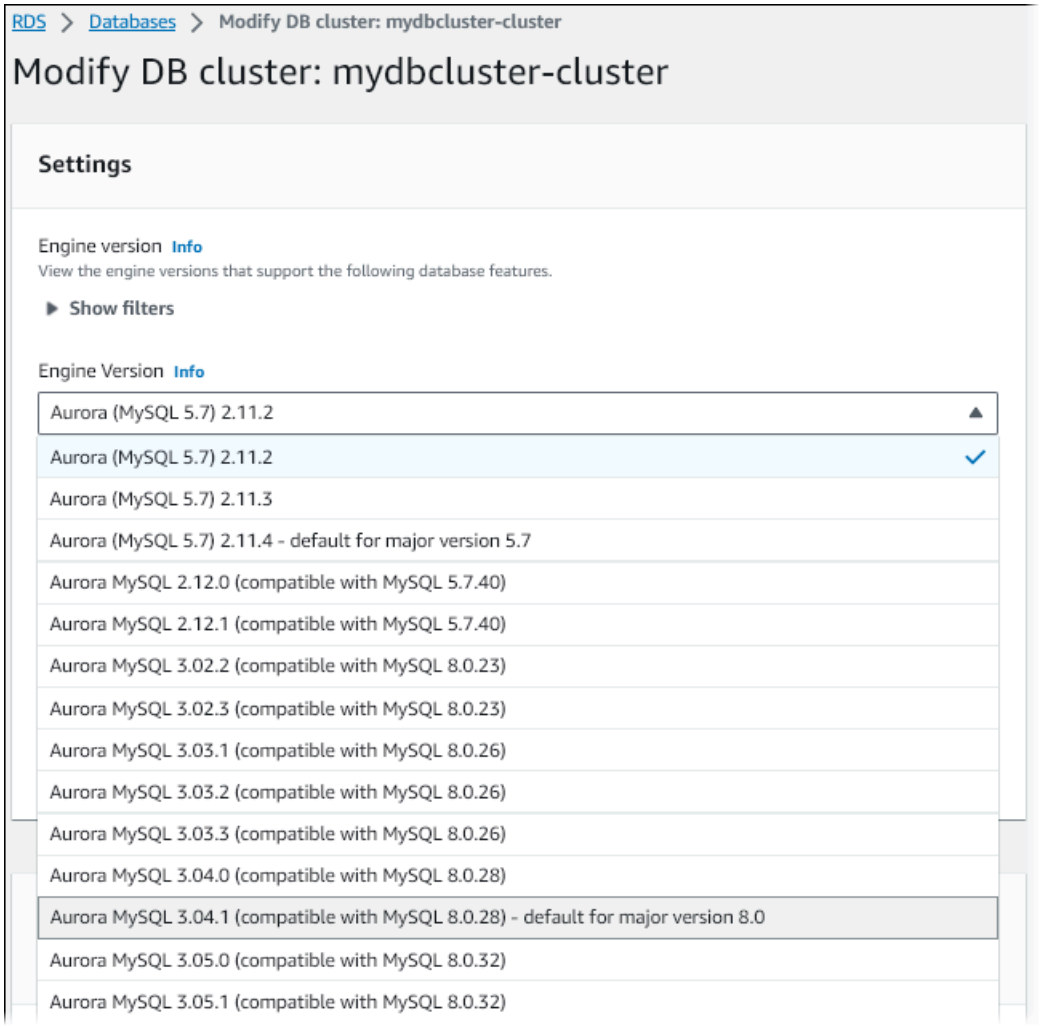
控制台

以下示例将 `mydbcluster-cluster` 数据库集群升级到 Aurora MySQL 版本 3.04.1。

要升级 Aurora MySQL 数据库集群的主要版本

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 如果您将自定义参数组用于原始数据库集群，请创建与新的主要版本兼容的相应参数组。对这个新参数组中的配置参数进行任何必要的调整。有关更多信息，请参阅“就地升级如何影响集群的参数组”。
3. 在导航窗格中，选择 Databases (数据库)。
4. 在列表中，选择您要修改的数据库集群。
5. 选择 Modify (修改)。
6. 对于 Version（版本），选择新的 Aurora MySQL 主要版本。

   我们通常建议使用主要版本的最新次要版本。在这里，我们选择当前的默认版本。



7. 选择 Continue (继续)。
8. 在下一页上，指定何时执行升级。选择 During the next scheduled maintenance window（在下一个计划的维护时段内）或 Immediately（立即）。
9. （可选）在升级过程中定期检查 RDS 控制台中的 Events（事件）页面。这样做可以帮助您监控升级进度并识别问题。如果升级遇到任何问题，请参阅Aurora MySQL 就地升级的故障排除以了解要采取的步骤。
10. 如果您在此过程开始时创建了一个新的参数组，请将自定义参数组与升级的集群关联起来。有关更多信息，请参阅 就地升级如何影响集群的参数组。

> **Note**
>
> 执行此步骤需要您再次重新启动集群以应用新的参数组。

11. （可选）完成任何升级后测试后，请删除升级开始时 Aurora 创建的手动快照。

AWS CLI

要升级 Aurora MySQL 数据库集群的主要版本，请结合使用 AWS CLI modify-db-cluster 命令与以下所需的参数：

- `--db-cluster-identifier`
- `--engine-version`
- `--allow-major-version-upgrade`
- `--apply-immediately` 或者 `--no-apply-immediately`

如果您的集群使用任何自定义参数组，则还要包含以下一个或两个选项：

- `--db-cluster-parameter-group-name`，如果集群使用自定义集群参数组
- `--db-instance-parameter-group-name`，如果集群中的任何实例使用自定义数据库参数组

以下示例将 `sample-cluster` 数据库集群升级到 Aurora MySQL 版本 3.04.1。升级会立即进行，而不是等待下一个维护时段。

Example

对于 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster \
          --db-cluster-identifier sample-cluster \
          --engine-version 8.0.mysql_aurora.3.04.1 \
          --allow-major-version-upgrade \
          --apply-immediately
```

对于 Windows：

```
aws rds modify-db-cluster ^
          --db-cluster-identifier sample-cluster ^
          --engine-version 8.0.mysql_aurora.3.04.1 ^
          --allow-major-version-upgrade ^
          --apply-immediately
```

您可以将其他 CLI 命令与 `modify-db-cluster` 结合使用，以创建执行和验证升级的自动端到端流程。有关更多信息以及示例，请参阅 Aurora MySQL 就地升级教程。

> **Note**
>
> 如果您的集群属于 Aurora 全局数据库的一部分，则就地升级程序会略有不同。您可以调用 modify-global-cluster 命令操作而不是 `modify-db-cluster`。有关更多信息，请参阅“全局数据库的就地主要版本升级”。

RDS API

要升级 Aurora MySQL 数据库集群的主要版本，请结合使用 RDS API 操作 ModifyDBCluster 与以下所需的参数：

- `DBClusterIdentifier`
- `Engine`
- `EngineVersion`
- `AllowMajorVersionUpgrade`
- `ApplyImmediately`（设置为 `true` 或 `false`）

> **Note**
>
> 如果您的集群属于 Aurora 全局数据库的一部分，则就地升级程序会略有不同。您将调用 ModifyGlobalCluster 操作而不是 `ModifyDBCluster`。有关更多信息，请参阅“全局数据库的就地主要版本升级”。

就地升级如何影响集群的参数组

对于与 MySQL 5.7 或 8.0 兼容的集群，Aurora 参数组具有不同的配置设置集。执行就地升级时，升级后的集群及其所有实例必须使用相应的集群和实例参数组：

您的集群和实例可能使用与 5.7 兼容的原定设置参数组。如果是这样，则升级后的集群和实例将以与 8.0 兼容的原定设置参数组开始。如果您的集群和实例使用任何自定义参数组，则确保创建相应的与 8.0 兼容的参数组。此外，请确保在升级过程中指定这些参数组。

> **Note**
>
> 对于大多数参数设置，您可以在两个点选择自定义参数组。也即，在您创建集群或稍后将参数组与集群关联时。

但是，如果您将非原定设置设置用于 lower_case_table_names 参数，则必须提前使用此设置来设置自定义参数组。然后，在执行快照还原操作以创建集群时指定参数组。创建集群后，lower_case_table_names 参数的任何更改不会产生任何影响。
我们建议您在从 Aurora MySQL 版本 2 升级到版本 3 时对 lower_case_table_names 使用相同的设置。
使用基于 Aurora MySQL 的 Aurora 全局数据库时，如果开启了 lower_case_table_names 参数，则无法执行从 Aurora MySQL 版本 2 到版本 3 的就地升级。有关可以使用的方法的更多信息，请参阅主要版本升级。

Important

如果在升级过程中指定了任何自定义参数组，请确保在升级完成后手动重启集群。这样做会使集群开始使用您的自定义参数设置。

Aurora MySQL 版本之间的集群属性更改

当从 Aurora MySQL 版本 2 升级到版本 3 时，请确保检查用于设置或管理 Aurora MySQL 集群和数据库实例的任何应用程序或脚本。

此外，请更改操作参数组的代码，以考虑到原定设置参数组名称对于 5.7 和 8.0 兼容的集群各不相同这一事实。Aurora MySQL 版本 2 和 3 集群的原定设置参数组名称分别为 default.aurora-mysql5.7 和 default.aurora-mysql8.0。

例如，升级之前，您可能有适用于您的集群的类似以下内容的代码。

```
# Check the default parameter values for MySQL 5.7-compatible clusters.
aws rds describe-db-parameters --db-parameter-group-name default.aurora-mysql5.7 --
region us-east-1
```

升级集群的主要版本后，请按如下方式修改该代码。

```
# Check the default parameter values for MySQL 8.0-compatible clusters.
aws rds describe-db-parameters --db-parameter-group-name default.aurora-mysql8.0 --
region us-east-1
```

全局数据库的就地主要版本升级

对于 Aurora Global Database，您可升级全局数据库集群。Aurora 会同时自动升级所有集群，并确保所有集群运行相同的引擎版本。此要求是因为对系统表、数据文件格式等所做的任何更改都会自动复制到所有辅助集群。

按照Aurora MySQL 主要版本就地升级的工作原理中的说明进行操作。指定要升级的内容时，请确保选择全局数据库集群，而不是其包含的集群之一。

如果您使用 AWS Management Console，请选择具有角色 Global database（全局数据库）的项目。

	DB identifier	Role	Engine	Engine version
◉	global-cluster	Global database	Aurora MySQL	5.7.mysql_aurora.2.09.2
○	cluster1	Primary cluster	Aurora MySQL	5.7.mysql_aurora.2.09.2
○	dbinstance-1	Writer instance	Aurora MySQL	5.7.mysql_aurora.2.09.2
○	cluster-2	Secondary cluster	Aurora MySQL	5.7.mysql_aurora.2.09.2
○	dbinstance-2	Reader instance	Aurora MySQL	5.7.mysql_aurora.2.09.2

如果您使用 AWS CLI 或 RDS API，请通过调用 modify-global-cluster 命令或 ModifyGlobalCluster 操作来启动升级过程。您可以使用其中一个操作来代替 `modify-db-cluster` 或 `ModifyDBCluster`。

> **Note**
>
> 在对该 Aurora 全局数据库执行主要版本升级时，无法为全局数据库集群指定自定义参数组。在全局集群的每个区域中创建自定义参数组。然后，在升级后手动将它们应用于区域集群。

要使用 AWS CLI 升级 Aurora MySQL 全局数据库集群的主要版本，请结合使用 modify-global-cluster 命令与以下所需的参数：

- `--global-cluster-identifier`
- `--engine aurora-mysql`
- `--engine-version`
- `--allow-major-version-upgrade`

以下示例将全局数据库集群升级到 Aurora MySQL 版本 2.10.2。

Example

对于 Linux、macOS 或 Unix：

```
aws rds modify-global-cluster \
          --global-cluster-identifier global_cluster_identifier \
          --engine aurora-mysql \
          --engine-version 5.7.mysql_aurora.2.10.2 \
          --allow-major-version-upgrade
```

对于 Windows：

```
aws rds modify-global-cluster ^
          --global-cluster-identifier global_cluster_identifier ^
          --engine aurora-mysql ^
          --engine-version 5.7.mysql_aurora.2.10.2 ^
          --allow-major-version-upgrade
```

回溯注意事项

如果您升级的集群启用了回溯功能，您无法将升级的集群回溯到升级之前的时间。

Aurora MySQL 就地升级教程

以下 Linux 示例展示了如何使用 AWS CLI 执行就地升级过程的一般步骤。

第一个示例创建了运行 Aurora MySQL 版本 2.x 的 Aurora 数据库集群。该集群包括写入器数据库实例和读取器数据库实例。`wait db-instance-available` 命令会暂停，直到写入器数据库实例可用。此时，集群做好升级准备。

```
aws rds create-db-cluster --db-cluster-identifier mynewdbcluster --engine aurora-mysql
 \
  --db-cluster-version 5.7.mysql_aurora.2.11.2
...
aws rds create-db-instance --db-instance-identifier mynewdbcluster-instance1 \
  --db-cluster-identifier mynewdbcluster --db-instance-class db.t4g.medium --engine
 aurora-mysql
...
aws rds wait db-instance-available --db-instance-identifier mynewdbcluster-instance1
```

您可以将集群升级到的 Aurora MySQL 3.x 版本取决于集群当前正在运行的 2.x 版本和集群所在的 AWS 区域。第一个命令（带 `--output text`）只显示可用的目标版本。第二个命令显示响应的完整

JSON 输出。在该响应中，您可以看到详细信息，例如您用于 engine 参数的 aurora-mysql 值。您还可以看到这样一个事实，即升级到 3.04.0 表示一次主要版本升级。

```
aws rds describe-db-clusters --db-cluster-identifier mynewdbcluster \
  --query '*[].{EngineVersion:EngineVersion}' --output text
5.7.mysql_aurora.2.11.2

aws rds describe-db-engine-versions --engine aurora-mysql --engine-version
 5.7.mysql_aurora.2.11.2 \
  --query '*[].[ValidUpgradeTarget]'
...
{
    "Engine": "aurora-mysql",
    "EngineVersion": "8.0.mysql_aurora.3.04.0",
    "Description": "Aurora MySQL 3.04.0 (compatible with MySQL 8.0.28)",
    "AutoUpgrade": false,
    "IsMajorVersionUpgrade": true,
    "SupportedEngineModes": [
        "provisioned"
    ],
    "SupportsParallelQuery": true,
    "SupportsGlobalDatabases": true,
    "SupportsBabelfish": false,
    "SupportsIntegrations": false
},
...
```

此示例表示，如果您输入的目标版本号不是集群的有效升级目标，Aurora 不会执行升级。Aurora 也不会执行主要版本升级，除非您包含 --allow-major-version-upgrade 参数。这样一来，您就不能意外执行有可能需要大量测试和更改应用程序代码的升级。

```
aws rds modify-db-cluster --db-cluster-identifier mynewdbcluster \
  --engine-version 5.7.mysql_aurora.2.09.2 --apply-immediately
An error occurred (InvalidParameterCombination) when calling the ModifyDBCluster
 operation: Cannot find upgrade target from 5.7.mysql_aurora.2.11.2 with requested
 version 5.7.mysql_aurora.2.09.2.

aws rds modify-db-cluster --db-cluster-identifier mynewdbcluster \
  --engine-version 8.0.mysql_aurora.3.04.0 --region us-east-1 --apply-immediately
An error occurred (InvalidParameterCombination) when calling the ModifyDBCluster
 operation: The AllowMajorVersionUpgrade flag must be present when upgrading to a new
 major version.
```

```
aws rds modify-db-cluster --db-cluster-identifier mynewdbcluster \
  --engine-version 8.0.mysql_aurora.3.04.0 --apply-immediately --allow-major-version-
upgrade
{
  "DBClusterIdentifier": "mynewdbcluster",
  "Status": "available",
  "Engine": "aurora-mysql",
  "EngineVersion": "5.7.mysql_aurora.2.11.2"
}
```

集群和关联数据库实例的状态需要一段时间才能更改为 upgrading。集群和数据库实例的版本号仅在升级完成后才会更改。同样，您可以使用 wait db-instance-available 命令让写入器数据库实例等到升级完成后再继续。

```
aws rds describe-db-clusters --db-cluster-identifier mynewdbcluster \
  --query '*[].[Status,EngineVersion]' --output text
upgrading 5.7.mysql_aurora.2.11.2

aws rds describe-db-instances --db-instance-identifier mynewdbcluster-instance1 \
  --query '*[].
{DBInstanceIdentifier:DBInstanceIdentifier,DBInstanceStatus:DBInstanceStatus} | [0]'
{
    "DBInstanceIdentifier": "mynewdbcluster-instance1",
    "DBInstanceStatus": "upgrading"
}

aws rds wait db-instance-available --db-instance-identifier mynewdbcluster-instance1
```

此时，集群的版本号与为升级指定的版本号匹配。

```
aws rds describe-db-clusters --db-cluster-identifier mynewdbcluster \
  --query '*[].[EngineVersion]' --output text

8.0.mysql_aurora.3.04.0
```

前面的示例通过指定 --apply-immediately 参数进行了立即升级。为了让升级在集群预计不繁忙的方便之时进行，您可以指定 --no-apply-immediately 参数。这样做可以在集群的下一个维护时段内启动升级。维护时段定义了可以启动维护操作的时间段。在维护时段内，长时间运行的操作可能无法完成。因此，即使您预计升级可能需要很长时间，也不需要定义更大的维护时段。

以下示例升级最初运行 Aurora MySQL 版本 2.11.2 的集群。在 describe-db-engine-versions 输出中，False 和 True 值表示 IsMajorVersionUpgrade 属性。您可以从版本 2.11.2 升级到一些其他 2.* 版本。这些升级不被视为主要版本升级，因此不需要就地升级。就地升级仅适用于升级到列表中显示的 3.* 版本。

```
aws rds describe-db-clusters --db-cluster-identifier mynewdbcluster \
  --query '*[].{EngineVersion:EngineVersion}' --output text
5.7.mysql_aurora.2.11.2

aws rds describe-db-engine-versions --engine aurora-mysql --engine-version
 5.7.mysql_aurora.2.10.2 \
  --query '*[].[ValidUpgradeTarget][0][0]|[*].[EngineVersion,IsMajorVersionUpgrade]'
 --output text

5.7.mysql_aurora.2.11.3 False
5.7.mysql_aurora.2.11.4 False
5.7.mysql_aurora.2.11.5 False
5.7.mysql_aurora.2.11.6 False
5.7.mysql_aurora.2.12.0 False
5.7.mysql_aurora.2.12.1 False
5.7.mysql_aurora.2.12.2 False
5.7.mysql_aurora.2.12.3 False
8.0.mysql_aurora.3.04.0 True
8.0.mysql_aurora.3.04.1 True
8.0.mysql_aurora.3.04.2 True
8.0.mysql_aurora.3.04.3 True
8.0.mysql_aurora.3.05.2 True
8.0.mysql_aurora.3.06.0 True
8.0.mysql_aurora.3.06.1 True
8.0.mysql_aurora.3.07.1 True

aws rds modify-db-cluster --db-cluster-identifier mynewdbcluster \
  --engine-version 8.0.mysql_aurora.3.04.0 --no-apply-immediately --allow-major-
version-upgrade
...
```

如果在没有指定的维护时段的情况下创建集群，Aurora 会随机选择一周中的一天。在这种情况下，modify-db-cluster 命令将在星期一提交。因此，我们将维护时段更改为星期二早上。所有时间都以 UTC 时区表示。tue:10:00-tue:10:30 时段对应于太平洋时间凌晨 2:00-2:30。维护时段的更改会立即生效。

```
aws rds describe-db-clusters --db-cluster-identifier mynewdbcluster --query '*[].
[PreferredMaintenanceWindow]'
[
    [
        "sat:08:20-sat:08:50"
    ]
]

aws rds modify-db-cluster --db-cluster-identifier mynewdbcluster --preferred-
maintenance-window tue:10:00-tue:10:30"
aws rds describe-db-clusters --db-cluster-identifier mynewdbcluster --query '*[].
[PreferredMaintenanceWindow]'
[
    [
        "tue:10:00-tue:10:30"
    ]
]
```

以下示例说明如何获取升级所生成的事件的报告。--duration 参数表示检索事件信息的分钟数。系统需要此参数，因为默认情况下，describe-events 仅返回最后一个小时的事件。

```
aws rds describe-events --source-type db-cluster --source-identifier mynewdbcluster --
duration 20160
{
    "Events": [
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "DB cluster created",
            "EventCategories": [
                "creation"
            ],
            "Date": "2022-11-17T01:24:11.093000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        },
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "Upgrade in progress: Performing online pre-upgrade checks.",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2022-11-18T22:57:08.450000+00:00",
```

```
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        },
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "Upgrade in progress: Performing offline pre-upgrade checks.",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2022-11-18T22:57:59.519000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        },
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "Upgrade in progress: Creating pre-upgrade snapshot [preupgrade-
mynewdbcluster-5-7-mysql-aurora-2-10-2-to-8-0-mysql-aurora-3-02-0-2022-11-18-22-55].",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2022-11-18T23:00:22.318000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        },
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "Upgrade in progress: Cloning volume.",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2022-11-18T23:01:45.428000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        },
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "Upgrade in progress: Purging undo records for old row versions.
 Records remaining: 164",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2022-11-18T23:02:25.141000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        },
```

```
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "Upgrade in progress: Purging undo records for old row versions.
 Records remaining: 164",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2022-11-18T23:06:23.036000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        },
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "Upgrade in progress: Upgrading database objects.",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2022-11-18T23:06:48.208000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        },
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "Database cluster major version has been upgraded",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2022-11-18T23:10:28.999000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        }
    ]
}
```

查找 Aurora MySQL 主要版本升级失败的原因

在教程中，从 Aurora MySQL 版本 2 升级到版本 3 已获成功。但是，如果升级失败，您会想知道原因。

您可以首先使用 `describe-events` AWS CLI 命令来查看数据库集群事件。此示例显示过去 10 小时 `mydbcluster` 的事件。

```
aws rds describe-events \
```

```
    --source-type db-cluster \
    --source-identifier mydbcluster \
    --duration 600
```

在本例中，我们遇到了升级预检查失败。

```
{
    "Events": [
        {
            "SourceIdentifier": "mydbcluster",
            "SourceType": "db-cluster",
            "Message": "Database cluster engine version upgrade started.",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2024-04-11T13:23:22.846000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster"
        },
        {
            "SourceIdentifier": "mydbcluster",
            "SourceType": "db-cluster",
            "Message": "Database cluster is in a state that cannot be upgraded: Upgrade
 prechecks failed. For more details, see the
             upgrade-prechecks.log file. For more information on troubleshooting the
 cause of the upgrade failure, see
             https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/
AuroraMySQL.Upgrading.Troubleshooting.html",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2024-04-11T13:23:24.373000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster"
        }
    ]
}
```

要诊断问题的确切原因，请检查写入器数据库实例的数据库日志。如果升级到 Aurora MySQL 版本 3 失败，写入器实例将包含一个名为 upgrade-prechecks.log 的日志文件。此示例说明了如何检测该日志是否存在，然后将其下载到本地文件以供检查。

```
aws rds describe-db-log-files --db-instance-identifier mydbcluster-instance \
    --query '*[].[LogFileName]' --output text
```

```
error/mysql-error-running.log
error/mysql-error-running.log.2024-04-11.20
error/mysql-error-running.log.2024-04-11.21
error/mysql-error.log
external/mysql-external.log
upgrade-prechecks.log

aws rds download-db-log-file-portion --db-instance-identifier mydbcluster-instance \
    --log-file-name upgrade-prechecks.log \
    --starting-token 0 \
    --output text >upgrade_prechecks.log
```

upgrade-prechecks.log 文件为 JSON 格式。我们使用 --output text 选项下载该文件，避免在另一个 JSON 包装器中对 JSON 输出进行编码。对于 Aurora MySQL 版本 3 升级，此日志始终包含某些信息和警告消息。如果升级失败，日志仅包含错误消息。如果升级成功，则根本不会生成日志文件。

要汇总所有错误并显示关联的对象和描述字段，您可以对 upgrade-prechecks.log 文件的内容运行命令 grep -A 2 '"level": "Error"'。这样做会显示每个错误行及其后两行。其中包含相应数据库对象的名称以及有关如何解决问题的指导。

```
$ cat upgrade-prechecks.log | grep -A 2 '"level": "Error"'

"level": "Error",
"dbObject": "problematic_upgrade.dangling_fulltext_index",
"description": "Table `problematic_upgrade.dangling_fulltext_index` contains dangling
 FULLTEXT index. Kindly recreate the table before upgrade."
```

在此示例中，您可以对有问题的表运行以下 SQL 命令来尝试修复问题，也可以重新创建没有悬挂索引的表。

```
OPTIMIZE TABLE problematic_upgrade.dangling_fulltext_index;
```

然后重试升级。

Aurora MySQL 就地升级的故障排除

可以使用以下提示帮助排查 Aurora MySQL 就地升级问题。这些提示不适用于 Aurora Serverless 数据库集群。

就地升级被取消或减慢的原因	效果	允许在维护时段内完成就地升级的解决方案
尚未修补关联的 Aurora 跨区域副本	Aurora 取消升级。	升级 Aurora 跨区域副本并重试。
集群具有处于准备状态的 XA 事务	Aurora 取消升级。	提交或回滚所有准备好的 XA 事务。
集群正在处理数据定义语言 (DDL) 语句	Aurora 取消升级。	考虑等待并在所有 DDL 语句完成后执行升级。
集群有很多行未提交的更改	升级可能需要较长时间。	升级过程将回滚未提交的更改。这种情况的指示器为 `TRX_ROWS_MODIFIED` 表中的 `INFORMATION_SCHEMA.INNODB_TRX` 值。 考虑仅在提交或回退所有大型事务后才执行升级。
集群有大量的撤消记录	升级可能需要较长时间。	即使未提交的事务不会影响大量行，也可能涉及大量数据。例如，您可能正在插入大型 BLOB。Aurora 不会自动检测或生成此类事务活动的事件。这种情况的指示器为历史记录列表长度（HLL）。升级过程将回滚未提交的更改。 您可以在 `SHOW ENGINE INNODB STATUS` SQL 命令的输出中检查 HLL，也可以直接使用以下 SQL 查询进行检查： `SELECT count FROM information_schema .innodb_metrics WHERE name = 'trx_rseg_history_len';` 还可以在 Amazon CloudWatch 中监控 `RollbackSegmentHistoryListLength` 指标。 考虑仅在 HLL 较小之后才执行升级。

就地升级被取消或减慢的原因	效果	允许在维护时段内完成就地升级的解决方案
集群正在提交大型二进制日志事务	升级可能需要较长时间。	升级过程将等到应用二进制日志更改时。在此期间，可能会启动更多事务或 DDL 语句，从而进一步减慢升级过程。 将升级过程安排在集群不忙于生成二进制日志复制更改的时间段。Aurora 不会自动检测或生成此情况的事件。
文件删除或损坏导致的模式不一致	Aurora 取消升级。	将临时表的原定设置存储引擎从 MyISAM 更改为 InnoDB。执行以下步骤： 1. 将 `default_tmp_storage_engine` 数据库参数修改为 `InnoDB`。 2. 重启数据库集群。 3. 重启后，确认 `default_tmp_storage_engine` 数据库参数设置为 `InnoDB`。使用以下命令： `show global variables like 'default_tmp_storage_engine';` 4. 确保不要创建任何使用 MyISAM 存储引擎的临时表。我们建议您暂停任何数据库工作负载，不要创建任何新的数据库连接，因为您即将升级。 5. 再次尝试就地升级。

就地升级被取消或减慢的原因	效果	允许在维护时段内完成就地升级的解决方案

已删除主用户	Aurora 取消升级。	**Important**
		不要删除主用户。
		
		但是，如果由于某种原因您碰巧删除了主用户，请使用以下 SQL 命令将其还原：
		
		`CREATE USER 'master_username '@'% IDENTIFIED BY ' master_user_password ' REQUIRE NONE PASSWORD EXPIRE DEFAULT ACCOUNT UNLOCK;`
		
		`GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, RELOAD, PROCESS, REFERENCES, INDEX, ALTER, SHOW DATABASES, CREATE TEMPORARY TABLES, LOCK TABLES, EXECUTE, REPLICATION SLAVE, REPLICATION CLIENT, CREATE VIEW, SHOW VIEW, CREATE ROUTINE, ALTER ROUTINE, CREATE USER, EVENT, TRIGGER, LOAD FROM S3, SELECT INTO S3, INVOKE LAMBDA, INVOKE SAGEMAKER, INVOKE COMPREHEND ON *.* TO 'master_username '@'% WITH GRANT OPTION;`

有关对导致升级预检查失败的问题进行故障排除的更多详细信息，请参阅以下博客：

- Amazon Aurora MySQL version 2 (with MySQL 5.7 compatibility) to version 3 (with MySQL 8.0 compatibility) upgrade checklist, Part 1
- Amazon Aurora MySQL version 2 (with MySQL 5.7 compatibility) to version 3 (with MySQL 8.0 compatibility) upgrade checklist, Part 2

您可以使用以下步骤对上表中的某些条件执行自己的检查。这样，您可以在知道数据库处于可以成功快速地完成升级的状态时安排升级。

- 您可以通过执行 XA RECOVER 语句来检查未完成的 XA 事务。然后，您可以在开始升级之前提交或回滚 XA 事务。
- 您可以通过执行 SHOW PROCESSLIST 语句并在输出中查找 CREATE、DROP、ALTER、RENAME 和 TRUNCATE 语句来检查 DDL 语句。在开始升级之前，等待所有 DDL 语句完成。
- 您可以通过查询 INFORMATION_SCHEMA.INNODB_TRX 表来检查未提交的行总数。该表为每个事务包含一行。TRX_ROWS_MODIFIED 列包含事务修改或插入的行数。
- 您可以通过执行 SHOW ENGINE INNODB STATUS SQL 语句并在输出中查找 History list length 来检查 InnoDB 历史记录列表的长度。您还可以通过运行以下查询直接检查值：

```
SELECT count FROM information_schema.innodb_metrics WHERE name =
 'trx_rseg_history_len';
```

  历史记录列表的长度对应于数据库为实施多版本并发控制 (MVCC) 而存储的撤消信息量。

Aurora MySQL 版本 3 的升级后清理

将任何 Aurora MySQL 版本 2 集群升级到 Aurora MySQL 版本 3 后，您可以执行以下其他清理操作：

- 为任何自定义参数组创建与 MySQL 8.0 兼容的新版本。将所有必要的自定义参数值应用于新参数组。
- 更新任何 CloudWatch 告警、设置脚本等，以便对名称受到包含性语言更改影响的任何指标使用新名称。有关此类指标的列表，请参阅 Aurora MySQL 版本 3 的包容性语言更改。
- 更新任何 AWS CloudFormation 模板，以对名称受到包含性语言更改影响的任何配置参数使用新名称。有关此类参数的列表，请参阅 Aurora MySQL 版本 3 的包容性语言更改。

空间索引

升级到 Aurora MySQL 版本 3 后，检查是否需要删除或重新创建与空间索引相关的对象和索引。在 MySQL 8.0 之前，Aurora 可以使用不包含空间资源标识符 (SRID) 的索引来优化空间查询。Aurora MySQL 版本 3 仅使用包含 SRID 的空间索引。在升级过程中，Aurora 会自动删除任何没有 SRID 的空间索引，并在数据库日志中打印警告消息。如果观察到此类警告消息，请在升级后使用 SRID 创建新的空间索引。有关 MySQL 8.0 中空间函数和数据类型更改的更多信息，请参阅 MySQL 参考手册中的 MySQL 8.0 中的变化。

Amazon Aurora MySQL 的数据库引擎更新和修复

您可以在《Amazon Aurora MySQL 兼容版发布说明》中找到以下信息：

- Database engine updates for Amazon Aurora MySQL version 3
- Database engine updates for Amazon Aurora MySQL version 2
- Database engine updates for Amazon Aurora MySQL version 1 (Deprecated)
- MySQL bugs fixed by Aurora MySQL database engine updates
- Security vulnerabilities fixed in Amazon Aurora MySQL

使用 Amazon Aurora PostgreSQL

Amazon Aurora PostgreSQL 是一个完全托管式、PostgreSQL 兼容且符合 ACID 的关系数据库引擎，结合了 Amazon Aurora 的速度、可靠性和可管理性，同时还具有开源数据库的简单性和成本效益。Aurora PostgreSQL 是 PostgreSQL 的直接替代品，可以让您通过简单且经济高效的方式设置、运行和扩展新的和现有的 PostgreSQL 部署，以使您腾出时间专注于业务和应用程序。要概括地了解有关 Aurora 的更多信息，请参阅什么是 Amazon Aurora？。

除了 Aurora 的优势之外，Aurora PostgreSQL 还提供了从 Amazon RDS 到 Aurora 的便捷迁移路径，并提供了按钮式迁移工具，以将现有的 RDS for PostgreSQL 应用程序转换为 Aurora PostgreSQL。使用 Aurora PostgreSQL 也可以轻松管理日常数据库任务（例如预调配、修补、备份、恢复、故障检测和修复）。

Aurora PostgreSQL 可以使用很多行业标准。例如，您可以使用 Aurora PostgreSQL 数据库构建符合 HIPAA 标准的应用程序，并依照与AWS签订的业务合作协议（BAA）存储医疗保健相关信息，包括保护的健康信息（PHI）。

Aurora PostgreSQL 符合 FedRAMP HIGH 要求。有关 AWS 和合规性工作的详细信息，请参阅 AWS 按合规性计划提供的范围内服务。

> **Important**
>
> 如果您的 Aurora PostgreSQL 数据库集群遇到问题，AWS 支持座席可能需要有关数据库运行状况的更多信息。目标是确保 AWS Support 在尽可能短的时间内获得尽可能多的所需信息。您可以使用 PG Collector 帮助在整合的 HTML 文件中收集有价值的数据库信息。有关 PG Collector、如何运行它以及如何下载 HTML 报告的更多信息，请参阅 PG Collector。
> 成功完成后，除非另有说明，否则脚本将以可读的 HTML 格式返回输出。该脚本设计为从 HTML 中排除任何可能危及您业务的数据或安全细节。它也不会对您的数据库或其环境进行任何修改。但是，如果您在 HTML 中发现任何不愿意分享的信息，您可以在上传 HTML 之前随意删除有问题的信息。当 HTML 的内容可以接受时，请在支持案例的案例详细信息中，使用附件部分进行上传。

主题

- 使用数据库预览环境
- 使用 Amazon Aurora PostgreSQL 实现高安全性
- 使用新 SSL/TLS 证书更新应用程序，以连接到 Aurora PostgreSQL 数据库集群

- 在 Aurora PostgreSQL 中使用 Kerberos 身份验证
- 将数据迁移到与 PostgreSQL 兼容的 Amazon Aurora
- 优化 Aurora PostgreSQL 中的查询性能
- 在 Aurora PostgreSQL 中使用未记录的表
- 在 Amazon Aurora PostgreSQL 上使用 PostgreSQL autovacuum
- 使用 Babelfish for Aurora PostgreSQL
- Amazon Aurora PostgreSQL 的性能和扩展
- 使用 Aurora PostgreSQL 的等待事件进行优化
- 使用 Amazon DevOps Guru 主动见解优化 Aurora PostgreSQL
- Amazon Aurora PostgreSQL 的最佳实践
- 使用 Amazon Aurora PostgreSQL 进行复制
- Aurora PostgreSQL 中的本地写入转发
- 使用 Aurora PostgreSQL 作为 Amazon Bedrock 的知识库
- 将 Amazon Aurora PostgreSQL 与其他AWS服务集成
- 监控 Aurora PostgreSQL 的查询执行计划和峰值内存
- 管理 Aurora PostgreSQL 的查询执行计划
- 使用扩展和外部数据包装器
- 使用适用于 PostgreSQL 的可信语言扩展
- Amazon Aurora PostgreSQL 参考
- Amazon Aurora PostgreSQL 的数据库引擎更新

使用数据库预览环境

PostgreSQL 社区每年发布新的 PostgreSQL 主要版本。同样，Amazon Aurora 将某些 PostgreSQL 主要版本作为预览版提供。这允许您使用预览版创建数据库集群，并在数据库预览环境中测试其功能。

数据库预览环境中的 Aurora PostgreSQL 数据库集群在功能上类似于其他 Aurora PostgreSQL 数据库集群。但是，不能将预览版用于生产。

请注意以下重要限制：

- 所有数据库实例和数据库集群在创建之后的 60 天删除，包括其所有备份和快照。
- 您只能在 Virtual Private Cloud (VPC) 中创建基于 Amazon VPC 服务的数据库实例。

- 您无法将数据库实例的快照复制到生产环境。

预览版支持以下选项：

- 只能使用 r5、r6g、r6i、r7g、r7i、r8g、x2g、t3 和 t4g 实例类型创建数据库实例。有关实例类的更多信息，请参阅 Amazon Aurora 数据库实例类。
- 您可以同时使用单可用区和多可用区部署。
- 您可以使用标准 PostgreSQL 转储和加载函数从数据库预览环境中导出数据库或将数据库导入数据库预览环境。

支持的数据库实例类类型

Amazon Aurora PostgreSQL 在预览版区域支持以下数据库实例类：

内存优化型类

- db.r5 – 内存优化型实例类
- db.r6g – 由 AWS Graviton2 处理器提供支持的内存优化型实例类
- db.r6i – 内存优化型实例类
- db.r7g – 由 AWS Graviton3 处理器提供支持的内存优化型实例类
- db.r7i – 内存优化型实例类
- db.r8g – 由 AWS Graviton4 处理器提供支持的内存优化型实例类
- db.x2g – 由 AWS Graviton2 处理器提供支持的内存优化型实例类

> **Note**
>
> 有关实例类列表的更多信息，请参阅数据库实例类类型。

可突增类

- db.t3.medium
- db.t3.large
- db.t4g.medium
- db.t4g.large

预览环境中不支持的功能

以下特征在预览环境中不可用：

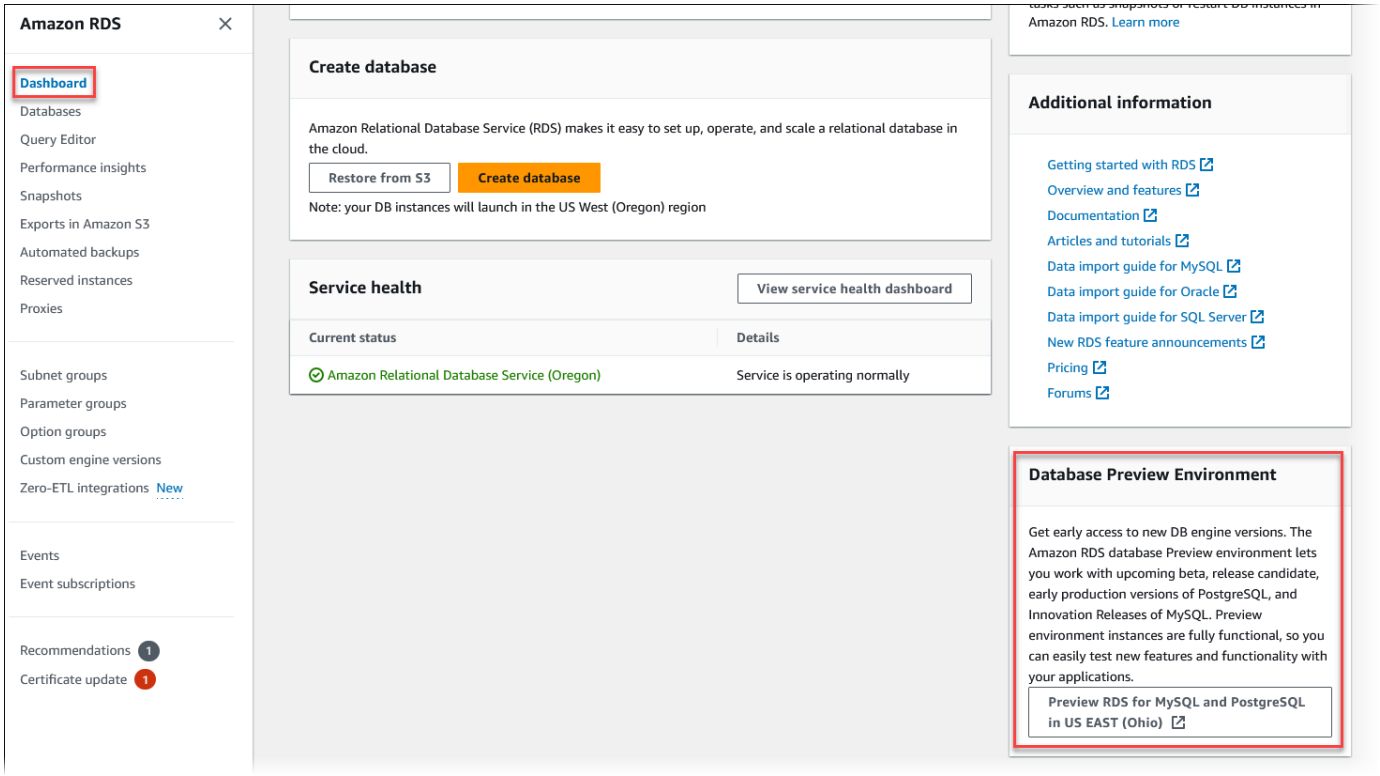
- Aurora Serverless v1 和 v2
- 主要版本升级（MVU）
- 预览版区域将不会发布任何新的次要版本
- RDS for PostgreSQL 到 Aurora PostgreSQL 入站复制
- Amazon RDS 蓝绿部署
- 跨区域快照复制
- Aurora Global Database
- 数据库活动流（DAS）、RDS 代理和 AWS DMS
- 自动扩缩只读副本
- AWS Bedrock
- RDS 导出
- 性能详情
- 自定义端点
- 快照复制
- 零 ETL
- Babelfish
- 所有扩展

在预览环境中创建新数据库集群

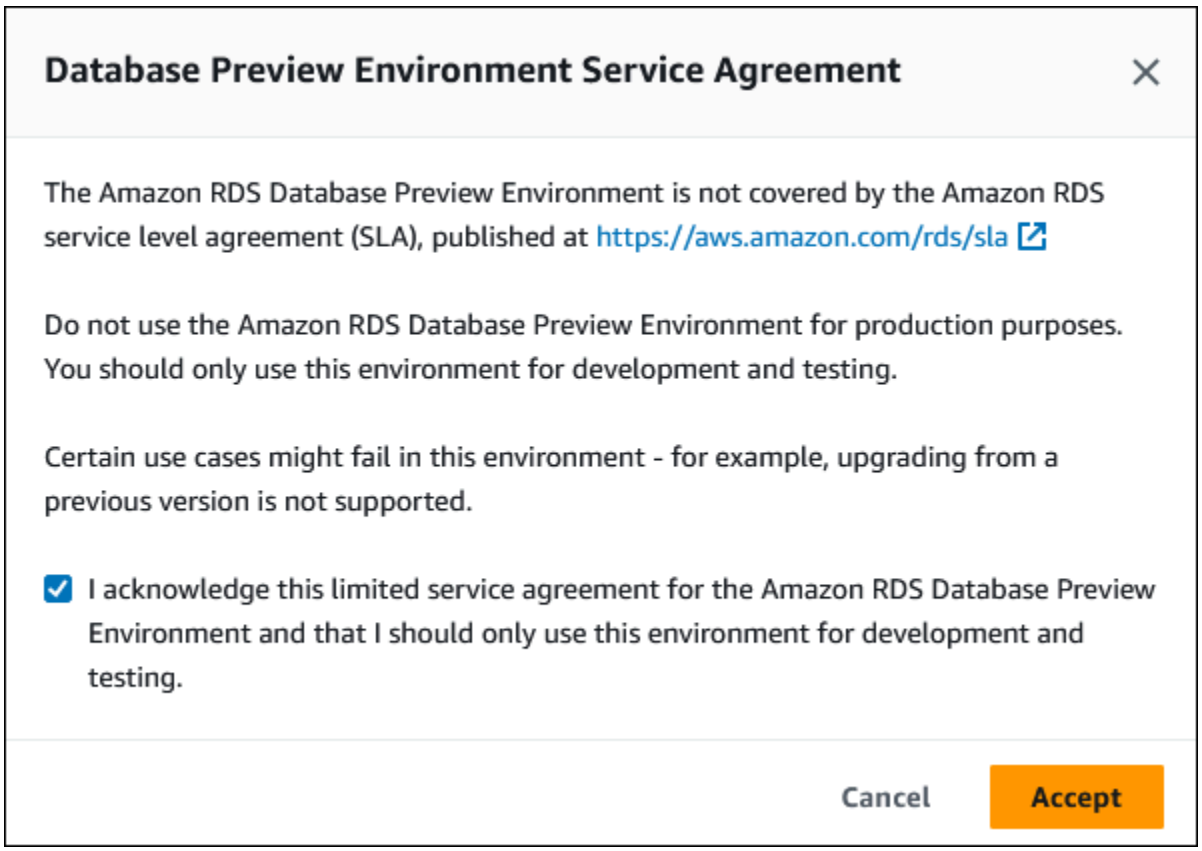
使用以下过程在预览环境中创建数据库集群。

在预览环境中创建数据库集群

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 从导航窗格选择 Dashboard (控制面板)。
3. 在控制面板页面中，找到控制面板页面上的 Database Preview Environment（数据库预览环境）部分，如下图所示。



您可以直接导航到 Database Preview Environment（数据库预览环境）。在继续操作之前，您必须确认并接受这些限制。



4. 要创建 Aurora PostgreSQL 数据库集群，请遵循与创建任何 Aurora 数据库集群相同的流程。有关更多信息，请参阅 创建 Amazon Aurora 数据库集群。

要使用 Aurora API 或 AWS CLI 在数据库预览环境中创建实例，请使用以下端点。

```
rds-preview.us-east-2.amazonaws.com
```

数据库预览环境中的 PostgreSQL 版本 17

⚠ 这是 Aurora PostgreSQL 版本 17 的预览文档。本文档随时可能更改。

PostgreSQL 版本 17.0 现可在 Amazon RDS 数据库预览环境中使用。PostgreSQL 版本 17 包含一些改进，如以下 PostgreSQL 文档中所述：

- PostgreSQL 17 已发布

有关数据库预览环境的信息，请参阅使用数据库预览环境。要从控制台中访问预览环境，请选择 https://console.aws.amazon.com/rds-preview/。

使用 Amazon Aurora PostgreSQL 实现高安全性

有关 Aurora 安全性的一般概述，请参阅 Amazon Aurora 中的安全性。您可以在几个不同级别管理 Amazon Aurora PostgreSQL 的安全性：

- 要控制可对 Aurora PostgreSQL 数据库集群和数据库实例执行 Amazon RDS 管理操作的人员，请使用 AWS Identity and Access Management (IAM)。IAM 在用户可以访问服务之前处理用户身份验证。它还处理授权，也就是说，是否允许用户执行他们想执行的操作。IAM 数据库身份验证是一种额外的身份验证方法，可以在创建 Aurora PostgreSQL 数据库集群时选择。有关更多信息，请参阅 Amazon Aurora 的 Identity and Access Management。

  如果您将 IAM 用于 Aurora PostgreSQL 数据库集群，请在打开 Amazon RDS 控制台 https://console.aws.amazon.com/rds/ 之前，先使用您的 IAM 凭据登录 AWS Management Console。
- 确保基于 Amazon VPC 服务在虚拟私有云（VPC）中创建 Aurora 数据库集群。要控制哪些设备和 Amazon EC2 实例能够建立与 VPC 中 Aurora 数据库集群的数据库实例端点和端口的连接，请使用 VPC 安全组。您可以使用安全套接字层 (SSL) 建立这些端点和端口连接。此外，公司的防火墙规则也可以控制公司中运行的哪些设备可以建立与数据库实例的连接。有关 VPC 的更多信息，请参阅 Amazon VPC 和 Amazon Aurora。

  支持的 VPC 租赁取决于 Aurora PostgreSQL 数据库集群使用的实例类。使用 `default` VPC 租赁，数据库集群在共享硬件上运行。使用 `dedicated` VPC 租赁，数据库集群在专用硬件实例上运行。可突增的性能数据库实例类仅支持原定设置 VPC 租赁。可突增的性能数据库实例类包括 db.t3 和 db.t4g 数据库实例类。所有其他 Aurora PostgreSQL 数据库实例类都支持原定设置和专用 VPC 租赁。

  有关实例类的更多信息，请参阅 Amazon Aurora 数据库实例类。有关 `default` 和 `dedicated` VPC 租赁的更多信息，请参阅 Amazon Elastic Compute Cloud 用户指南 中的专用实例。
- 要为在 Amazon Aurora 数据库集群上运行的 PostgreSQL 数据库授予权限，可采用与独立 PostgreSQL 实例相同的通用方式。`CREATE ROLE`、`ALTER ROLE`、`GRANT` 和 `REVOKE` 等命令的作用与它们在本地数据库中的作用相同，就像直接修改数据库、架构和表一样。

  PostgreSQL 通过使用角色来管理权限。`rds_superuser` 角色是 Aurora PostgreSQL 数据库集群权限最高的角色。此角色是自动创建的，并授予创建数据库集群的用户（主用户账户，默认情况下是 `postgres`）。要了解更多信息，请参阅了解 PostgreSQL 角色和权限。

所有可用的 Aurora PostgreSQL 版本（包括版本 10、11、12、13、14 及更高版本）支持对密码使用加盐质询响应身份验证机制（SCRAM），作为消息摘要（MD5）的替代方案。我们建议您使用 SCRAM，因为它比 MD5 更安全。有关更多信息，包括如何将数据库用户密码从 MD5 迁移到 SCRAM，请参阅使用 SCRAM 进行 PostgreSQL 密码加密。

了解 PostgreSQL 角色和权限

在使用 AWS Management Console 创建 Aurora PostgreSQL 数据库集群 时，将同时创建管理员账户。默认情况下，其名称为 `postgres`，如以下屏幕截图所示：



您可以选择其他名称，而不是接受默认值（`postgres`）。如果这样做，您选择的名称必须以字母开头，并且必须介于 1 到 16 个字母数字字符之间。为简单起见，在整个指南中，我们将使用默认值 (`postgres`) 来指代此主用户账户。

如果您使用 `create-db-cluster` AWS CLI 而不是 AWS Management Console，则可以通过 `master-username` 参数传递名称来创建用户名。有关更多信息，请参阅 步骤 2：创建 Aurora PostgreSQL 数据库集群。

无论您是使用 AWS Management Console、AWS CLI 还是 Amazon RDS API，也无论您是使用默认 `postgres` 名称还是选择其他名称，这第一个数据库用户账户都是 `rds_superuser` 组的成员并具有 `rds_superuser` 权限。

主题

- 了解 rds_superuser 角色
- 控制用户对 PostgreSQL 数据库的访问
- 委托和控制用户密码管理
- 使用 SCRAM 进行 PostgreSQL 密码加密

了解 rds_superuser 角色

在 PostgreSQL 中，角色可以针对数据库中的各种对象定义一个用户、一个组或一组授予组或用户的特定权限。PostgreSQL 命令 CREATE USER 和 CREATE GROUP 已替换为更通用的 CREATE ROLE，并使用特定属性来区分数据库用户。数据库用户可以被视为具有 LOGIN 权限的角色。

> **Note**
>
> 仍然可以使用 CREATE USER 和 CREATE GROUP 命令。有关更多信息，请参阅 PostgreSQL 文档中的数据库角色。

postgres 用户是您的 Aurora PostgreSQL 数据库集群 上权限最高的数据库用户。它具有以下 CREATE ROLE 语句所定义的特征。

```
CREATE ROLE postgres WITH LOGIN NOSUPERUSER INHERIT CREATEDB CREATEROLE NOREPLICATION
 VALID UNTIL 'infinity'
```

属性 NOSUPERUSER、NOREPLICATION、INHERIT 和 VALID UNTIL 'infinity' 是 CREATE ROLE 的默认选项，除非另有说明。

默认情况下，postgres 拥有授予 rds_superuser 角色的权限以及创建角色和数据库的权限。rds_superuser 角色允许 postgres 用户执行以下操作：

- 添加可用于 Aurora PostgreSQL。有关更多信息，请参阅使用扩展和外部数据包装器。
- 为用户创建角色并向用户授予权限。有关更多信息，请参阅 PostgreSQL 文档中的 CREATE ROLE 和 GRANT。
- 创建数据库。有关更多信息，请参阅 PostgreSQL 文档中的 CREATE DATABASE。
- 将 rds_superuser 权限授予没有这些权限的用户角色，并根据需要撤销权限。建议您仅向执行超级用户任务的那些用户授予此角色。换句话说，您可以将此角色授予数据库管理员 (DBA) 或系统管理员。
- 向没有 rds_superuser 角色的数据库用户授予（和撤销）rds_replication 角色。
- 向没有 rds_superuser 角色的数据库用户授予（和撤销）rds_password 角色。
- 通过使用 pg_stat_activity 视图获取有关所有数据库连接的状态信息。需要时，rds_superuser 可以通过使用 pg_terminate_backend 或 pg_cancel_backend 停止任何连接。

在 CREATE ROLE postgres... 语句中，您可以看到 postgres 用户角色明确禁止 PostgreSQL superuser 权限。Aurora PostgreSQL 是一项托管服务，因此您无法访问主机操作系统，也无法使用 PostgreSQL superuser 账户进行连接。许多需要独立 PostgreSQL 上的 superuser 访问权限的任务都由 Aurora 自动管理。

有关授权权限的更多信息，请参阅 PostgreSQL 文档中的 GRANT。

rds_superuser 角色是 Aurora PostgreSQL 数据库集群中的几个预定义角色之一。

> **Note**
>
> 在 PostgreSQL 13 和更早版本中，预定义角色称为默认角色。

在下面的列表中，您可以找到为新 Aurora PostgreSQL 数据库集群自动创建的一些其他预定义角色。无法更改预定义角色及其权限。无法删除、重命名或修改这些预定义角色的权限。此类尝试会导致错误。

- rds_password – 可以为数据库用户更改密码和设置密码限制的角色。默认情况下，rds_superuser 角色被授予此角色，并且可以将此角色授予数据库用户。有关更多信息，请参阅 控制用户对 PostgreSQL 数据库的访问。
  - 对于早于 14 的 RDS for PostgreSQL 版本，rds_password 角色可以为数据库用户和具有 rds_superuser 角色的用户更改密码和设置密码限制。在 RDS for PostgreSQL 14 及更高版本中，rds_password 角色只能为数据库用户更改密码和设置密码限制。只有具有 rds_superuser 角色的用户才能对具有 rds_superuser 角色的其他用户执行这些操作。
- rdsadmin – 具有 superuser 权限的管理员将对独立的 PostgreSQL 数据库执行许多管理任务，此角色专为处理这些管理任务而创建。此角色由 Aurora PostgreSQL 在内部用于许多管理任务。

查看角色及其权限

根据您的 PostgreSQL 版本，您可以使用不同的命令在 RDS for PostgreSQL 数据库实例中查看预定义角色及其权限。

对于 PostgreSQL 15 及更早版本

连接到 RDS for PostgreSQL 数据库实例，并在 psql 中使用 \du 命令：

```
postgres=> \du
```

```
                                                                List of roles
    Role name    |                          Attributes                          |
                  Member of
-----------------+-------------------------------------------------------------
+-------------------------------------------------------
 postgres        | Create role, Create DB                                     +|
 {rds_superuser}
                 | Password valid until infinity                               |
 rds_ad          | Cannot login                                                | {}
 rds_iam         | Cannot login                                                | {}
 rds_password    | Cannot login                                                | {}
 rds_replication | Cannot login                                                | {}
 rds_superuser   | Cannot login                                                |
 {pg_monitor,pg_signal_backend,rds_password,rds_replication}
 rdsadmin        | Superuser, Create role, Create DB, Replication, Bypass RLS+| {}
                 | Password valid until infinity                               |
```

对于 PostgreSQL 16 及更高版本

从 PostgreSQL 16 开始，可以使用 `\drg+` 命令来查看详细的角色成员资格信息：

```
postgres=> \drg+
                              List of role grants
   Role name   |          Member of          |       Options       | Grantor
---------------+-----------------------------+---------------------+----------
 postgres      | rds_superuser               | INHERIT, SET        | rdsadmin
 rds_superuser | pg_checkpoint               | ADMIN, INHERIT, SET | rdsadmin
 rds_superuser | pg_monitor                  | ADMIN, INHERIT, SET | rdsadmin
 rds_superuser | pg_signal_backend           | ADMIN, INHERIT, SET | rdsadmin
 rds_superuser | pg_use_reserved_connections | ADMIN, INHERIT, SET | rdsadmin
 rds_superuser | rds_password                | ADMIN, INHERIT, SET | rdsadmin
 rds_superuser | rds_replication             | ADMIN, INHERIT, SET | rdsadmin
```

要检查角色成员资格，无论 PostgreSQL 版本如何，都可以使用以下 SQL 查询：

```
SELECT m.rolname AS "Role name", r.rolname AS "Member of"
FROM pg_catalog.pg_roles m
JOIN pg_catalog.pg_auth_members pam ON (pam.member = m.oid)
LEFT JOIN pg_catalog.pg_roles r ON (pam.roleid = r.oid)
LEFT JOIN pg_catalog.pg_roles g ON (pam.grantor = g.oid)
WHERE m.rolname !~ '^pg_'
ORDER BY 1, 2;
```

在输出中，您可以看到 rds_superuser 不是数据库用户角色（无法登录），但它具有许多其他角色的权限。您还可以看到数据库用户 postgres 是 rds_superuser 角色的成员。如前所述，postgres 是 Amazon RDS 控制台的 Create database（创建数据库）页面中的默认值。如果您选择了另一个名称，则该名称将显示在角色列表中。

> **Note**
>
> Aurora PostgreSQL 版本 15.2 和 14.7 引入了 rds_superuser 角色的限制行为。Aurora PostgreSQL 用户需要在相应的数据库上获得 CONNECT 权限才能连接，即使向该用户授予了 rds_superuser 角色也是如此。在 Aurora PostgreSQL 版本 14.7 和 15.2 之前，如果向用户授予了 rds_superuser 角色，则用户可以连接到任何数据库和系统表。这种限制行为符合 AWS 的要求和 Amazon Aurora 对持续改善安全性的承诺。
> 如果上述增强功能有影响，请更新应用程序中的相应逻辑。

控制用户对 PostgreSQL 数据库的访问

PostgreSQL 中的新数据库始终使用数据库 public 架构中的一组默认权限创建，允许所有数据库用户和角色创建对象。例如，这些权限使数据库用户能够连接数据库，并在连接后创建临时表格。

为了更好地控制用户对您在 Aurora PostgreSQL 数据库集群主节点上创建的数据库实例的访问，我们建议您撤消这些默认 public 权限。撤消后，您可以更精确地为数据库用户授权，如以下过程中所示。

为新数据库实例设置角色和权限

假设您正在新创建的 Aurora PostgreSQL 数据库集群上设置数据库，以供几位研究人员使用，他们都需要对数据库的读写访问权限。

1. 使用 psql（或 pgAdmin）连接到 Aurora PostgreSQL 数据库集群上的主数据库实例：

   ```
   psql --host=your-cluster-instance-1.666666666666.aws-region.rds.amazonaws.com --
   port=5432 --username=postgres --password
   ```

   出现提示时请输入密码。psql 客户端会建立连接并显示默认的管理连接数据库 postgres=>，作为提示符。
2. 要阻止数据库用户在 public 架构中创建对象，执行以下操作：

   ```
   postgres=> REVOKE CREATE ON SCHEMA public FROM PUBLIC;
   ```

```
REVOKE
```

3. 接下来，创建一个新数据库实例：

```
postgres=> CREATE DATABASE lab_db;
CREATE DATABASE
```

4. 在这个新数据库上，撤消 PUBLIC 架构的所有权限。

```
postgres=> REVOKE ALL ON DATABASE lab_db FROM public;
REVOKE
```

5. 为数据库用户创建角色。

```
postgres=> CREATE ROLE lab_tech;
CREATE ROLE
```

6. 为具有此角色的数据库用户提供连接到数据库的能力。

```
postgres=> GRANT CONNECT ON DATABASE lab_db TO lab_tech;
GRANT
```

7. 向具有 lab_tech 角色的所有用户授予对此数据库的所有权限。

```
postgres=> GRANT ALL PRIVILEGES ON DATABASE lab_db TO lab_tech;
GRANT
```

8. 创建数据库用户，如下所示：

```
postgres=> CREATE ROLE lab_user1 LOGIN PASSWORD 'change_me';
CREATE ROLE
postgres=> CREATE ROLE lab_user2 LOGIN PASSWORD 'change_me';
CREATE ROLE
```

9. 向这两个用户授予与 lab_tech 角色关联的权限：

```
postgres=> GRANT lab_tech TO lab_user1;
GRANT ROLE
postgres=> GRANT lab_tech TO lab_user2;
GRANT ROLE
```

此时，lab_user1 和 lab_user2 可以连接到 lab_db 数据库。此示例未遵循企业使用的最佳实践，其中可能包括创建多个数据库实例、不同的架构和授予有限权限。有关更多完整信息和其他方案，请参阅管理 PostgreSQL 用户和角色。

有关 PostgreSQL 数据库中特权的更多信息，请参阅 PostgreSQL 文档中的 GRANT 命令。

委托和控制用户密码管理

作为 DBA，您可能需要委托用户密码的管理。或者，您可能希望防止数据库用户更改其密码或重新配置密码限制，例如密码生命周期。要确保只有您选择的数据库用户才能更改密码设置，可以启用受限密码管理特征。激活此特征时，只有那些已被授予 rds_password 角色的数据库用户可以管理密码。

> **Note**
>
> 要使用受限密码管理，您的 Aurora PostgreSQL 数据库集群必须运行 Amazon Aurora PostgreSQL 10.6 或更高版本。

默认情况下，此特征为 off，如下所示：

```
postgres=> SHOW rds.restrict_password_commands;
  rds.restrict_password_commands
--------------------------------
 off
(1 row)
```

要启用此特征，请使用自定义参数组并将 rds.restrict_password_commands 的设置更改为 1。一定要重新启动 Aurora PostgreSQL 的主数据库实例，此设置才能生效。

激活此特征后，以下 SQL 命令需要 rds_password 权限：

```
CREATE ROLE myrole WITH PASSWORD 'mypassword';
CREATE ROLE myrole WITH PASSWORD 'mypassword' VALID UNTIL '2023-01-01';
ALTER ROLE myrole WITH PASSWORD 'mypassword' VALID UNTIL '2023-01-01';
ALTER ROLE myrole WITH PASSWORD 'mypassword';
ALTER ROLE myrole VALID UNTIL '2023-01-01';
ALTER ROLE myrole RENAME TO myrole2;
```

如果密码使用 MD5 哈希算法，重命名角色 (ALTER ROLE myrole RENAME TO newname) 也会受到限制。

激活此特征后，在没有 rds_password 角色权限的情况下尝试这些 SQL 命令中的任何一个都会生成以下错误：

```
ERROR: must be a member of rds_password to alter passwords
```

我们建议您仅将 rds_password 授予少数几个仅用于密码管理的角色。如果您将 rds_password 权限授予没有 rds_superuser 权限的数据库用户，则还需要授他们 CREATEROLE 属性。

请确保您验证了密码要求，例如客户端上的过期时间以及所需的复杂性。如果您使用自己的客户端实用程序进行与密码相关的更改，则该实用程序需要是 rds_password 的成员并具有 CREATE ROLE 权限。

使用 SCRAM 进行 PostgreSQL 密码加密

在对密码进行加密时，加盐质询响应身份验证机制（SCRAM）是 PostgreSQL 的默认消息摘要（MD5）算法的替代方案。–般认为 SCRAM 身份验证机制比 MD5 更安全。要了解有关这两种不同的密码保护方法的更多信息，请参阅 PostgreSQL 文档中的密码身份验证。

我们建议您使用 SCRAM 而不是 MD5 作为您的 Aurora PostgreSQL 数据库集群 的密码加密方案。Aurora PostgreSQL 版本 10 以及所有更高的主要和次要版本均支持 SCRAM。这是一种加密质询-响应机制，它使用 scram-sha-256 算法进行密码身份验证和加密。

要支持 SCRAM，您可能需要更新客户端应用程序的库。例如，42.2.0 之前的 JDBC 版本不支持 SCRAM。有关更多信息，请参阅 PostgreSQL JDBC 驱动程序文档中的 PostgreSQL JDBC 驱动程序。有关其他 PostgreSQL 驱动程序和 SCRAM 支持的列表，请参阅 PostgreSQL 文档中的驱动程序列表。

默认情况下，Aurora PostgreSQL 版本 14 和更高版本支持使用 scram-sha-256 对新数据库集群进行密码加密。对于这些版本，默认数据库集群参数组 (default.aurora-postgresql14) 将其 password_encryption 值设置为 scram-sha-256。Aurora Serverless v1 不支持 SCRAM。

设置 Aurora PostgreSQL 数据库集群以要求使用 SRAM

对于 Aurora PostgreSQL 14.3 及更高版本，您可以要求 Aurora PostgreSQL 数据库集群仅接受使用 scram-sha-256 算法的密码。

> **Important**
>
> 对于带有 PostgreSQL 数据库的现有 RDS 代理，如果您将数据库身份验证修改为仅使用 SCRAM，代理将在长达 60 秒的时间内不可用。要避免此问题，请执行以下操作之一：

- 确保数据库同时允许 SCRAM 和 MD5 身份验证。
- 要仅使用 SCRAM 身份验证，请创建一个新代理，将应用程序流量迁移到新代理，然后删除先前与数据库关联的代理。

在对系统进行更改之前，请务必了解完整的过程，如下所示：

- 获取有关所有数据库用户的所有角色和密码加密的信息。
- 仔细检查 Aurora PostgreSQL 数据库集群 的参数设置，以了解用于控制密码加密的参数。
- 如果您的 Aurora PostgreSQL 数据库集群 使用默认参数组，您需要创建自定义数据库集群参数组，然后将其应用到您的 Aurora PostgreSQL 数据库集群 ，以便您可以在需要时修改参数。如果您的 Aurora PostgreSQL 数据库集群 使用自定义参数组，您可以稍后根据需要在此过程中修改必要的参数。
- 将 `password_encryption` 参数更改为 `scram-sha-256`。
- 通知所有数据库用户他们需要更新密码。对您的 `postgres` 账户执行相同的操作。使用 scram-sha-256 算法对新密码进行加密和存储。
- 验证是否使用加密类型对所有密码加密。
- 如果所有密码都使用 scram-sha-256，您可以将 `rds.accepted_password_auth_method` 参数从 `md5+scram` 更改为 `scram-sha-256`。

> **Warning**
>
> 仅将 `rds.accepted_password_auth_method` 更改为 scram-sha-256 之后，使用 `md5` 加密的密码的任何用户（角色）都将无法连接。

做好准备，以要求 Aurora PostgreSQL 数据库集群使用 SCRAM

在对 Aurora PostgreSQL 数据库集群进行任何更改之前，检查所有现有的数据库用户账户。另外，请检查用于密码的加密类型。您可以使用 `rds_tools` 扩展来执行这些任务。要了解哪些 PostgreSQL 版本支持 `rds_tools`，请参阅 Extension versions for Amazon RDS for PostgreSQL。

获取数据库用户（角色）和密码加密方法的列表

1. 使用 `psql` 连接到 Aurora PostgreSQL 数据库集群的主实例，如下所示。

```
psql --host=cluster-name-instance-1.111122223333.aws-region.rds.amazonaws.com --
port=5432 --username=postgres --password
```

2. 安装 rds_tools 扩展。

```
postgres=> CREATE EXTENSION rds_tools;
CREATE EXTENSION
```

3. 获取角色和加密的列表。

```
postgres=> SELECT * FROM
      rds_tools.role_password_encryption_type();
```

您将看到类似以下内容的输出。

```
       rolname        | encryption_type
----------------------+-----------------
 pg_monitor           |
 pg_read_all_settings |
 pg_read_all_stats    |
 pg_stat_scan_tables  |
 pg_signal_backend    |
 lab_tester           | md5
 user_465             | md5
 postgres             | md5
(8 rows)
```

创建自定义数据库集群参数组

> **Note**
>
> 如果您的 Aurora PostgreSQL 数据库集群 已使用自定义参数组，您不需要创建新参数组。

有关 Aurora 的参数组的概述，请参阅在 Amazon Aurora 中创建数据库集群参数组。

用于密码的密码加密类型在一个参数（即 password_encryption）中设置。Aurora PostgreSQL 数据库集群 允许的加密在另一个参数 rds.accepted_password_auth_method 中设置。更改其中任何一个的默认值都要求您创建自定义数据库集群参数组，然后将其应用到您的集群。

也可以使用 AWS Management Console 或 RDS API 创建自定义数据库集群参数组。有关更多信息，请参阅在 Amazon Aurora 中创建数据库集群参数组。

现在可以将自定义参数组与数据库实例关联。

创建自定义数据库集群参数组

1. 使用 CLI 命令 create-db-cluster-parameter-group 为集群创建自定义参数组。以下示例使用 aurora-postgresql13 作为此自定义参数组的来源。

   对于 Linux、macOS 或 Unix：

   ```
   aws rds create-db-cluster-parameter-group --db-cluster-parameter-group-name 'docs-lab-scram-passwords' \
     --db-parameter-group-family aurora-postgresql13  --description 'Custom DB cluster parameter group for SCRAM'
   ```

   对于 Windows：

   ```
   aws rds create-db-cluster-parameter-group --db-cluster-parameter-group-name "docs-lab-scram-passwords" ^
     --db-parameter-group-family aurora-postgresql13  --description "Custom DB cluster parameter group for SCRAM"
   ```

   现在可以将自定义参数组与集群关联。

2. 使用 CLI 命令 modify-db-cluster 将此自定义参数组应用于 Aurora PostgreSQL 数据库集群。

   对于 Linux、macOS 或 Unix：

   ```
   aws rds modify-db-cluster --db-cluster-identifier 'your-instance-name' \
           --db-cluster-parameter-group-name "docs-lab-scram-passwords
   ```

   对于 Windows：

   ```
   aws rds modify-db-cluster --db-cluster-identifier "your-instance-name" ^
           --db-cluster-parameter-group-name "docs-lab-scram-passwords
   ```

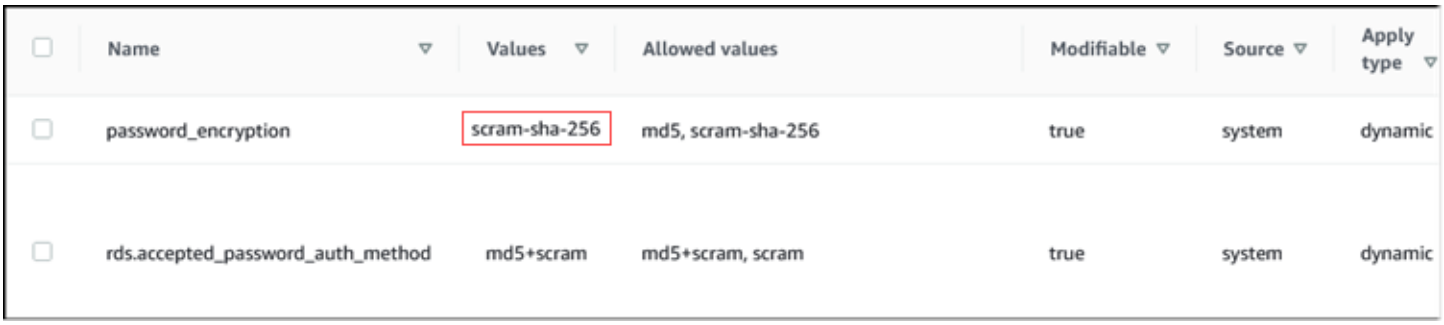
要重新同步 Aurora PostgreSQL 数据库集群与您的自定义数据库集群参数组，您需要重启集群的主实例和所有其他实例。

配置密码加密以使用 SCRAM

Aurora PostgreSQL 数据库集群使用的密码加密机制在数据库集群参数组的 `password_encryption` 参数中设置。允许的值为未设置、`md5` 或 `scram-sha-256`。默认值取决于 Aurora PostgreSQL 版本，如下所示：

- Aurora PostgreSQL 14 – 默认为 `scram-sha-256`
- Aurora PostgreSQL 13 – 默认为 `md5`

使用附加到 Aurora PostgreSQL 数据库集群 的自定义数据库集群参数组，您可以修改密码加密参数的值。



Name	Values	Allowed values	Modifiable	Source	Apply type
password_encryption	scram-sha-256	md5, scram-sha-256	true	system	dynamic
rds.accepted_password_auth_method	md5+scram	md5+scram, scram	true	system	dynamic

将密码加密设置更改为 scram-sha-256

- 将密码加密的值更改为 scram-sha-256，如下所示。可以立即应用更改，因为参数是动态的，这样，无需重新启动即可使更改生效。

  对于 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-name \
  'docs-lab-scram-passwords' --parameters
 'ParameterName=password_encryption,ParameterValue=scram-
sha-256,ApplyMethod=immediate'
```

  对于 Windows：

```
aws rds modify-db-parameter-group --db-parameter-group-name ^
```

```
  "docs-lab-scram-passwords" --parameters
 "ParameterName=password_encryption,ParameterValue=scram-
sha-256,ApplyMethod=immediate"
```

将用户角色的密码迁移到 SCRAM

您可以将用户角色的密码迁移到 SCRAM，如下所述。

将数据库用户（角色）密码从 MD5 迁移到 SCRAM

1. 以管理员用户身份（默认用户名 `postgres`）登录，如下所示。

```
psql --host=cluster-name-instance-1.111122223333.aws-region.rds.amazonaws.com --
port=5432 --username=postgres --password
```

2. 通过使用以下命令，在 RDS for PostgreSQL 数据库实例上检查 `password_encryption` 参数的设置。

```
postgres=> SHOW password_encryption;
 password_encryption
---------------------
 md5
 (1 row)
```

3. 将此参数的值更改为 scram-sha-256。这是一个动态参数，因此您不需要在进行此更改后重新启动该实例。再次检查该值，以确保它现在已设置为 `scram-sha-256`，如下所示。

```
postgres=> SHOW password_encryption;
 password_encryption
---------------------
 scram-sha-256
 (1 row)
```

4. 通知所有数据库用户更改其密码。请确保还要更改您自己的用于 `postgres` 账户的密码（具有 `rds_superuser` 权限的数据库用户）。

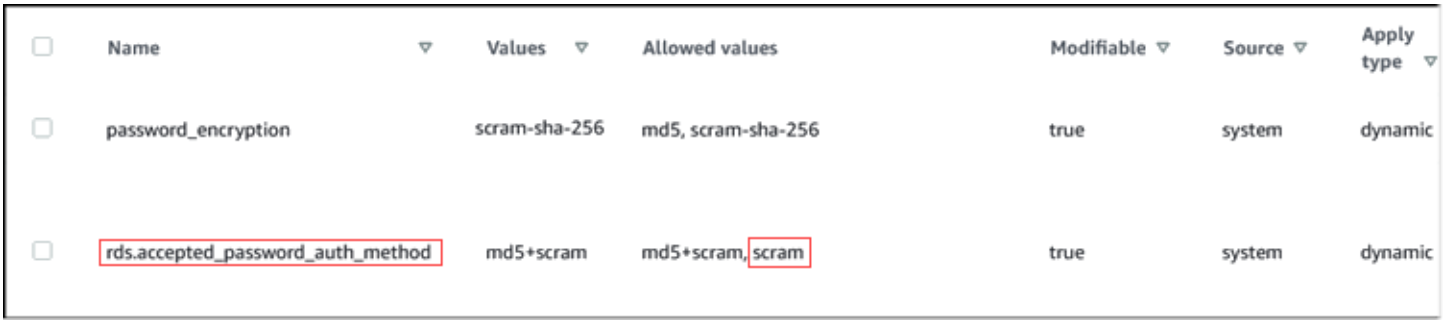
```
labdb=> ALTER ROLE postgres WITH LOGIN PASSWORD 'change_me';
ALTER ROLE
```

5. 对于您的 Aurora PostgreSQL 数据库集群 上的所有数据库重复此过程。

更改参数以要求使用 SCRAM

这是该过程的最后一步。在以下过程中进行更改后，任何仍对密码使用 md5 加密的用户账户（角色）都将无法登录到 Aurora PostgreSQL 数据库集群 。

rds.accepted_password_auth_method 指定 Aurora PostgreSQL 数据库集群 在登录过程中接受的用户密码加密方法。默认值为 md5+scram，这意味着可以接受任一种方法。在下面的图中，您可以找到此参数的默认设置。



Name	Values	Allowed values	Modifiable	Source	Apply type
password_encryption	scram-sha-256	md5, scram-sha-256	true	system	dynamic
rds.accepted_password_auth_method	md5+scram	md5+scram, scram	true	system	dynamic

此参数允许的值仅为 md5+scram 或 scram。如果将此参数值更改为 scram，则必须使用这种方法。

更改参数值以要求对密码进行 SCRAM 身份验证

1. 验证 Aurora PostgreSQL 数据库集群 上所有数据库的所有数据库用户密码是否使用 scram-sha-256 进行密码加密。为此，查询 rds_tools 以获得角色（用户）和加密类型，如下所示。

```
postgres=> SELECT * FROM rds_tools.role_password_encryption_type();
  rolname        | encryption_type
  ---------------------+-----------------
  pg_monitor           |
  pg_read_all_settings |
  pg_read_all_stats    |
  pg_stat_scan_tables  |
  pg_signal_backend    |
  lab_tester           | scram-sha-256
  user_465             | scram-sha-256
  postgres             | scram-sha-256
  ( rows)
```

2. 在您的 Aurora PostgreSQL 数据库集群 中的所有数据库实例中重复此查询。

   如果所有密码都使用 scram-sha-256，您可以继续操作。

3. 将接受的密码身份验证的值更改为 scram-sha-256，如下所示。

对于 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-name 'docs-
lab-scram-passwords' \
  --parameters
 'ParameterName=rds.accepted_password_auth_method,ParameterValue=scram,ApplyMethod=immediat
```

对于 Windows：

```
aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-name "docs-
lab-scram-passwords" ^
  --parameters
 "ParameterName=rds.accepted_password_auth_method,ParameterValue=scram,ApplyMethod=immediat
```

利用 SSL/TLS 保护 Aurora PostgreSQL 数据

Amazon RDS 支持对 Aurora PostgreSQL 数据库集群进行安全套接字层 (SSL) 和传输层安全性 (TLS) 加密。使用 SSL/TLS 可加密应用程序与 Aurora PostgreSQL 数据库集群之间的连接。您还可强制至 Aurora PostgreSQL 数据库集群的所有连接使用 SSL/TLS。Amazon Aurora PostgreSQL 支持传输层安全性 (TLS) 版本 1.1 和 1.2。我们建议使用 TLS 1.2 加密连接。我们在以下 Aurora PostgreSQL 版本中增加了对 TLSv1.3 的支持：

- 15.3 及更高版本
- 14.8 及更高的 14 版本
- 13.11 及更高的 13 版本
- 12.15 及更高的 12 版本
- 11.20 及更高的 11 版本

有关 SSL/TLS 支持和 PostgreSQL 数据库的一般信息，请参阅 PostgreSQL 文档中的 SSL 支持。有关通过 JDBC 使用 SSL/TLS 连接的信息，请参阅 PostgreSQL 文档中的配置客户端。

主题

- 需要与 Aurora PostgreSQL 数据库集群建立 SSL/TLS 连接
- 确定 SSL/TLS 连接状态
- 配置密码套件以连接到 Aurora PostgreSQL 数据库集群

SSL/TLS 支持已在 Aurora PostgreSQL 的所有 AWS 区域可用。在创建 Aurora PostgreSQL 数据库集群时，Amazon RDS 会为该数据库集群创建一个 SSL/TLS 证书。如果启用 SSL/TLS 证书验证，SSL/TLS 证书会将数据库集群终端节点作为 SSL/TLS 证书的公用名 (CN) 包含在内以防止欺诈攻击。

通过 SSL/TLS 连接到 Aurora PostgreSQL 数据库集群

1. 下载证书。

   有关下载证书的信息，请参阅 使用 SSL/TLS 加密与数据库集群的连接。
2. 将证书导入您的操作系统。
3. 通过 SSL/TLS 连接到 Aurora PostgreSQL 数据库集群。

   使用 SSL/TLS 连接时，客户端可以选择是否验证证书链。如果连接参数指定 `sslmode=verify-ca` 或 `sslmode=verify-full`，则客户端要求 RDS CA 证书位于其信任存储中或在连接 URL 中进行引用。此要求是为了验证签署您的数据库证书的证书链。

   当客户端（如 psql 或 JDBC）配置有 SSL/TLS 支持时，预设情况下，该客户端会首先尝试使用 SSL/TLS 连接到数据库。如果该客户端无法使用 SSL/TLS 进行连接，它将恢复为不使用 SSL/TLS 进行连接。原定设置情况下，适用于 JDBC 客户端和基于 libpq 的客户端的 `sslmode` 选项设置为 `prefer`。

   使用 `sslrootcert` 参数引用证书，例如，`sslrootcert=rds-ssl-ca-cert.pem`。

下面是使用 psql 连接到 Aurora PostgreSQL 数据库集群的示例。

```
$ psql -h testpg.cdhmuqifdpib.us-east-1.rds.amazonaws.com -p 5432 \
    "dbname=testpg user=testuser sslrootcert=rds-ca-2015-root.pem sslmode=verify-full"
```

需要与 Aurora PostgreSQL 数据库集群建立 SSL/TLS 连接

要想要求使用 SSL/TLS 连接到 Aurora PostgreSQL 数据库集群，请使用 `rds.force_ssl` 参数。

- 要想要求使用 SSL/TLS 连接，请将 `rds.force_ssl` 参数值设置为 1（开启）。
- 要想不要求使用 SSL/TLS 连接，请将 `rds.force_ssl` 参数值设置为 0（关闭）。

此参数的默认值取决于 Aurora PostgreSQL 版本：

- 对于 Aurora PostgreSQL 版本 17 及更高版本：默认值为 1（开启）。

- 对于 Aurora PostgreSQL 版本 16 及较低版本：默认值为 0（关闭）。

> **Note**
>
> 当您从 Aurora PostgreSQL 版本 16 或更早版本执行主要版本升级，从而升级到版本 17 或更高版本时，参数的默认值将从 0（关闭）更改为 1（开启）。对于未配置 SSL 的应用程序，此更改可能会导致连接故障。您可以将此参数设置为 0（关闭），从而恢复到之前的默认行为。

有关处理参数的更多信息，请参阅 Amazon Aurora 的参数组。

更新 `rds.force_ssl` 参数还会将 PostgreSQL `ssl` 参数设置为 1 (on) 并将数据库集群的 `pg_hba.conf` 文件修改为支持新的 SSL/TLS 配置。

当数据库集群的 `rds.force_ssl` 参数设置为 1 时，您将在连接时看到类似以下的输出，指示现在需要 SSL/TLS：

```
$ psql postgres -h SOMEHOST.amazonaws.com -p 8192 -U someuser
psql (9.3.12, server 9.4.4)
WARNING: psql major version 9.3, server major version 9.4.
Some psql features might not work.
SSL connection (cipher: DHE-RSA-AES256-SHA, bits: 256)
Type "help" for help.

postgres=>
```

确定 SSL/TLS 连接状态

当您连接到数据库集群后，登录横幅中将显示连接的加密状态。

```
Password for user master:
psql (9.3.12)
SSL connection (cipher: DHE-RSA-AES256-SHA, bits: 256)
Type "help" for help.

postgres=>
```

也可加载 sslinfo 扩展，然后调用 ssl_is_used() 函数以判断是否在使用 SSL/TLS。如果连接使用的是 SSL/TLS，则此函数将返回 t；否则返回 f。

```
postgres=> create extension sslinfo;
CREATE EXTENSION

postgres=> select ssl_is_used();
 ssl_is_used
---------
t
(1 row)
```

您可以使用 select ssl_cipher() 命令确定 SSL/TLS 密码：

```
postgres=> select ssl_cipher();
ssl_cipher
--------------------
DHE-RSA-AES256-SHA
(1 row)
```

如果您启用 set rds.force_ssl 并重新启动数据库集群，则将拒绝非 SSL 连接并返回以下消息：

```
$ export PGSSLMODE=disable
$ psql postgres -h SOMEHOST.amazonaws.com -p 8192 -U someuser
psql: FATAL: no pg_hba.conf entry for host "host.ip", user "someuser", database
 "postgres", SSL off
$
```

有关 sslmode 选项的信息，请参阅 PostgreSQL 文档中的数据库连接控制函数。

配置密码套件以连接到 Aurora PostgreSQL 数据库集群

通过使用可配置的密码套件，您可以更好地控制数据库连接的安全性。您可以指定想要允许安全客户端 SSL/TLS 与数据库连接的密码套件列表。使用可配置的密码套件，您现在可以控制数据库服务器接受的连接加密。这样做有助于防止使用不安全已或弃用的密码。

Aurora PostgreSQL 版本 11.8 及更高版本支持可配置的密码套件。

要指定加密连接的允许密码列表，请修改 `ssl_ciphers` 集群参数。使用 AWS Management Console、AWS CLI 或 RDS API 在集群参数组中将 `ssl_ciphers` 参数设置为逗号分隔的密码值字符串。要设置集群参数，请参阅 在 Amazon Aurora 中修改数据库集群参数组中的参数。

下表显示了有效 Aurora PostgreSQL 引擎版本支持的密码。

Aurora PostgreSQL 引擎版本	支持的密码	TLS 1.1	TLS 1.2	TLS 1.3
9.6、10.20 及更低版本、11.15 及更低版本、12.10 及更低版本、13.6 及更低版本	DHE-RSA-AES128-SHA	是	否	否
	DHE-RSA-AES128-SHA256	否	是	否
	DHE-RSA-AES128-GCM-SHA256	否	是	否
	DHE-RSA-AES256-SHA	否	是	否
	DHE-RSA-AES256-SHA256	是	是	否
	DHE-RSA-AES256-GCM-SHA384	否	是	否
	ECDHE-ECDSA-AES256-SHA	否	是	否
	ECDHE-ECDSA-AES256- GCM-SHA384	否	是	否
	ECDHE-RSA-AES256-SHA384	是	是	否
	ECDHE-RSA-AES128-SHA	否	是	否

Aurora PostgreSQL 引擎版本	支持的密码	TLS 1.1	TLS 1.2	TLS 1.3
	ECDHE-RSA-AES128-SHA256 ECDHE-RSA-AES128- GCM-SHA256 ECDHE-RSA-AES256-SHA ECDHE-RSA-AES256- GCM-SHA384			

Aurora PostgreSQL 引擎版本	支持的密码	TLS 1.1	TLS 1.2	TLS 1.3
10.21、11.16、12.11、13.7、14.3 和 14.4	ECDHE-RSA-AES128-SHA	是	是	否
	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	否	是	否
	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	是	是	否
	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	否	是	否
	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	是	是	否
	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	否	是	否
	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	是	是	否
	TLS_ECDHE_RSA_WITH	否	是	否

Aurora PostgreSQL 引擎版本	支持的密码	TLS 1.1	TLS 1.2	TLS 1.3
	_AES_256_ GCM_SHA384 TLS_RSA_W ITH_AES_2 56_GCM_SHA384 TLS_RSA_W ITH_AES_2 56_CBC_SHA TLS_RSA_W ITH_AES_1 28_GCM_SHA256 TLS_RSA_W ITH_AES_1 28_CBC_SHA TLS_ECDHE _RSA_WITH _CHACHA20 _POLY1305_SHA256			

Aurora PostgreSQL 引擎版本	支持的密码	TLS 1.1	TLS 1.2	TLS 1.3
10.22、11.17、12.12、13.8、14.5 和 15.2	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	是	是	否
		否	是	否
	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	否	是	否
		是	是	否
		否	是	否
	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	是	是	否
		否	是	否
		否	是	否
	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	是	是	否
		否	是	否
	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	否	是	否
		是	是	否
		否	是	否
	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	是	是	否
		是	是	否
	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	否	是	否
	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256			

Aurora PostgreSQL 引擎版本	支持的密码	TLS 1.1	TLS 1.2	TLS 1.3
	TLS_ECDHE _RSA_WITH _AES_256_CBC_SHA TLS_ECDHE _RSA_WITH _AES_256_ GCM_SHA384 TLS_RSA_W ITH_AES_2 56_GCM_SHA384 TLS_RSA_W ITH_AES_2 56_CBC_SHA TLS_RSA_W ITH_AES_1 28_GCM_SHA256 TLS_RSA_W ITH_AES_1 28_CBC_SHA256 TLS_RSA_W ITH_AES_1 28_CBC_SHA TLS_ECDHE _RSA_WITH _CHACHA20 _POLY1305_SHA256			

Aurora PostgreSQL 引擎版本	支持的密码	TLS 1.1	TLS 1.2	TLS 1.3
11.20、12.15、13.11、14.8、15.3、16.1 及更高版本	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	是	是	否
	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	否	是	否
	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	否	是	否
	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	是	是	否
	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	否	是	否
	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	是	是	否
	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	否	是	否
	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	否	否	是

Aurora PostgreSQL 引擎版本	支持的密码	TLS 1.1	TLS 1.2	TLS 1.3
	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 TLS_RSA_WITH_AES_256_GCM_SHA384 TLS_RSA_WITH_AES_256_CBC_SHA TLS_RSA_WITH_AES_128_GCM_SHA256 TLS_RSA_WITH_AES_128_CBC_SHA256 TLS_RSA_WITH_AES_128_CBC_SHA TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 TLS_AES_128_GCM_SHA256			

Aurora PostgreSQL 引擎版本	支持的密码	TLS 1.1	TLS 1.2	TLS 1.3
	TLS_AES_256_GCM_SHA384			

您也可以使用 describe-engine-default-cluster-parameters CLI 命令来确定特定参数组系列当前支持哪些密码套件。以下示例展示如何获取 Aurora PostgreSQL 11 的 `ssl_cipher` 集群参数组允许的值。

```
aws rds describe-engine-default-cluster-parameters --db-parameter-group-family aurora-
postgresql11

    ...some output truncated...
 {
  "ParameterName": "ssl_ciphers",
  "Description": "Sets the list of allowed TLS ciphers to be used on secure
 connections.",
  "Source": "engine-default",
  "ApplyType": "dynamic",
  "DataType": "list",
  "AllowedValues": "DHE-RSA-AES128-SHA,DHE-RSA-AES128-SHA256,DHE-RSA-AES128-GCM-
SHA256,DHE-RSA-AES256-SHA,DHE-RSA-AES256-SHA256,DHE-RSA-AES256-GCM-SHA384,
  ECDHE-RSA-AES128-SHA,ECDHE-RSA-AES128-SHA256,ECDHE-RSA-AES128-GCM-
SHA256,ECDHE-RSA-AES256-SHA,ECDHE-RSA-AES256-SHA384,ECDHE-RSA-AES256-GCM-
SHA384,TLS_RSA_WITH_AES_256_GCM_SHA384,

 TLS_RSA_WITH_AES_256_CBC_SHA,TLS_RSA_WITH_AES_128_GCM_SHA256,TLS_RSA_WITH_AES_128_CBC_SHA256,T

 TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA,TLS_ECDHE_RSA_WITH_AE
  "IsModifiable": true,
  "MinimumEngineVersion": "11.8",
  "SupportedEngineModes": [
   "provisioned"
  ]
 },
    ...some output truncated...
```

`ssl_ciphers` 参数原定设置为所有允许的密码套件。有关密码的更多信息，请参阅 PostgreSQL 文档中的 ssl_ciphers 变量。

使用新 SSL/TLS 证书更新应用程序，以连接到 Aurora PostgreSQL 数据库集群

自 2023 年 1 月 13 日起，Amazon RDS 发布了新的证书颁发机构（CA）证书，以便使用安全套接字层或传输层安全性协议（SSL/TLS）连接到 Aurora 数据库集群。接下来，您可以找到有关更新应用程序以使用新证书的信息。

本主题可帮助您确定是否有任何客户端应用程序使用 SSL/TLS 连接到您的数据库集群。如果是这样，您可以进一步检查这些应用程序是否需要证书验证才能连接。

> **Note**
>
> 某些应用程序被配置为仅在它们可以成功验证服务器上的证书时才连接到 Aurora PostgreSQL 数据库集群。
> 对于此类应用程序，您必须更新客户端应用程序信任存储，以包括新的 CA 证书。

更新客户端应用程序信任存储中的 CA 证书后，可以在数据库集群上轮换这些证书。强烈建议在生产环境中实现这些过程之前，先在开发或测试环境中测试它们。

有关证书轮换的更多信息，请参阅轮换 SSL/TLS 证书。有关下载证书的更多信息，请参阅使用 SSL/TLS 加密与数据库集群的连接。有关将 SSL/TLS 与 PostgreSQL 数据库集群配合使用的信息，请参阅利用 SSL/TLS 保护 Aurora PostgreSQL 数据。

主题

- 确定是否有应用程序正使用 SSL 连接到 Aurora PostgreSQL 数据库集群
- 确定客户端是否需要证书验证才能连接
- 更新应用程序信任存储
- 对不同类型的应用程序使用 SSL/TLS 连接

确定是否有应用程序正使用 SSL 连接到 Aurora PostgreSQL 数据库集群

检查数据库集群配置中 `rds.force_ssl` 参数的值。默认情况下，`rds.force_ssl` 参数设置为 `0`（关）。如果 `rds.force_ssl` 参数设置为 `1`（开），则客户端需要使用 SSL/TLS 进行连接。有关参数组的更多信息，请参阅 Amazon Aurora 的参数组。

如果 rds.force_ssl 未设置为 1（开），则查询 pg_stat_ssl 视图以检查使用 SSL 的连接。例如，以下查询仅返回 SSL 连接和有关使用 SSL 的客户端的信息。

```
select datname, usename, ssl, client_addr from pg_stat_ssl inner join pg_stat_activity
 on pg_stat_ssl.pid = pg_stat_activity.pid where ssl is true and usename<>'rdsadmin';
```

只有使用 SSL/TLS 连接的行才显示有关连接的信息。下面是示例输出。

```
 datname  | usename | ssl | client_addr
----------+---------+-----+-------------
 benchdb  | pgadmin | t   | 53.95.6.13
 postgres | pgadmin | t   | 53.95.6.13
(2 rows)
```

上述查询仅显示进行查询时的有效连接。没有结果并不表示没有应用程序在使用 SSL 连接。其他 SSL 连接可能在不同的时间建立。

确定客户端是否需要证书验证才能连接

当客户端（如 psql 或 JDBC）配置有 SSL 支持时，默认情况下，该客户端会首先尝试使用 SSL 连接到数据库。如果该客户端无法使用 SSL 进行连接，它将恢复为不使用 SSL 进行连接。用于基于 libpq 的客户端（例如 psql）和 JDBC 的默认 sslmode 模式设置为 prefer。仅当提供 sslrootcert 且 sslmode 设置为 verify-ca 或 verify-full 时，才会验证服务器上的证书。如果证书无效，则会引发错误。

使用 PGSSLROOTCERT 验证具有 PGSSLMODE 环境变量的证书，且 PGSSLMODE 设置为 verify-ca 或 verify-full。

```
PGSSLMODE=verify-full PGSSLROOTCERT=/fullpath/ssl-cert.pem psql -h
 pgdbidentifier.cxxxxxxxx.us-east-2.rds.amazonaws.com -U primaryuser -d postgres
```

使用 sslrootcert 参数验证 sslmode 为连接字符串格式的证书，且 sslmode 设置为 verify-ca 或 verify-full。

```
psql "host=pgdbidentifier.cxxxxxxxx.us-east-2.rds.amazonaws.com sslmode=verify-full
 sslrootcert=/full/path/ssl-cert.pem user=primaryuser dbname=postgres"
```

例如，在上述情况下，如果您使用无效的根证书，则会在客户端上看到类似于以下内容的错误。

```
psql: SSL error: certificate verify failed
```

更新应用程序信任存储

有关更新 PostgreSQL 应用程序的信任存储的信息，请参阅 PostgreSQL 文档中的使用 SSL 保护 TCP/IP 连接。

> **Note**
>
> 更新信任存储时，除了添加新证书外，还可以保留较旧证书。

更新 JDBC 应用程序信任存储

您可以更新使用 JDBC 的应用程序的信任存储以进行 SSL/TLS 连接。

有关下载根证书的信息，请参阅 使用 SSL/TLS 加密与数据库集群的连接。

有关导入证书的示例脚本，请参阅 将证书导入信任存储的示例脚本。

对不同类型的应用程序使用 SSL/TLS 连接

下面提供了有关对不同类型应用程序使用 SSL/TLS 连接的信息：

- psql

  通过将选项指定为连接字符串或环境变量，可以从命令行调用客户端。对于 SSL/TLS 连接，相关选项为 `sslmode`（环境变量 `PGSSLMODE`）和 `sslrootcert`（环境变量 `PGSSLROOTCERT`）。

  有关完整的选项列表，请参阅 PostgreSQL 文档中的参数关键字。有关完整的环境变量列表，请参阅 PostgreSQL 文档中的环境变量。
- pgAdmin

  这个基于浏览器的客户端是一个更加用户友好的界面，用于连接到 PostgreSQL 数据库。

  有关配置连接的信息，请参阅 pgAdmin 文档。
- JDBC

  JDBC 支持与 Java 应用程序的数据库连接。

  有关使用 JDBC 连接到 PostgreSQL 数据库的一般信息，请参阅 PostgreSQL 文档中的连接到数据库。有关使用 SSL/TLS 进行连接的信息，请参阅 PostgreSQL 文档中的配置客户端。
- Python

一个常用的连接到 PostgreSQL 数据库的 Python 库是 psycopg2。

有关如何使用 psycopg2 的信息，请参阅 psycopg2 文档。有关如何连接到 PostgreSQL 数据库的简短教程，请参阅 Psycopg2 教程。您可以在 psycopg2 模块内容中找到有关连接命令接受的选项的信息。

> **Important**
>
> 在确定了数据库连接使用 SSL/TLS 并更新了应用程序信任存储之后，可以更新数据库以使用 rds-ca-rsa2048-g1 证书。有关说明，请参阅通过修改数据库实例来更新 CA 证书中的步骤 3。

在 Aurora PostgreSQL 中使用 Kerberos 身份验证

在用户连接到运行 PostgreSQL 的数据库集群时，您可以使用 Kerberos 对用户进行身份验证。为此，请将数据库集群配置为使用 AWS Directory Service for Microsoft Active Directory 进行 Kerberos 身份验证。AWS Directory Service for Microsoft Active Directory 也称为 AWS Managed Microsoft AD。这是 AWS Directory Service 提供的一项功能。要了解更多信息，请参阅《AWS Directory Service 管理指南》中的什么是 AWS Directory Service？

要开始操作，请创建一个 AWS Managed Microsoft AD 目录来存储用户凭证。然后，将 Active Directory 的域和其他信息提供给 PostgreSQL 数据库集群。当用户使用 PostgreSQL 数据库集群进行身份验证时，身份验证请求将转发到 AWS Managed Microsoft AD 目录。

将所有凭证保存在同一目录中可以节省您的时间和精力。您具有一个集中位置用于存储和管理多个数据库集群的凭证。使用目录还可以改善您的整体安全概要。

此外，还可以从自己的本地 Microsoft Active Directory 访问凭证。为此，请创建一个信任域关系，以便 AWS Managed Microsoft AD 目录信任您的本地 Microsoft Active Directory。通过这种方式，您的用户可以使用 Windows 单点登录 (SSO) 访问 PostgreSQL 集群，获得与访问本地网络中的工作负载相同的体验。

数据库可以使用 Kerberos、AWS Identity and Access Management (IAM)，或同时使用 Kerberos 和 IAM 身份验证。但是，由于 Kerberos 和 IAM 身份验证提供了不同的身份验证方法，因此，特定数据库用户只能使用一种或另一种身份验证方法登录数据库，但不能同时使用这两种方法。有关 IAM 身份验证的更多信息，请参阅 IAM 数据库身份验证。

主题

- 区域和版本可用性
- PostgreSQL 数据库集群的 Kerberos 身份验证概述
- 为 PostgreSQL 数据库集群设置 Kerberos 身份验证
- 在 Active Directory 域中管理 Aurora PostgreSQL 数据库集群
- 使用 Kerberos 身份验证连接到 PostgreSQL
- 使用 AD 安全组进行 Aurora PostgreSQL 访问控制

区域和版本可用性

功能可用性和支持因每个数据库引擎的特定版本以及 AWS 区域而异。有关使用 Kerberos 身份验证的 Aurora PostgreSQL 的版本和区域可用性的更多信息，请参阅 适用于 Aurora PostgreSQL 的 Kerberos 身份验证。

PostgreSQL 数据库集群的 Kerberos 身份验证概述

要为 PostgreSQL 数据库集群设置 Kerberos 身份验证，请执行以下步骤，稍后将详细介绍这些步骤：

1. 使用 AWS Managed Microsoft AD 创建 AWS Managed Microsoft AD 目录。您可以使用 AWS Management Console、AWS CLI 或 AWS Directory Service API 创建目录。确保打开目录安全组上的相关出站端口，以便目录可以与集群进行通信。
2. 创建一个角色，以提供对您的 AWS Managed Microsoft AD 目录进行调用的 Amazon Aurora 访问权限。为此，请创建使用托管 IAM 策略 `AmazonRDSDirectoryServiceAccess` 的 AWS Identity and Access Management (IAM) 角色。

   要使 IAM 角色允许访问，必须在您的 AWS 账户的正确 AWS 区域中激活 AWS Security Token Service（AWS STS）端点。原定设置情况下，AWS STS 端点在所有 AWS 区域中处于活动状态，您可以直接使用这些端点，而无需执行任何其他操作。有关更多信息，请参阅 IAM 用户指南中的在 AWS STS 区域中激活和停用 AWS。
3. 使用 Microsoft Active Directory 工具在 AWS Managed Microsoft AD 目录中创建和配置用户。有关在 Active Directory 中创建用户的更多信息，请参阅 AWS 管理指南中的在 AWS Directory Service 托管 Microsoft AD 中管理用户和组。
4. 如果您计划在不同的 AWS 账户或 Virtual Private Cloud (VPC) 中查找目录和数据库实例，请配置 VPC 对等连接。有关更多信息，请参阅 Amazon VPC Peering Guide 中的什么是 VPC 对等连接？。
5. 使用以下方法之一，从控制台、CLI 或 RDS API 中创建或修改 PostgreSQL 数据库集群：
   - 创建 Aurora PostgreSQL 数据库集群并连接到该集群

- 修改 Amazon Aurora 数据库集群
- 从数据库集群快照还原
- 将数据库集群还原到指定时间

您可以在与目录相同的 Amazon Virtual Private Cloud (VPC) 中或在不同的 AWS 账户或 VPC 中定位集群。创建或修改 PostgreSQL 数据库集群时，请执行以下操作：

- 请提供创建目录时生成的域标识符（d-* 标识符）。
- 还要提供您创建的 IAM 角色的名称。
- 确保数据库实例安全组可以从目录安全组接收入站流量。

6. 使用 RDS 主用户凭证连接到 PostgreSQL 数据库集群。在 PostgreSQL 中创建用户以在外部进行标识。外部标识的用户可以使用 Kerberos 身份验证登录到 PostgreSQL 数据库集群。

为 PostgreSQL 数据库集群设置 Kerberos 身份验证

您可以使用 AWS Directory Service for Microsoft Active Directory (AWS Managed Microsoft AD) 为 PostgreSQL 数据库集群设置 Kerberos 身份验证。要设置 Kerberos 身份验证，请执行以下步骤。

主题

- 步骤 1：使用 AWS Managed Microsoft AD 创建目录
- 步骤 2：（可选）在本地 Active Directory 和 AWS Directory Service 之间创建信任关系
- 步骤 3：为 Amazon Aurora 创建 IAM 角色以访问 AWS Directory Service
- 步骤 4：创建和配置用户
- 步骤 5：在目录和数据库实例之间启用跨 VPC 流量
- 步骤 6：创建或修改 PostgreSQL 数据库集群
- 步骤 7：为您的 Kerberos 主体创建 PostgreSQL 用户
- 步骤 8：配置 PostgreSQL 客户端

步骤 1：使用 AWS Managed Microsoft AD 创建目录

AWS Directory Service 将在 AWS 云中创建一个完全托管的 Active Directory。在创建 AWS Managed Microsoft AD 目录时，AWS Directory Service 将创建两个域控制器和 DNS 服务器。目录服务器在 VPC 中的不同子网中创建。这种冗余有助于确保始终可以访问目录，即使发生了故障。

创建 AWS Managed Microsoft AD 目录时，AWS Directory Service 将代表您执行下列任务：

- 在您的 VPC 中设置 Active Directory。
- 创建具有用户名 `Admin` 和指定密码的目录管理员账户。您可以使用此账户管理您的目录。

> **Important**
>
> 请务必保存此密码。AWS Directory Service 不会存储此密码并且它无法取回或重置。

- 为目录控制器创建安全组。安全组必须允许与 PostgreSQL 数据库集群进行通信。

在启动 AWS Directory Service for Microsoft Active Directory 时，AWS 创建一个组织单位 (OU)，其中包含目录的所有对象。此 OU（具有您在创建目录时输入的 NetBIOS 名称）位于域根目录中。此域根目录由 AWS 拥有和管理。

使用 `Admin` 目录创建的 AWS Managed Microsoft AD 账户有权为您的 OU 执行最常见的管理活动：

- 创建、更新或删除用户
- 将资源添加到域（如文件或打印服务器），然后为 OU 中的用户分配这些资源的权限
- 创建额外的 OU 和容器
- 委托授权
- 从 Active Directory 回收站还原删除的对象
- 在 Active Directory Web 服务上为 Windows PowerShell 运行 Active Directory 和域名服务 (DNS) 模块

`Admin` 账户还有权执行以下域范围的活动：

- 管理 DNS 配置（添加、删除或更新记录、区域和转发器）
- 查看 DNS 事件日志
- 查看安全事件日志

使用 AWS Managed Microsoft AD 创建目录

1. 在 AWS Directory Service 控制台导航窗格中，选择 Directories (目录)，然后选择 Set up directory (设置目录)。
2. 选择 AWS Managed Microsoft AD。AWS Managed Microsoft AD 是当前唯一支持用于 Amazon Aurora 的选项。

3. 选择下一步。
4. 在输入目录信息页面上，提供以下信息：

   版本

   选择符合您要求的版本。

   目录 DNS 名称

   目录的完全限定名称，例如 **corp.example.com**。

   目录 NetBIOS 名称

   可选的目录短名称，如 CORP。

   目录描述

   目录的可选描述。

   管理员密码

   目录管理员的密码。目录创建过程将创建一个具有 Admin 用户名和此密码的管理员账户。

   目录管理员密码不能包含单词“admin”。此密码区分大小写，且长度必须介于 8 – 64 个字符之间。至少，它还必须包含下列四种类别中三种类别的一个字符：

   - 小写字母 (a–z)
   - 大写字母 (A–Z)
   - 数字 (0–9)
   - 非字母数字字符 (~!@#$%^&*_-+=`|\(){}[]:;'"<>,.?/)

   确认密码

   重新键入管理员密码。

   > **Important**
   >
   > 确保您保存该密码。AWS Directory Service 不会存储该密码，也无法检索或重置该密码。

5. 选择下一步。
6. 在选择 VPC 和子网页面上，提供以下信息：

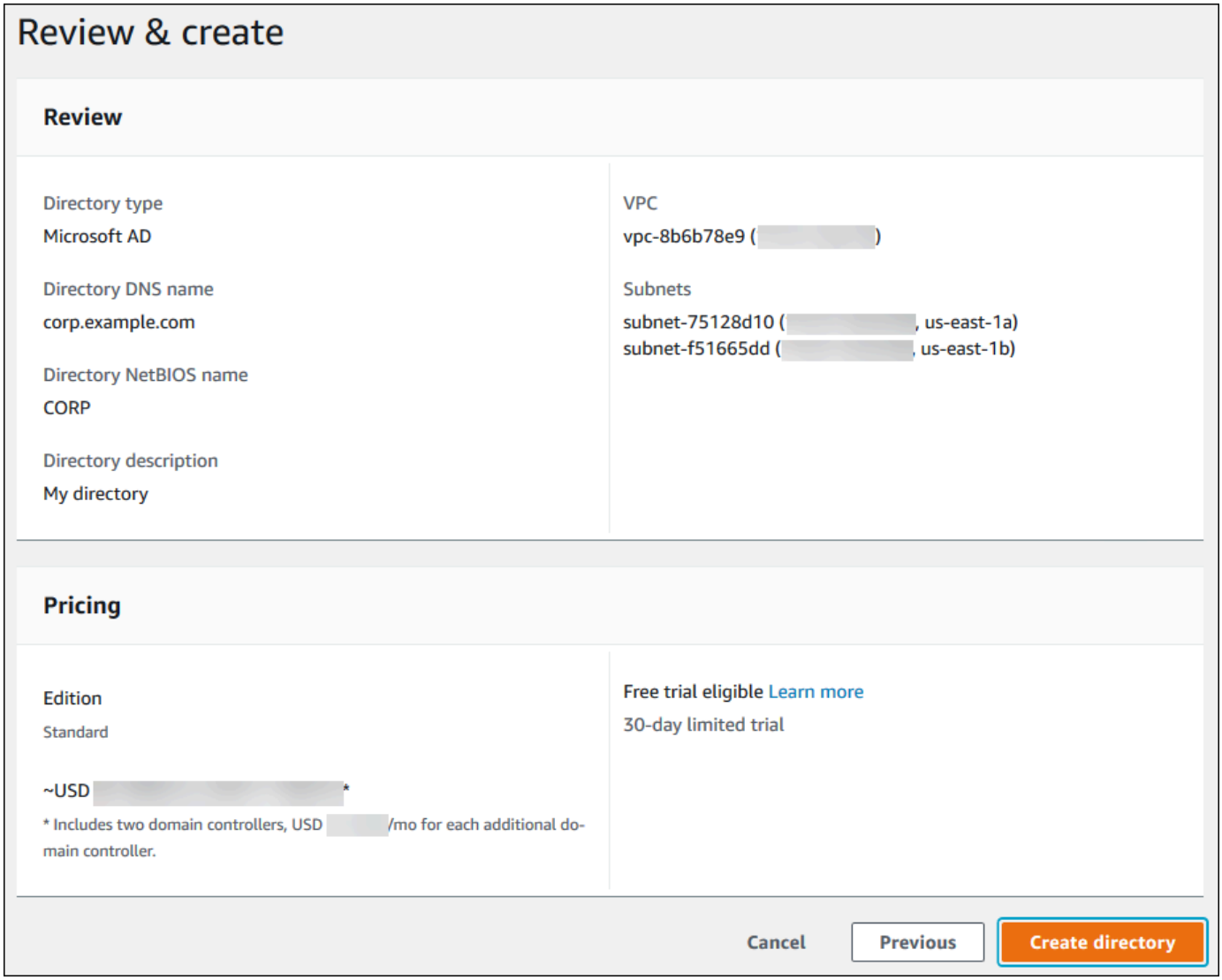
VPC

为目录选择 VPC。您可以在该相同 VPC 或不同的 VPC 中创建 PostgreSQL 数据库集群。

子网

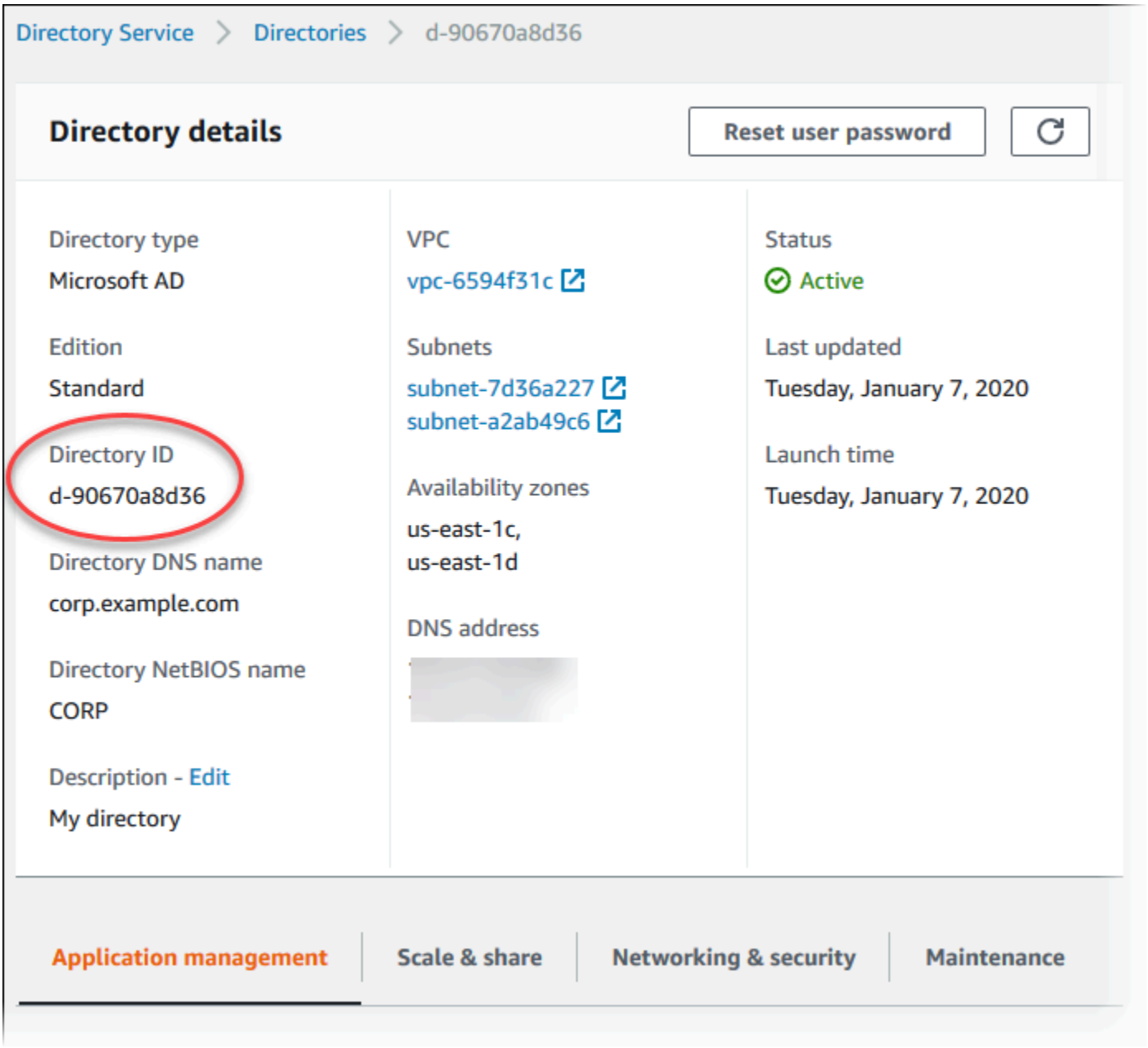
为目录服务器选择子网。两个子网必须位于不同的可用区。

7. 选择下一步。
8. 检查目录信息。如果需要进行更改，请选择 Previous (上一步) 并作出更改。如果信息正确，请选择 Create directory (创建目录)。



目录创建需要几分钟时间。创建成功后，Status (状态) 值将更改为 Active (活动)。

要查看有关您的目录的信息，请在目录列表中选择目录。记下目录 ID 值。在创建或修改 PostgreSQL 数据库实例时，您需要使用该值。



步骤 2：（可选）在本地 Active Directory 和 AWS Directory Service 之间创建信任关系

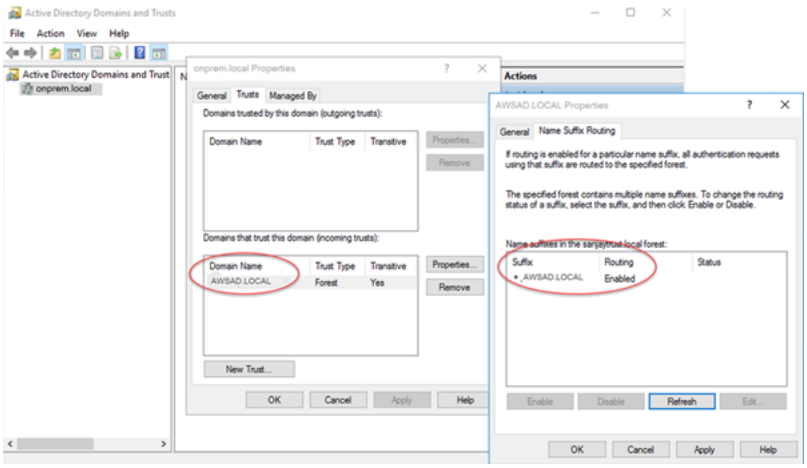
如果您不打算使用自己的本地 Microsoft Active Directory，请跳转至 步骤 3：为 Amazon Aurora 创建 IAM 角色以访问 AWS Directory Service。

要使用本地 Active Directory 获取 Kerberos 身份验证，您需要使用林信任在本地 Microsoft Active Directory 和 AWS Managed Microsoft AD 目录（在 步骤 1：使用 AWS Managed Microsoft AD 创建目录 中创建）之间创建信任域关系。信任可以是单向的，此时 AWS Managed Microsoft AD 目录信任本

地 Microsoft Active Directory。信任也可以是双向的，此时两个 Active Directory 相互信任。有关使用 AWS Directory Service 设置信任的更多信息，请参阅 AWS Directory Service 管理指南中的何时创建信任关系。

> **Note**
>
> 如果使用本地 Microsoft Active Directory：
>
> - Windows 客户端必须使用端点中 AWS Directory Service 的域名而不是 rds.amazonaws.com 进行连接。有关更多信息，请参阅 使用 Kerberos 身份验证连接到 PostgreSQL。
> - Windows 客户端无法使用 Aurora 自定义端点进行连接。要了解更多信息，请参阅Amazon Aurora 端点连接。
> - 对于全局数据库：
>   - Windows 客户端可以使用全局数据库的主 AWS 区域 中的实例端点或集群端点进行连接。
>   - Windows 客户端无法使用辅助 AWS 区域 中的集群端点进行连接。

请确保您的本地 Microsoft Active Directory 域名包含与新创建的信任关系对应的 DNS 后缀路由。以下屏幕截图显示一个示例。



步骤 3：为 Amazon Aurora 创建 IAM 角色以访问 AWS Directory Service

要使 Amazon Aurora 为您调用 AWS Directory Service，您的 AWS 账户需要一个使用托管式 IAM 策略 `AmazonRDSDirectoryServiceAccess` 的 IAM 角色。该角色允许 Amazon Aurora 对 AWS Directory Service 进行调用。（请注意，访问 AWS Directory Service 的 IAM 角色不同于用于 IAM 数据库身份验证 的 IAM 角色。）

当您使用 AWS Management Console 创建数据库实例并且控制台用户账户具有 `iam:CreateRole` 权限时，控制台将自动创建所需的 IAM 角色。在这种情况下，角色名为 `rds-directoryservice-kerberos-access-role`。否则，您必须手动创建 IAM 角色。在创建该 IAM 角色时，请选择 `Directory Service`，然后将 AWS 托管策略 `AmazonRDSDirectoryServiceAccess` 附加到该角色。

有关为服务创建 IAM 角色的更多信息，请参阅 IAM 用户指南中的创建向AWS服务委托权限的角色。

> **Note**
>
> 用于 RDS for Microsoft SQL Server 的 Windows 身份验证的 IAM 角色不能用于 Amazon Aurora。

作为使用 `AmazonRDSDirectoryServiceAccess` 托管式策略的替代方法，您可以创建具有所需权限的策略。在这种情况下，IAM 角色必须具有以下 IAM 信任策略。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "directoryservice.rds.amazonaws.com",
          "rds.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

角色还必须具有以下 IAM 角色策略。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "ds:DescribeDirectories",
        "ds:AuthorizeApplication",
        "ds:UnauthorizeApplication",
        "ds:GetAuthorizedApplicationDetails"
      ],
    "Effect": "Allow",
    "Resource": "*"
    }
  ]
}
```

对于选择加入的 AWS 区域，请在 IAM 角色信任策略中使用区域特定的服务主体。在为这些区域中的服务创建信任策略时，请在服务主体中指定区域代码。

以下示例显示了包括区域特定的服务主体的信任策略：

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "directoryservice.rds.REGION-CODE.amazonaws.com",
          "rds.REGION-CODE.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

将 REGION-CODE 替换为特定区域的代码。例如，对亚太地区（墨尔本）区域使用以下服务主体：

```
"Service": [
  "directoryservice.rds.ap-southeast-4.amazonaws.com",
  "rds.ap-southeast-4.amazonaws.com"
]
```

步骤 4：创建和配置用户

您可以使用“Active Directory 用户和计算机”工具创建用户。该工具是 Active Directory Domain Services 和 Active Directory Lightweight Directory Services 工具之一。有关更多信息，请参阅 Microsoft 文档中的将用户和计算机添加到 Active Directory 域。在这种情况下，用户是个人或其他实体，例如属于域的计算机，其身份在目录中维护。

要在 AWS Directory Service 目录中创建用户，您必须连接到属于 AWS Directory Service 目录成员的基于 Windows 的 Amazon EC2 实例。同时，您必须以具有创建用户权限的用户身份登录。有关更多信息，请参阅 AWS Directory Service 管理指南中的创建用户。

步骤 5：在目录和数据库实例之间启用跨 VPC 流量

如果您打算将目录和数据库集群放在同一 VPC 中，请跳过该步骤，然后转到 步骤 6：创建或修改 PostgreSQL 数据库集群。

如果您计划在不同 VPC 中查找目录和数据库实例，请使用 VPC 对等连接或 AWS Transit Gateway 配置跨 VPC 流量。

以下过程使用 VPC 对等连接启用 VPC 之间的流量。请按照 Amazon Virtual Private Cloud 对等连接指南中的什么是 VPC 对等连接？操作。

使用 VPC 对等连接启用跨 VPC 流量

1. 设置适合的 VPC 路由规则，以便确保网络流量可以双向流动。
2. 确保数据库实例安全组可以从目录安全组接收入站流量。
3. 确保没有任何网络访问控制列表 (ACL) 规则会阻止流量。

如果该目录由不同的 AWS 账户拥有，则您必须共享该目录。

在 AWS 账户之间共享目录

1. 按照 AWS 管理指南中的教程：共享 AWS 托管 Microsoft AD 目录以实现无缝 EC2 域加入中的说明，开始与将要在其中创建数据库实例的 AWS Directory Service 账户共享目录。
2. 使用数据库实例的账户登录到 AWS Directory Service 控制台，并确保在处理之前域具有 SHARED 状态。
3. 使用数据库实例的账户登录 AWS Directory Service 控制台时，请记录目录 ID 值。您可以使用此目录 ID 将数据库实例加入域。

步骤 6：创建或修改 PostgreSQL 数据库集群

创建或修改 PostgreSQL 数据库集群以用于您的目录。您可以使用控制台、CLI 或 RDS API 将数据库集群与目录关联。您可以通过下列方式之一来执行该操作：

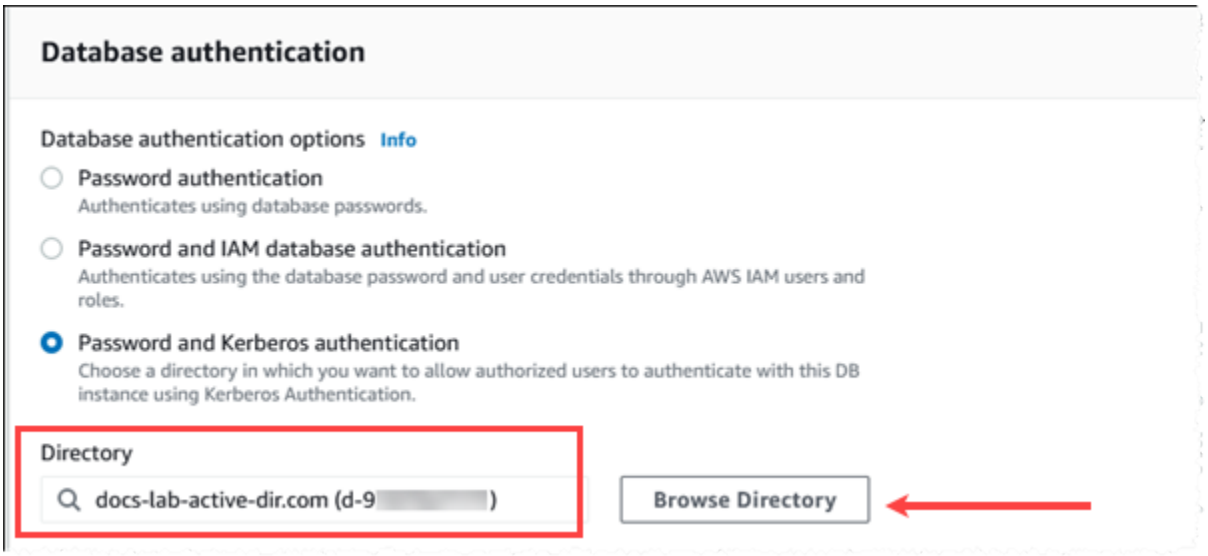
- 使用控制台、create-db-cluster CLI 命令或 CreateDBCluster RDS API 操作创建新的 PostgreSQL 数据库集群。有关说明，请参阅 创建 Aurora PostgreSQL 数据库集群并连接到该集群。
- 使用控制台、modify-db-cluster CLI 命令或 ModifyDBCluster RDS API 操作修改现有的 PostgreSQL 数据库集群。有关说明，请参阅 修改 Amazon Aurora 数据库集群。
- 使用控制台、restore-db-cluster-from-db-snapshot CLI 命令或 RestoreDBClusterFromDBSnapshot RDS API 操作，从数据库快照还原 PostgreSQL 数据库集群。有关说明，请参阅从数据库集群快照还原。
- 使用控制台、restore-db-instance-to-point-in-time CLI 命令或 RestoreDBClusterToPointInTime RDS API 操作，将 PostgreSQL 数据库集群还原到某个时间点。有关说明，请参阅将数据库集群还原到指定时间。

仅 VPC 中的 PostgreSQL 数据库集群支持 Kerberos 身份验证。数据库集群可以与目录在同一 VPC 中或在不同 VPC 中。数据库集群必须使用允许在目录的 VPC 中传入和传出的安全组，以便数据库集群与目录通信。

> **Note**
>
> 在从 RDS for PostgreSQL 迁移期间，Aurora PostgreSQL 数据库集群目前不支持启用 Kerberos 身份验证。您只能在独立的 Aurora PostgreSQL 数据库集群上启用 Kerberos 身份验证。

控制台

在使用控制台创建、修改或还原数据库集群时，请选择数据库身份验证部分中的 Kerberos 身份验证。然后选择浏览目录。选择目录或选择创建一个新目录以使用 Directory Service。



AWS CLI

使用 AWS CLI 时，数据库集群需要以下参数才能使用您创建的目录：

- 对于 `--domain` 参数，请使用创建目录时生成的域标识符（“d-*”标识符）。
- 对于 `--domain-iam-role-name` 参数，请使用您使用托管 IAM 策略 `AmazonRDSDirectoryServiceAccess` 创建的角色。

例如，以下 CLI 命令会修改数据库集群以使用目录。

```
aws rds  modify-db-cluster --db-cluster-identifier mydbinstance --domain d-Directory-ID
 --domain-iam-role-name role-name
```

> **Important**
>
> 如果修改数据库集群以启用 Kerberos 身份验证，请在做出更改之后重新启动数据库集群。

步骤 7：为您的 Kerberos 主体创建 PostgreSQL 用户

此时，您的 Aurora PostgreSQL 数据库集群已加入 AWS Managed Microsoft AD 域。您需要将您在步骤 4：创建和配置用户 的目录中创建的用户设置为 PostgreSQL 数据库用户，并授予登录数据库

的权限。为此，请以具有 rds_superuser 权限的数据库用户身份登录。例如，如果在创建 Aurora PostgreSQL 数据库集群时接受了缺省设置值，请使用 postgres，如以下步骤所示。

为 Kerberos 主体创建 PostgreSQL 数据库用户

1. 使用 psql 通过 psql 连接到 Aurora PostgreSQL 数据库集群的 数据库实例端点。以下示例对于 rds_superuser 角色使用原定设置 postgres 账户。

```
psql --host=cluster-instance-1.111122223333.aws-region.rds.amazonaws.com --
port=5432 --username=postgres --password
```

2. 为要访问数据库的每个 Kerberos 主体（Active Directory 用户名）创建一个数据库用户名。使用 Active Directory 实例中定义的规范用户名（身份），即该用户名的小写 alias（Active Directory 中的用户名）和大写 Active Directory 域名。Active Directory 用户名是经过外部身份验证的用户，因此请在名称前后使用引号，如下所示。

```
postgres=> CREATE USER "username@CORP.EXAMPLE.COM" WITH LOGIN;
CREATE ROLE
```

3. 将 rds_ad 角色授予数据库用户。

```
postgres=> GRANT rds_ad TO "username@CORP.EXAMPLE.COM";
GRANT ROLE
```

为 Active Directory 用户身份创建完所有 PostgreSQL 用户后，用户可以使用他们的 Kerberos 凭证访问 Aurora PostgreSQL 数据库集群。

使用 Kerberos 进行身份验证的数据库用户需要从属于 Active Directory 域成员的客户端计算机进行身份验证。

被授予 rds_ad 角色的数据库用户不能同时拥有 rds_iam 角色。这也适用于嵌套成员资格。有关更多信息，请参阅 IAM 数据库身份验证。

将 Aurora PostgreSQL 数据库集群配置为使用不区分大小写的用户名

Aurora PostgreSQL 版本 14.5、13.8、12.12 和 11.17 支持 krb_caseins_users PostgreSQL 参数。此参数支持不区分大小写的 Active Directory 用户名。缺省情况下，此参数设置为 false，因此 Aurora PostgreSQL 会以区分大小写的方式解释用户名。这是 Aurora PostgreSQL 的所有较旧版本中的缺省设置行为。但是，您可在自定义数据库集群参数组中将此参数设置为 true，并允许 Aurora

PostgreSQL 数据库集群以不区分大小写的方式解释用户名。考虑这样做是为了方便数据库用户，他们在使用 Active Directory 进行身份验证时，有时可能会错误地键入用户名的大小写。

要更改 `krb_caseins_users` 参数，您的 Aurora PostgreSQL 数据库集群必须使用自定义数据库集群参数组。有关使用自定义数据库集群参数组的信息，请参阅 Amazon Aurora 的参数组。

您可以使用 AWS CLI 或 AWS Management Console 来更改设置。有关更多信息，请参阅 在 Amazon Aurora 中修改数据库集群参数组中的参数。

步骤 8：配置 PostgreSQL 客户端

要配置 PostgreSQL 客户端，请采取以下步骤：

- 创建一个 krb5.conf 文件（或等效的文件）以指向域。
- 验证流量是否可以在客户端主机和 AWS Directory Service 之间流动。使用网络实用程序（如 Netcat）执行以下操作：
  - 验证端口 53 上通过 DNS 的流量。
  - 验证端口 53 上通过 TCP/UDP 的流量以及 Kerberos 的流量，包括用于 AWS Directory Service 的端口 88 和 464。
- 验证流量是否可以通过数据库端口在客户端主机和数据库实例之间流动。例如，使用 psql 连接和访问数据库。

以下是 AWS Managed Microsoft AD 的示例 Krb5.conf 内容。

```
[libdefaults]
 default_realm = EXAMPLE.COM
[realms]
 EXAMPLE.COM = {
  kdc = example.com
  admin_server = example.com
 }
[domain_realm]
 .example.com = EXAMPLE.COM
 example.com = EXAMPLE.COM
```

以下是本地 Microsoft Active Directory 的示例 krb5.conf 内容。

```
[libdefaults]
 default_realm = EXAMPLE.COM
[realms]
```

```
 EXAMPLE.COM = {
  kdc = example.com
  admin_server = example.com
 }
 ONPREM.COM = {
  kdc = onprem.com
  admin_server = onprem.com
 }
[domain_realm]
 .example.com = EXAMPLE.COM
 example.com = EXAMPLE.COM
 .onprem.com = ONPREM.COM
 onprem.com = ONPREM.COM
 .rds.amazonaws.com = EXAMPLE.COM
 .amazonaws.com.cn = EXAMPLE.COM
 .amazon.com = EXAMPLE.COM
```

在 Active Directory 域中管理 Aurora PostgreSQL 数据库集群

可以使用控制台、CLI 或 RDS API 来管理数据库集群及其与 Microsoft Active Directory 的关系。例如，您可以关联 Active Directory 以启用 Kerberos 身份验证。您也可以删除 Active Directory 关联以禁用 Kerberos 身份验证。您也可以将由一个 Microsoft Active Directory 在外部进行身份验证的数据库集群移动到另一个 Active Directory。

例如，使用 CLI，您可以执行下列操作：

- 要再次尝试为失败的成员启用 Kerberos 身份验证，请使用 modify-db-cluster CLI 命令。为 `--domain` 选项指定当前成员的目录 ID。
- 要在数据库实例上禁用 Kerberos 身份验证，请使用 modify-db-cluster CLI 命令。为 `none` 选项指定 `--domain`。
- 要将数据库实例从一个域移动到另一个域，请使用 modify-db-cluster CLI 命令。为 `--domain` 选项指定新域的域标识符。

了解域成员资格

在创建或修改数据库集群后，数据库实例将成为域的成员。您可以在控制台中查看域成员身份状态，也可以通过运行 describe-db-instances CLI 命令来查看。数据库实例的状态可以是以下状态之一：

- `kerberos-enabled` – 数据库实例已启用 Kerberos 身份验证。
- `enabling-kerberos` – AWS是在此数据库实例上启用 Kerberos 身份验证的过程。

- `pending-enable-kerberos` – 启用 Kerberos 身份验证正在此数据库实例上等待处理。
- `pending-maintenance-enable-kerberos` – AWS将尝试在下一个计划的维护时段在数据库实例上启用 Kerberos 身份验证。
- `pending-disable-kerberos` – 禁用 Kerberos 身份验证正在此数据库实例上等待处理。
- `pending-maintenance-disable-kerberos` – AWS 将尝试在下一个计划的维护时段在数据库实例上禁用 Kerberos 身份验证。
- `enable-kerberos-failed` – 出现一个配置问题，导致 AWS 无法在数据库实例上启用 Kerberos 身份验证。在重新发出命令以修改数据库实例之前纠正配置问题。
- `disabling-kerberos` – AWS是在此数据库实例上启用 Kerberos 身份验证的过程。

启用 Kerberos 身份验证的请求可能因网络连接问题或不正确的 IAM 角色而失败。在某些情况下，在创建或修改数据库集群时，尝试启用 Kerberos 身份验证可能会失败。如果是这样，请确保使用正确的 IAM 角色，然后修改数据库集群以加入域。

使用 Kerberos 身份验证连接到 PostgreSQL

您可以使用 pgAdmin 界面或命令行界面（如 psql）通过 Kerberos 身份验证连接到 PostgreSQL。有关连接的更多信息，请参阅 连接到 Amazon Aurora PostgreSQL 数据库集群。有关获取连接所需的端点、端口号和其他详细信息的信息，请参阅查看 Aurora 集群的终端节点。

> **Note**
>
> PostgreSQL 中的 GSSAPI 身份验证和加密由 Kerberos 库 `libkrb5.so` 实施。`postgres_fdw` 和 `dblink` 等功能也依赖于同一个库，以通过 Kerberos 身份验证或加密进行出站连接。

pgAdmin

要使用 pgAdmin 通过 Kerberos 身份验证连接到 PostgreSQL，请执行以下步骤：

1. 在您的客户端计算机上启动 pgAdmin 应用程序。
2. 在 Dashboard (控制面板) 选项卡上，选择 Add New Server (添加新服务器)。
3. 在 Create - Server (创建 - 服务器) 对话框中，在 General (常规) 选项卡上键入名称以在 pgAdmin 中标识该服务器。
4. 在 Connection (连接) 选项卡上，键入您的 Aurora PostgreSQL 数据库的以下信息。

- 对于 Host（主机），输入 Aurora PostgreSQL 数据库集群的写入器实例 的端点。端点看起来类似于以下所示：

  ```
  AUR-cluster-instance.111122223333.aws-region.rds.amazonaws.com
  ```

  要从 Windows 客户端连接到本地 Microsoft Active Directory，请使用 AWS Managed Active Directory 的域名，而不是主机端点中的 `rds.amazonaws.com`。例如，假设 AWS Managed Active Directory 的域名为 `corp.example.com`。然后，对于 Host（主机），按如下方式指定端点：

  ```
  AUR-cluster-instance.111122223333.aws-region.corp.example.com
  ```

- 对于 Port (端口)，输入分配的端口。
- 对于 Maintenance database (维护数据库)，输入客户端将连接到的初始数据库的名称。
- 对于用户名，键入您在 步骤 7：为您的 Kerberos 主体创建 PostgreSQL 用户 中为 Kerberos 身份验证输入的用户名。

5. 选择 Save。

psql

要使用 psql 通过 Kerberos 身份验证连接到 PostgreSQL，请执行以下步骤：

1. 在命令提示符处，运行以下命令。

   ```
   kinit username
   ```

   将 *username* 替换为用户名。在提示符下，输入在 Microsoft Active Directory 中为用户存储的密码。

2. 如果 PostgreSQL 数据库集群使用可公开访问的 VPC，请将数据库集群端点的 IP 地址放在 EC2 客户端上的 `/etc/hosts` 文件中。例如，以下命令获取 IP 地址，然后将其放在 `/etc/hosts` 文件中。

   ```
   % dig +short PostgreSQL-endpoint.AWS-Region.rds.amazonaws.com
   ;; Truncated, retrying in TCP mode.
   ec2-34-210-197-118.AWS-Region.compute.amazonaws.com.
   34.210.197.118
   ```

```
% echo " 34.210.197.118  PostgreSQL-endpoint.AWS-Region.rds.amazonaws.com" >> /etc/
hosts
```

如果您从 Windows 客户端使用本地 Microsoft Active Directory，则需要使用特殊终端节点进行连接。在主机终端节点中使用 `rds.amazonaws.com` Managed Active Directory 的域名，而不是 Amazon 域 AWS。

例如，假设 AWS Managed Active Directory 的域名为 `corp.example.com`。为终端节点使用格式 *`PostgreSQL-endpoint`*`.`*`AWS-Region`*`.corp.example.com`，将其放入 `/etc/hosts` 文件。

```
% echo " 34.210.197.118  PostgreSQL-endpoint.AWS-Region.corp.example.com" >> /etc/
hosts
```

3. 使用以下 psql 命令登录到与 Active Directory 集成的 PostgreSQL 数据库集群。使用集群或实例终端节点。

```
psql -U username@CORP.EXAMPLE.COM -p 5432 -h PostgreSQL-endpoint.AWS-
Region.rds.amazonaws.com postgres
```

要从 Windows 客户端使用本地 Active Directory 登录 PostgreSQL 数据库集群，请使用以下 psql 命令以及上一步中的域名 (`corp.example.com`)：

```
psql -U username@CORP.EXAMPLE.COM -p 5432 -h PostgreSQL-endpoint.AWS-
Region.corp.example.com postgres
```

使用 AD 安全组进行 Aurora PostgreSQL 访问控制

从 Aurora PostgreSQL 14.10 和 15.5 版本起，可以使用 AWS Directory Service for Microsoft Active Directory（AD）安全组来管理 Aurora PostgreSQL 访问控制。早期版本的 Aurora PostgreSQL 仅支持单个用户使用 AD 进行基于 Kerberos 的身份验证。每个 AD 用户都必须显式预调配到数据库集群，才能获得访问权限。

您可以按如下所述利用 AD 安全组，而不是根据业务需求将每个 AD 用户显式预调配到数据库集群：

- AD 用户是 Active Directory 中不同 AD 安全组的成员。这些不是由数据库集群管理员决定，而是基于业务要求，并由 AD 管理员处理。
- 数据库集群管理员根据业务需求在数据库实例中创建数据库角色。这些数据库角色可能具有不同的权限或特权。
- 数据库集群管理员根据每个数据库集群配置从 AD 安全组到数据库角色的映射。
- 数据库用户可以使用其 AD 凭证访问数据库集群。访问权限基于 AD 安全组成员资格。AD 用户根据其 AD 组成员资格自动获得或失去访问权限。

先决条件

在为 AD 安全组设置扩展之前，请确保您具备以下条件：

- 为 PostgreSQL 数据库集群设置 Kerberos 身份验证。有关更多信息，请参阅为 PostgreSQL 数据库集群设置 Kerberos 身份验证。

  > **Note**
  >
  > 对于 AD 安全组，请跳过此设置过程中的“步骤 7：为 Kerberos 主体创建 PostgreSQL 用户”。

- 在域中管理数据库集群。有关更多信息，请参阅在域中管理数据库集群。

设置 pg_ad_mapping 扩展

Aurora PostgreSQL 现在提供 `pg_ad_mapping` 扩展来管理 Aurora PostgreSQL 集群中 AD 安全组和数据库角色之间的映射。有关 `pg_ad_mapping` 提供的函数的更多信息，请参阅使用 `pg_ad_mapping` 扩展中的函数。

要在 Aurora PostgreSQL 数据库集群上设置 pg_ad_mapping 扩展，首先要将 pg_ad_mapping 添加到 Aurora PostgreSQL 数据库集群的自定义数据库集群参数组上的共享库中。有关创建自定义数据库集群参数组的信息，请参阅Amazon Aurora 的参数组。接下来，安装 pg_ad_mapping 扩展。本部分中的过程向您展示如何操作。您可以使用 AWS Management Console或 AWS CLI。

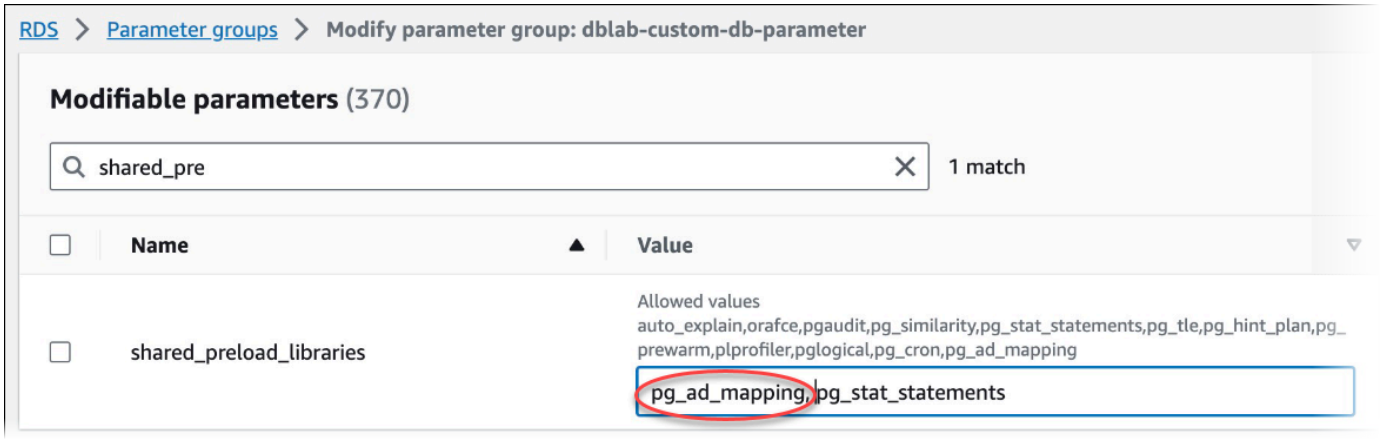
您必须拥有 rds_superuser 角色的权限才能执行所有这些任务。

以下步骤假设 Aurora PostgreSQL 数据库集群与自定义数据库集群参数组相关联。

控制台

设置 **pg_ad_mapping** 扩展

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Aurora PostgreSQL 数据库集群的写入器实例。
3. 打开 Aurora PostgreSQL 数据库集群写入器实例的配置选项卡。在实例详细信息中，找到 Parameter group（参数组）链接。
4. 选择此链接以打开与 Aurora PostgreSQL 数据库集群关联的自定义参数。
5. 在 Parameters（参数）搜索字段中，键入 shared_pre 以查找 shared_preload_libraries 参数。
6. 选择 Edit parameters（编辑参数）以访问属性值。
7. 将 pg_ad_mapping 添加到 Values（值）字段的列表中。使用逗号分隔值列表中的项目。



8. 重启 Aurora PostgreSQL 数据库集群的写入器实例，以使对 shared_preload_libraries 参数的更改生效。
9. 当实例可用时，验证 pg_ad_mapping 是否已初始化。使用 psql 连接到 Aurora PostgreSQL 数据库集群的写入器实例，然后运行以下命令。

```
SHOW shared_preload_libraries;
shared_preload_libraries
--------------------------
rdsutils,pg_ad_mapping
(1 row)
```

10. 初始化 pg_ad_mapping 后，您现在可以创建扩展了。您需要在初始化库之后创建扩展，才能开始使用此扩展提供的函数。

```
CREATE EXTENSION pg_ad_mapping;
```

11. 关闭 psql 会话。

```
labdb=> \q
```

AWS CLI

设置 pg_ad_mapping

要使用 AWS CLI 设置 pg_ad_mapping，您可以调用 modify-db-parameter-group 操作来在自定义参数组中添加此参数，如以下过程所示。

1. 使用以下 AWS CLI 命令向 shared_preload_libraries 参数中添加 pg_ad_mapping。

```
aws rds modify-db-parameter-group \
   --db-parameter-group-name custom-param-group-name \
   --parameters
 "ParameterName=shared_preload_libraries,ParameterValue=pg_ad_mapping,ApplyMethod=pending-
reboot" \
   --region aws-region
```

2. 使用以下 AWS CLI 命令重启 Aurora PostgreSQL 数据库集群的写入器实例，以便初始化 pg_ad_mapping。

```
aws rds reboot-db-instance \
    --db-instance-identifier writer-instance \
    --region aws-region
```

3. 当实例可用时，您可以验证 pg_ad_mapping 是否已初始化。使用 psql 连接到 Aurora PostgreSQL 数据库集群的写入器实例，然后运行以下命令。

```
SHOW shared_preload_libraries;
shared_preload_libraries
--------------------------
rdsutils,pg_ad_mapping
(1 row)
```

初始化 pg_ad_mapping 后，您现在可以创建扩展了。

```
CREATE EXTENSION pg_ad_mapping;
```

4. 关闭 psql 会话以便您可以使用 AWS CLI。

```
labdb=> \q
```

在 PowerShell 中检索 Active Directory 组 SID

安全标识符（SID）用于唯一标识安全主体或安全组。每当在 Active Directory 中创建安全组或账户时，都会为其分配一个 SID。要从 Active Directory 获取 AD 安全组 SID，可以从加入该 Active Directory 域的 Windows 客户端计算机中使用 Get-ADGroup cmdlet。Identity 参数指定用于获取相应 SID 的 Active Directory 组名称。

以下示例返回 AD 组 *adgroup1* 的 SID。

```
C:\Users\Admin> Get-ADGroup -Identity adgroup1 | select SID

             SID
-----------------------------------------------
S-1-5-21-3168537779-1985441202-1799118680-1612
```

映射数据库角色与 AD 安全组

您需要将数据库中的 AD 安全组显式预调配为 PostgreSQL 数据库角色。属于至少一个预调配 AD 安全组的 AD 用户将获得数据库的访问权限。您不应该向基于 AD 组安全性的数据库角色授予 rds_ad role。安全组的 Kerberos 身份验证将通过使用域名后缀（例如 *user1@example.com*）来触发。此数据库角色无法使用密码或 IAM 身份验证来访问数据库。

> **Note**
>
> 在数据库中具有相应数据库角色并获得 `rds_ad` 角色的 AD 用户不能作为 AD 安全组的一部分登录。他们将以单个用户身份通过数据库角色获得访问权限。

例如，accounts-group 是 AD 中的一个安全组，您希望在 Aurora PostgreSQL 中将该安全组预调配为 accounts-role。

AD 安全组	PosgreSQL 数据库角色
accounts-group	accounts-role

将数据库角色与 AD 安全组映射时，必须确保该数据库角色设置了 LOGIN 属性并且具有访问所需登录数据库的 CONNECT 权限。

```
postgres => alter role accounts-role login;

ALTER ROLE
postgres => grant connect on database accounts-db to accounts-role;
```

管理员现在可以继续创建 AD 安全组和 PostgreSQL 数据库角色之间的映射。

```
admin=>select pgadmap_set_mapping('accounts-group', 'accounts-role', <SID>, <Weight>);
```

有关检索 AD 安全组的 SID 的信息，请参阅在 PowerShell 中检索 Active Directory 组 SID。

在某些情况下，AD 用户可能属于多个组，在这种情况下，AD 用户将继承预调配了最高权重的数据库角色的权限。如果两个角色的权重相同，则 AD 用户将继承与最近添加的映射相对应的数据库角色的权限。建议指定权重，以反映各个数据库角色的相对权限/特权。数据库角色的权限或特权越高，应与映射条目关联的权重就越高。这将避免两个具有相同权重的映射产生歧义。

下表显示了从 AD 安全组到 Aurora PostgreSQL 数据库角色的映射示例。

AD 安全组	PosgreSQL 数据库角色	权重
accounts-group	accounts-role	7

AD 安全组	PosgreSQL 数据库角色	权重
sales-group	sales-role	10
dev-group	dev-role	7

在以下示例中，user1 将继承 sales-role 的权限，因为它的权重更高；而 user2 将继承 dev-role 的权限，因为该角色的映射是在 accounts-role 后创建的，它们的权重与 accounts-role 相同。

用户名	安全组成员资格
user1	accounts-group sales-group
user2	accounts-group dev-group

用于建立、列出和清除映射的 psql 命令如下所示。目前，无法修改单个映射条目。需要删除现有条目并重新创建映射。

```
admin=>select pgadmap_set_mapping('accounts-group', 'accounts-role', 'S-1-5-67-890',
 7);
admin=>select pgadmap_set_mapping('sales-group', 'sales-role', 'S-1-2-34-560', 10);
admin=>select pgadmap_set_mapping('dev-group', 'dev-role', 'S-1-8-43-612', 7);

admin=>select * from pgadmap_read_mapping();

ad_sid       | pg_role        | weight | ad_grp
-------------+----------------+--------+---------------
S-1-5-67-890 | accounts-role | 7      | accounts-group
S-1-2-34-560 | sales-role    | 10     | sales-group
S-1-8-43-612 | dev-role      | 7      | dev-group
(3 rows)
```

AD 用户身份日志记录/审计

使用以下命令确定当前用户或会话用户继承的数据库角色：

```
postgres=>select session_user, current_user;

session_user | current_user
-------------+--------------
dev-role     | dev-role

(1 row)
```

要确定 AD 安全主体身份，请使用以下命令：

```
postgres=>select principal from pg_stat_gssapi where pid = pg_backend_pid();

 principal
-------------------------
 user1@example.com

(1 row)
```

目前，AD 用户身份在审计日志中不可见。可以启用 log_connections 参数来记录数据库会话建立。有关更多信息，请参阅 log_connections。其输出包括 AD 用户身份，如下所示。然后，与此输出关联的后端 PID 有助于将操作归因回实际 AD 用户。

```
pgrole1@postgres:[615]:LOG: connection authorized: user=pgrole1
 database=postgres application_name=psql GSS (authenticated=yes, encrypted=yes,
 principal=Admin@EXAMPLE.COM)
```

限制

- 不支持称为 Azure Active Directory 的 Microsoft Entra ID。

使用 pg_ad_mapping 扩展中的函数

pg_ad_mapping 扩展为以下函数提供了支持：

pgadmap_set_mapping

此函数在 AD 安全组与具有关联权重的数据库角色之间建立映射。

语法

```
pgadmap_set_mapping(
ad_group,
db_role,
ad_group_sid,
weight)
```

参数

参数	描述
ad_group	AD 组的名称。值不能为 null 或空字符串。
db_role	要映射到指定 AD 组的数据库角色。值不能为 Null 或空字符串。
ad_group_sid	用于唯一标识 AD 组的安全标识符。值以“S-1-”开头，不能为 null 或空字符串。有关更多信息，请参阅 在 PowerShell 中检索 Active Directory 组 SID。
weight	与数据库角色关联的权重。当用户是多个组的成员时，权重最高的角色优先。权重的默认值为 1。

返回类型

None

使用说明

此函数添加了从 AD 安全组到数据库角色的新映射。它只能由具有 rds_superuser 权限的用户在数据库集群的主数据库实例上执行。

示例

```
postgres=> select pgadmap_set_mapping('accounts-group','accounts-
role','S-1-2-33-12345-67890-12345-678',10);

pgadmap_set_mapping

(1 row)
```

pgadmap_read_mapping

此函数列出使用 pgadmap_set_mapping 函数设置的 AD 安全组和数据库角色之间的映射。

语法

```
pgadmap_read_mapping()
```

参数

None

返回类型

参数	描述
ad_group_sid	用于唯一标识 AD 组的安全标识符。值以“S-1-”开头，不能为 Null 或空字符串。有关更多信息，请参阅在 PowerShell 中检索 Active Directory 组 SID。accounts-role@example.com
db_role	要映射到指定 AD 组的数据库角色。值不能为 Null 或空字符串。
weight	与数据库角色关联的权重。当用户是多个组的成员时，权重最高的角色优先。权重的默认值为 1。
ad_group	AD 组的名称。值不能为 Null 或空字符串。

使用说明

调用此函数以列出 AD 安全组和数据库角色之间的所有可用映射。

示例

```
postgres=> select * from pgadmap_read_mapping();

ad_sid                              | pg_role       | weight | ad_grp
------------------------------------+---------------+--------+------------------
S-1-2-33-12345-67890-12345-678      | accounts-role | 10     | accounts-group
(1 row)
```

```
(1 row)
```

pgadmap_reset_mapping

此函数重置使用 pgadmap_set_mapping 函数设置的一个或所有映射。

语法

```
pgadmap_reset_mapping(
ad_group_sid,
db_role,
weight)
```

参数

参数	描述
ad_group_sid	用于唯一标识 AD 组的安全标识符。
db_role	要映射到指定 AD 组的数据库角色。
weight	与数据库角色关联的权重。

如果未提供任何参数，则将重置 AD 组到数据库角色的所有映射。要么需要提供所有参数，要么不提供任何参数。

返回类型

None

使用说明

调用此函数以删除 AD 组到数据库角色的特定映射或重置所有映射。此函数只能由具有 rds_superuser 权限的用户在数据库集群的主数据库实例上执行。

示例

```
postgres=> select * from pgadmap_read_mapping();

   ad_sid                         | pg_role      | weight      | ad_grp
----------------------------------+--------------+-------------+--------------------
```

```
 S-1-2-33-12345-67890-12345-678 | accounts-role| 10          | accounts-group
 S-1-2-33-12345-67890-12345-666 | sales-role   | 10          | sales-group

(2 rows)
postgres=> select pgadmap_reset_mapping('S-1-2-33-12345-67890-12345-678', 'accounts-
role', 10);

pgadmap_reset_mapping
(1 row)

postgres=> select * from pgadmap_read_mapping();

   ad_sid                        | pg_role      | weight      | ad_grp
---------------------------------+--------------+-------------+---------------
 S-1-2-33-12345-67890-12345-666 | sales-role   | 10          | sales-group

(1 row)
postgres=> select pgadmap_reset_mapping();

pgadmap_reset_mapping
(1 row)

postgres=> select * from pgadmap_read_mapping();

   ad_sid                        | pg_role      | weight      | ad_grp
---------------------------------+--------------+-------------+--------------
 (0 rows)
```

将数据迁移到与 PostgreSQL 兼容的 Amazon Aurora

对于将数据从现有数据库迁移到 Amazon Aurora PostgreSQL 兼容版 数据库集群，您有多种选择。您的迁移选项还取决于您从中迁移数据的数据库和您迁移数据的规模。以下是您的选择：

使用快照迁移 RDS for PostgreSQL 数据库实例

您可以将数据直接从 RDS for PostgreSQL 数据库快照迁移到 Aurora PostgreSQL 数据库集群。

使用 Aurora 只读副本迁移 RDS for PostgreSQL 数据库实例

也可以通过创建 PostgreSQL 数据库实例的 Aurora PostgreSQL 只读副本从 RDS for PostgreSQL 数据库实例进行迁移。当 RDS for PostgreSQL 数据库实例和 Aurora PostgreSQL 只读副本之

间的副本滞后为 0 时，您可以停止复制。此时，您可以将 Aurora 只读副本作为独立的 Aurora PostgreSQL 数据库集群以进行读取和写入。

将 Amazon S3 中的数据导入到 Aurora PostgreSQL

您可以通过将数据从 Amazon S3 导入属于 Aurora PostgreSQL 数据库集群的表来迁移数据。

从与 PostgreSQL 不兼容的数据库迁移

您可以使用 AWS Database Migration Service (AWS DMS) 从非 PostgreSQL 兼容数据库中迁移数据。有关 AWS DMS 的更多信息，请参阅 AWS Database Migration Service 用户指南中的什么是 AWSDatabase Migration Service？。

> **Note**
>
> 在从 RDS for PostgreSQL 迁移期间，Aurora PostgreSQL 数据库集群目前不支持启用 Kerberos 身份验证。您只能在独立的 Aurora PostgreSQL 数据库集群上启用 Kerberos 身份验证。

有关 Aurora 可用的 AWS 区域的列表，请参阅《AWS 一般参考》中的 Amazon Aurora。

> **Important**
>
> 如果您计划在不久的将来将 RDS for PostgreSQL 数据库实例迁移到 Aurora PostgreSQL 数据库集群，我们强烈建议您在迁移规划阶段的早期禁用数据库实例的自动次要版本升级。如果 Aurora PostgreSQL 尚不支持 RDS for PostgreSQL 版本，迁移到 Aurora PostgreSQL 可能会延迟。
>
> 有关 Aurora PostgreSQL 版本的信息，请参阅 Amazon Aurora PostgreSQL 的引擎版本。

将 RDS for PostgreSQL 数据库的快照迁移到 Aurora PostgreSQL 数据库集群

要创建 Aurora PostgreSQL 数据库集群，您可以迁移 RDS for PostgreSQL 数据库实例的数据库快照。将使用原始 RDS for PostgreSQL 数据库实例中的数据填充新的 Aurora PostgreSQL 数据库集群。有关创建数据库快照的信息，请参阅创建数据库快照。

在某些情况下，数据库快照可能不在要查找数据的 AWS 区域 中。如果是这样的话，请使用 Amazon RDS 控制台将数据库快照复制到该 AWS 区域。有关复制数据库快照的信息，请参阅复制数据库快照。

您可以迁移与给定 AWS 区域 中可用 Aurora PostgreSQL 版本兼容的 RDS for PostgreSQL 快照。例如，您可以将 RDS for PostgreSQL 11.1 数据库实例中的快照迁移到美国西部（加利福尼亚北部）区域中的 Aurora PostgreSQL 版本 11.4、11.7、11.8 或 11.9。您可以将 RDS for PostgreSQL 10.11 快照迁移到 Aurora PostgreSQL 10.11、10.12、10.13 和 10.14。换言之，RDS for PostgreSQL 快照必须使用与 Aurora PostgreSQL 相同的版本或较低的次要版本。

您还可以选择使用 AWS KMS key 静态加密新的 Aurora PostgreSQL 数据库集群。该选项仅适用于未加密的数据库快照。

要将 RDS for PostgreSQL 数据库快照迁移到 Aurora PostgreSQL 数据库集群，您可以使用 AWS Management Console、AWS CLI 或 RDS API。使用 AWS Management Console 时，控制台将执行必要的操作创建数据库集群和主实例。

控制台

使用 RDS 控制台迁移 PostgreSQL 数据库快照

1. 登录AWS Management Console并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 选择 Snapshots。
3. 在 Snapshots (快照) 页面上，选择要迁移到 Aurora PostgreSQL 数据库集群的 RDS for PostgreSQL 快照。
4. 选择 Actions（操作），然后选择 Migrate snapshot（迁移快照）。
5. 在 Migrate Database（迁移数据库）页面上设置以下值：

   - DB engine version（数据库引擎版本）：选择要用于新迁移实例的数据库引擎版本。
   - DB instance identifier（数据库实例标识符）：为数据库集群输入一个名称，该名称在您选择的 AWS 区域 中对于您的账户是唯一的。此标识符将在数据库集群中实例的终端节点地址中使用。您可以选择对该名称进行一些巧妙处理，例如，包括选定的 AWS 区域 和数据库引擎（例如 **`aurora-cluster1`**）。

     数据库实例标识符具有以下限制：

     - 它必须包含 1–63 个字母数字字符或连字符。
     - 它的第一个字符必须是字母。

  - 它不能以连字符结尾，也不能包含两个连续连字符。
  - 它对于每个 AWS 区域 的每个 AWS 账户的所有数据库实例必须是唯一的。
- DB instance class (数据库实例类)：选择具有数据库所需的存储和容量的数据库实例类，例如 `db.r6g.large`。Aurora 集群卷随着数据库中数据量的增加自动增大。因此，您只需选择满足当前存储要求的数据库实例类。有关更多信息，请参阅 Amazon Aurora 存储概述。
- Virtual Private Cloud (VPC)：如果已具有 VPC，您可以选择 VPC 标识符（如 `vpc-a464d1c1`）以将该 VPC 用于 Aurora PostgreSQL 数据库集群。有关如何创建 VPC 的信息，请参阅教程：创建 VPC 以用于数据库集群（仅限 IPv4）。

  否则，可以通过选择 Create a new VPC（新建 VPC），让 Amazon RDS 为您创建 VPC。
- DB subnet group（数据库子网组）：如果已具有子网组，则可通过选择您的子网组标识符（例如 `gs-subnet-group1`）来将该子网组用于 Aurora PostgreSQL 数据库集群。
- Public access（公开访问）：选择 No（否）可指定数据库集群中的实例只能由 VPC 内的资源访问。选择 Yes (是) 可指定数据库集群中的实例可以由公用网络上的资源访问。

> **Note**
>
> 您的生产数据库集群可能不需要位于公有子网中，因为仅应用程序服务器将需要访问数据库集群。如果数据库集群不需要位于公有子网中，请将 Public access（公开访问）设置为 No（否）。

- VPC security group（VPC 安全组）：选择 VPC 安全组以允许访问数据库。
- Availability Zone (可用区)：选择为 Aurora PostgreSQL 数据库集群托管主实例的可用区。要让 Amazon RDS 为您选择可用区，请选择 No Preference（无首选项）。
- Database port（数据库端口）：输入在连接到 Aurora PostgreSQL 数据库集群中的实例时使用的默认端口。默认为 `5432`。

> **Note**
>
> 您可能位于企业防火墙后面，该防火墙不允许访问默认端口 (例如，PostgreSQL 默认端口 5432)。在此情况下，请提供企业防火墙允许的端口值。请记住此端口值，以便在稍后连接到 Aurora PostgreSQL 数据库集群时使用。

- Enable Encryption（启用加密）：为要静态加密的新 Aurora PostgreSQL 数据库集群选择 Enable Encryption（启用加密）。另外，选择 KMS 密钥作为 AWS KMS key 值。

- Auto minor version upgrade (自动次要版本升级)：选择 Enable auto minor version upgrade (启用自动次要版本升级) 可让您的 Aurora PostgreSQL 数据库集群在次要 PostgreSQL 数据库引擎版本升级可用时自动接收这些升级。

  Auto Minor Version Upgrade（自动次要版本升级）选项仅适用于升级到 Aurora PostgreSQL 数据库集群的 PostgreSQL 次要引擎版本。它不适用于应用于维持系统稳定性的常规修补程序。

6. 选择 Migrate 以迁移您的数据库快照。
7. 选择 Databases（数据库）以查看新的数据库集群。选择新的数据库集群以监控迁移进度。迁移完成后，集群的状态为 Available（可用）。在 Connectivity & security（连接和安全性）选项卡上，您可以找到用于连接数据库集群主写入器实例的集群终端节点。有关连接到 Aurora PostgreSQL 数据库集群的更多信息，请参阅连接到 Amazon Aurora 数据库集群。

AWS CLI

使用 AWS CLI 将 RDS for PostgreSQL 数据库快照迁移到 Aurora PostgreSQL 涉及两个单独的 AWS CLI 命令。首先，请使用 `restore-db-cluster-from-snapshot` AWS CLI 命令创建新的 Aurora PostgreSQL 数据库集群。然后，使用 `create-db-instance` 命令在新集群中创建主数据库实例来完成迁移。以下过程创建 Aurora PostgreSQL 数据库集群，其主数据库实例配置与用于创建快照的数据库实例配置相同。

将 RDS for PostgreSQL 数据库快照迁移到 Aurora PostgreSQL 数据库集群

1. 使用 describe-db-snapshots 命令来获取要迁移的数据库快照的信息。您可以在命令中指定 `--db-instance-identifier` 参数或 `--db-snapshot-identifier`。如果未指定参数，则将获得所有快照。

   ```
   aws rds describe-db-snapshots --db-instance-identifier <your-db-instance-name>
   ```

2. 该命令返回从指定数据库实例创建的快照的所有配置详细信息。在输出中，找到要迁移的快照及其 Amazon Resource Name (ARN)。要了解有关 Amazon RDS ARN 的更多信息，请参阅 Amazon Relational Database Service (Amazon RDS)。ARN 看上去类似于以下输出。

   ```
   “DBSnapshotArn": "arn:aws:rds:aws-region:111122223333:snapshot:<snapshot_name>"
   ```

   此外，您还可以在输出中找到 RDS for PostgreSQL 数据库实例的配置详细信息，例如引擎版本、分配的存储空间、数据库实例是否加密，等等。

3. 使用 restore-db-cluster-from-snapshot 命令启动迁移。指定以下参数：

- `--db-cluster-identifier`：要为 Aurora PostgreSQL 数据库集群指定的名称。此 Aurora 数据库集群是数据库快照迁移的目标。
- `--snapshot-identifier`：待迁移数据库快照的 Amazon Resource Name (ARN)。
- `--engine`：为 Aurora 数据库集群引擎指定 `aurora-postgresql`。
- `--kms-key-id`：此可选参数允许您从未加密的数据库快照创建加密的 Aurora PostgreSQL 数据库集群。此参数还允许您为数据库集群选择与数据库快照所用密钥不同的加密密钥。

> **Note**
>
> 您无法从加密的数据库快照创建未加密的 Aurora PostgreSQL 数据库集群。

如果没有如下所示指定的 `--kms-key-id` 参数，restore-db-cluster-from-snapshot AWS CLI 命令将创建一个空的 Aurora PostgreSQL 数据库集群，该集群将使用与数据库快照相同的密钥加密，如果源数据库快照未加密，则该集群将不会加密。

对于 Linux、macOS 或 Unix：

```
aws rds restore-db-cluster-from-snapshot \
    --db-cluster-identifier cluster-name \
    --snapshot-identifier arn:aws:rds:aws-region:111122223333:snapshot:your-
snapshot-name \
    --engine aurora-postgresql
```

对于 Windows：

```
aws rds restore-db-cluster-from-snapshot ^
    --db-cluster-identifier new_cluster ^
    --snapshot-identifier arn:aws:rds:aws-region:111122223333:snapshot:your-
snapshot-name ^
    --engine aurora-postgresql
```

4. 该命令将返回为迁移创建的 Aurora PostgreSQL 数据库集群的详细信息。您可以使用 describe-db-clusters AWS CLI 命令查看 Aurora PostgreSQL 数据库集群的状态。

```
aws rds describe-db-clusters --db-cluster-identifier cluster-name
```

5. 当数据库集群变为“可用”时，您可以使用 create-db-instance 命令，使用基于 Amazon RDS 数据库快照的数据库实例填充 Aurora PostgreSQL 数据库集群。指定以下参数：

   - `--db-cluster-identifier`：在前述步骤中创建的新 Aurora PostgreSQL 数据库集群的名称。
   - `--db-instance-identifier`：要为该数据库实例指定的名称。此实例将成为 Aurora PostgreSQL 数据库集群中的主节点。
   - `----db-instance-class` ：指定要使用的数据库实例类。从要迁移到的 Aurora PostgreSQL 版本支持的数据库实例类中进行选择。有关更多信息，请参阅 数据库实例类类型 和 数据库实例类支持的数据库引擎。
   - `--engine`：为数据库实例指定 `aurora-postgresql`。

   您还可以使用与源数据库快照不同的配置来创建数据库实例，方法是在 `create-db-instance` AWS CLI 命令中传入适当选项。有关更多信息，请参阅 create-db-instance 命令。

   对于 Linux、macOS 或 Unix：

```
aws rds create-db-instance \
    --db-cluster-identifier cluster-name \
    --db-instance-identifier --db-instance-class db.instance.class \
    --engine aurora-postgresql
```

   对于 Windows：

```
aws rds create-db-instance ^
    --db-cluster-identifier cluster-name ^
    --db-instance-identifier --db-instance-class db.instance.class ^
    --engine aurora-postgresql
```

迁移过程完成后，Aurora PostgreSQL 集群拥有已填充的主数据库实例。

使用 Aurora 只读副本将数据从 RDS for PostgreSQL 数据库实例迁移到 Aurora PostgreSQL 数据库集群

通过使用 Aurora 只读副本进行迁移过程，您可以使用 RDS for PostgreSQL 数据库实例作为新 Aurora PostgreSQL 数据库集群的基础。Aurora 只读副本选项仅适用于在同一 AWS 区域和账户内迁移，并且仅当该区域为 RDS for PostgreSQL 数据库实例提供兼容版本的 Aurora PostgreSQL 时才可用。兼容意味着 Aurora PostgreSQL 版本与 RDS for PostgreSQL 版本相同，或者它是同一主要版本系列中更高的次要版本。

例如，要使用此方法迁移 RDS for PostgreSQL 11.14 数据库实例，该区域必须提供 PostgreSQL 版本 11 系列中的 Aurora PostgreSQL 11.14 版或更高的次要版本。

主题

- 使用 Aurora 只读副本迁移数据概览
- 准备使用 Aurora 只读副本迁移数据
- 创建 Aurora 只读副本
- 提升 Aurora 只读副本

使用 Aurora 只读副本迁移数据概览

从 RDS for PostgreSQL 数据库实例迁移到 Aurora PostgreSQL 数据库集群是一个多步骤过程。首先，您可创建源 RDS for PostgreSQL 数据库实例的 Aurora 只读副本。这将启动从 RDS for PostgreSQL 数据库实例到特殊用途数据库集群（称为副本集群）的复制过程。副本集群仅由 Aurora 只读副本（读取器实例）组成。

> **Note**
>
> 每 TB 数据可能需要几小时才能完成迁移。

提升 Aurora PostgreSQL 副本

创建了 Aurora PostgreSQL 数据库集群后，请按照以下步骤提升 Aurora 副本：

1. 停止源 RDS for PostgreSQL 数据库实例上的所有数据库写入工作负载。
2. 从源 RDS for PostgreSQL 数据库实例获取当前 `WAL LSN`：

```
SELECT pg_current_wal_lsn();
pg_current_wal_lsn
--------------------
0/F0000318
(1 row)
```

3. 在 Aurora PostgreSQL 副本集群上，检查重播的 LSN 是否大于步骤 2 中的 LSN：

```
SELECT pg_last_wal_replay_lsn();
pg_last_wal_replay_lsn
------------------------
0/F0000400
(1 row)
```

或者，对源 RDS for PostgreSQL 数据库实例使用以下查询：

```
SELECT restart_lsn FROM pg_replication_slots;
```

4. 提升 Aurora PostgreSQL 副本集群。

当复制停止时，副本集群即被提升为独立 Aurora PostgreSQL 数据库集群，而读取器被提升为集群的写入器实例。此时，您可以将实例添加到 Aurora PostgreSQL 数据库集群，以根据自己的用例调整规模。如果您不再需要原始的 RDS for PostgreSQL 数据库实例，可以删除它。

如果 RDS for PostgreSQL 数据库实例已有 Aurora 只读副本或已有跨区域只读副本，则无法创建 Aurora 只读副本。

准备使用 Aurora 只读副本迁移数据

在使用 Aurora 只读副本的迁移过程中，对源 RDS for PostgreSQL 数据库实例进行的更新将异步复制到副本集群的 Aurora 只读副本。该过程使用 PostgreSQL 的本机流式复制功能，该功能将预写日志 (WAL) 段存储在源实例上。在开始此迁移过程之前，请检查表中列出的指标的值，确保您的实例具有足够的存储容量。

指标	描述
FreeStorageSpace	可用存储空间。 单位：字节
OldestReplicationSlotLag	最滞后的副本中的 WAL 数据的滞后大小。 单位：MB
RDSToAuroraPostgreSQLReplicaLag	Aurora PostgreSQL 数据库集群滞后于源 RDS 数据库实例的时间长度（以秒为单位）。
TransactionLogsDiskUsage	事务日志使用的磁盘空间。 单位：MB

有关监控 RDS 实例的更多信息，请参阅 Amazon RDS 用户指南 中的监控。

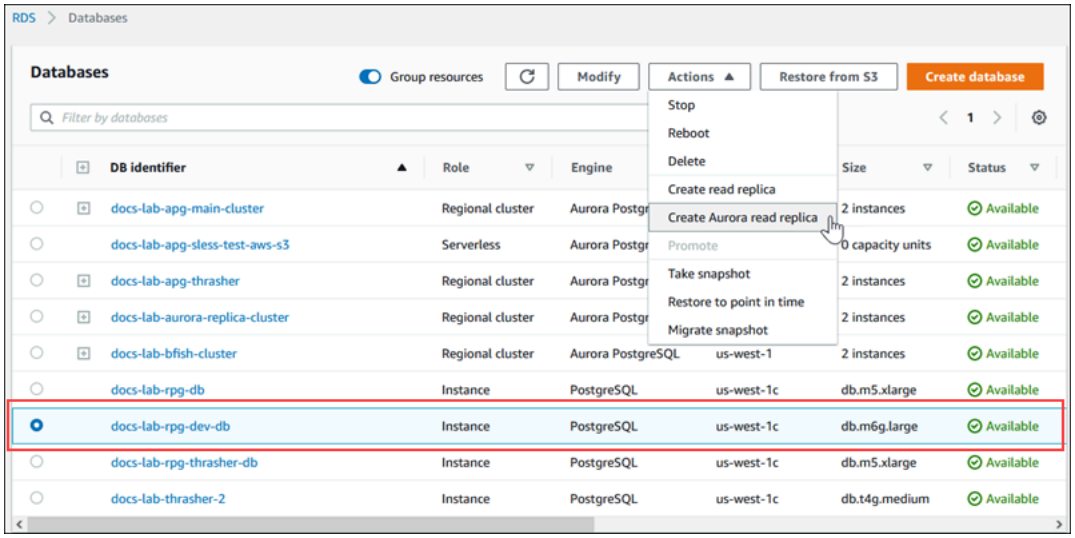
创建 Aurora 只读副本

您可以使用 AWS Management Console 或 AWS CLI 为 RDS for PostgreSQL 数据库实例创建 Aurora 只读副本。仅当 AWS 区域提供兼容的 Aurora PostgreSQL 版本时，使用 AWS Management Console 创建 Aurora 只读副本的选项才可用。也就是说，仅当 Aurora PostgreSQL 版本与 RDS for PostgreSQL 版本相同或者是同一主要版本系列中更高的次要版本时可用。

控制台

从源 PostgreSQL 数据库实例创建 Aurora 只读副本

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Databases (数据库)。
3. 选择要作为 Aurora 只读副本源的 RDS for PostgreSQL 数据库实例。对于操作，请选择创建 Aurora 只读副本。如果未显示此选项，则意味着该区域中没有兼容的 Aurora PostgreSQL 版本。



4. 在创建 Aurora 只读副本设置页面上，您可以配置 Aurora PostgreSQL 数据库集群的属性，如下表中所示。副本数据库集群是使用与源相同的“主”用户名和密码从源数据库实例的快照创建的，因此您目前无法更改这些属性。

选项	描述
数据库实例类	选择满足数据库集群中主实例的处理和内存要求的数据库实例类。有关更多信息，请参阅Amazon Aurora 数据库实例类。
多可用区部署	迁移期间不可用
数据库实例标识符	输入要为该数据库实例指定的名称。此标识符在新数据库集群主实例的终端节点地址中使用。 数据库实例标识符具有以下限制： • 它必须包含 1–63 个字母数字字符或连字符。 • 它的第一个字符必须是字母。 • 它不能以连字符结尾，也不能包含两个连续连字符。 • 它对于每个 AWS 区域 每个 AWS 账户的所有数据库实例必须是唯一的。

选项	描述
Virtual Private Cloud (VPC)	选择要托管数据库集群的 VPC。选择 Create a new VPC (新建 VPC) 以让 Amazon RDS 为您创建 VPC。有关更多信息，请参阅数据库集群先决条件。
DB subnet group（数据库子网组）	选择要用于数据库集群的数据库子网组。选择 Create new DB Subnet Group (创建新的数据库子网组) 以让 Amazon RDS 为您创建数据库子网组。有关更多信息，请参阅“数据库集群先决条件”。
公开可用性	选择 Yes (是) 向数据库集群提供公有 IP 地址；否则，请选择 No (否)。数据库集群中的实例可以混合使用公有和私有数据库实例。有关隐藏实例以防止公开访问的更多信息，请参阅对互联网隐藏 VPC 中的数据库集群。
可用区	确定您是否希望指定特定的可用区。有关可用区的更多信息，请参阅区域及可用区。
VPC 安全组	选择一个或多个 VPC 安全组以保护对数据库集群的网络访问。选择 Create new VPC security group (新建 VPC 安全组) 以让 Amazon RDS 为您创建 VPC 安全组。有关更多信息，请参阅“数据库集群先决条件”。
数据库端口	指定应用程序和实用程序用来访问数据库的端口。Aurora PostgreSQL 数据库集群默认为使用默认 PostgreSQL 端口 5432。有些公司的防火墙不允许连接到此端口。如果您的公司防火墙阻止使用默认端口，请为新数据库集群选择其他端口。
数据库参数组	为 Aurora PostgreSQL 数据库集群选择数据库参数组。Aurora 具有一个可使用的默认数据库参数组，您也可以创建自己的数据库参数组。有关数据库参数组的更多信息，请参阅 Amazon Aurora 的参数组。

选项	描述
数据库集群参数组	为 Aurora PostgreSQL 数据库集群选择数据库集群参数组。Aurora 具有一个可使用的默认数据库集群参数组，您也可以创建自己的数据库集群参数组。有关数据库集群参数组的更多信息，请参阅 Amazon Aurora 的参数组。
加密	为要静态加密的新 Aurora 数据库集群选择启用加密。如果选择启用加密，请也选择一个 KMS 密钥作为 AWS KMS key 值。
优先级	选择数据库集群的失效转移优先级。如果您未选择值，则默认值为 tier-1。此优先级决定从主实例故障恢复时提升 Aurora 副本的顺序。有关更多信息，请参阅“Aurora 数据库集群的容错能力”。
备份保留期	为 Aurora 选择将保留数据库的备份副本的时间长度（1–35 天）。可使用备份副本对数据库执行时间点还原 (PITR)，以还原到第二个时间点。
增强监控	选择启用增强监控可启用您的数据库集群在其上运行的操作系统的实时指标收集。有关更多信息，请参阅“使用增强监控来监控操作系统指标”。
监控角色	仅在选择 Enable enhanced monitoring (启用增强监控) 时可用。要用于增强监控的 AWS Identity and Access Management（IAM）角色。有关更多信息，请参阅“设置和启用增强监控”。
粒度	仅在选择启用增强监控时可用。设置为数据库集群收集指标的时间间隔（以秒为单位）。

选项	描述
自动次要版本升级	选择 Yes (是) 以使 Aurora PostgreSQL 数据库集群能够在次要 PostgreSQL 数据库引擎版本升级可用时自动接收这些升级。 Auto Minor Version Upgrade（自动次要版本升级）选项仅适用于升级到 Aurora PostgreSQL 数据库集群的 PostgreSQL 次要引擎版本。它不适用于应用于维持系统稳定性的常规修补程序。
维护窗口	选择可以进行系统维护的每周时间范围。

5. 选择 Create read replica (创建只读副本)。

AWS CLI

要使用 AWS CLI 从源 RDS for PostgreSQL 数据库实例创建 Aurora 只读副本，请首先使用 create-db-cluster CLI 命令创建一个空 Aurora 数据库集群。一旦存在数据库集群，您就可以使用 create-db-instance 命令为数据库集群创建主实例。主实例是在 Aurora 数据库集群中创建的第一个实例。在这种情况下，它最初作为 RDS for PostgreSQL 数据库实例的 Aurora 只读副本而创建。当该过程结束时，您的 RDS for PostgreSQL 数据库实例已有效迁移到 Aurora PostgreSQL 数据库集群。

您不需要指定主用户账户（通常为 `postgres`）、其密码或数据库名称。Aurora 只读副本会自动从您在调用 AWS CLI 命令时标识的源 RDS for PostgreSQL 数据库实例获取这些信息。

您需要指定要用于 Aurora PostgreSQL 数据库集群和数据库实例的引擎版本。您指定的版本应与源 RDS for PostgreSQL 数据库实例匹配。如果源 RDS for PostgreSQL 数据库实例已加密，您还需要为 Aurora PostgreSQL 数据库集群主实例指定加密。不支持将已加密的实例迁移到未加密的 Aurora 数据库集群。

以下示例创建一个名为 `my-new-aurora-cluster` 的 Aurora PostgreSQL 数据库集群，该集群将使用未加密的 RDS 数据库源实例。首先，您通过调用 `create-db-cluster` CLI 命令创建 Aurora PostgreSQL 数据库集群。该示例演示了如何使用可选 `--storage-encrypted` 参数指定应加密数据库集群。由于源数据库没有加密，因此 `--kms-key-id` 用于指定要使用的密钥。有关必需参数和可选参数的更多信息，请参阅示例后面的列表。

对于 Linux、macOS 或 Unix：

```
aws rds create-db-cluster  \
```

```
    --db-cluster-identifier my-new-aurora-cluster \
    --db-subnet-group-name my-db-subnet
    --vpc-security-group-ids sg-11111111
    --engine aurora-postgresql \
    --engine-version same-as-your-rds-instance-version \
    --replication-source-identifier arn:aws:rds:aws-region:111122223333:db/rpg-source-
db \
    --storage-encrypted \
    --kms-key-id arn:aws:kms:aws-
region:111122223333:key/11111111-2222-3333-444444444444
```

对于 Windows：

```
aws rds create-db-cluster  ^
    --db-cluster-identifier my-new-aurora-cluster ^
    --db-subnet-group-name my-db-subnet ^
    --vpc-security-group-ids sg-11111111 ^
    --engine aurora-postgresql ^
    --engine-version same-as-your-rds-instance-version ^
    --replication-source-identifier arn:aws:rds:aws-region:111122223333:db/rpg-source-
db ^
    --storage-encrypted ^
    --kms-key-id arn:aws:kms:aws-
region:111122223333:key/11111111-2222-3333-444444444444
```

在下面的列表中，您可以找到有关示例中所示某些选项的更多信息。除非另有说明，否则这些参数是必需的。

- `--db-cluster-identifier` – 您需要为新的 Aurora PostgreSQL 数据库集群命名。
- `--db-subnet-group-name` – 在与源数据库实例相同的数据库子网中创建 Aurora PostgreSQL 数据库集群。
- `--vpc-security-group-ids` – 指定 Aurora PostgreSQL 数据库集群的安全组。
- `--engine-version` – 指定要用于 Aurora PostgreSQL 数据库集群的版本。此次要版本应与源 RDS for PostgreSQL 数据库实例使用的版本相同，或比后者更高。
- `--replication-source-identifier` – 使用 RDS for PostgreSQL 数据库实例的 Amazon Resource Name (ARN) 识别该实例。有关 Amazon RDS ARN 的更多信息，请参阅数据库集群的《AWS 一般参考》中的 Amazon Relational Database Service（Amazon RDS）。
- `--storage-encrypted`：可选。仅在需要指定加密时使用，如下所示：

  - 当源数据库实例具有加密存储时，请使用此参数。如果您不对具有加密存储的源数据库实例使用此参数，则对 create-db-cluster 的调用会失败。如果您想为 Aurora PostgreSQL 数据库集群使用的密钥与源数据库实例使用的密钥不同，则还需要指定 --kms-key-id。
  - 如果源数据库实例的存储未加密，但您希望 Aurora PostgreSQL 数据库集群使用加密，则使用此参数。如果是这样，您还需要识别与 --kms-key-id 参数一起使用的加密密钥。
- --kms-key-id：可选。使用时，您可以通过使用密钥的 ARN、ID、别名 ARN 或其别名来指定用于存储加密 (--storage-encrypted) 的密钥。仅在以下情况下需要此参数：
  - 为 Aurora PostgreSQL 数据库集群选择一个密钥，该密钥不同于源数据库实例使用的密钥。
  - 从未加密的源创建加密集群。在这种情况下，您需要指定 Aurora PostgreSQL 应该用于加密的密钥。

创建 Aurora PostgreSQL 数据库集群后，您可以使用 create-db-instance CLI 命令创建主实例，如以下所示：

对于 Linux、macOS 或 Unix：

```
aws rds create-db-instance \
    --db-cluster-identifier my-new-aurora-cluster \
    --db-instance-class db.x2g.16xlarge \
    --db-instance-identifier rpg-for-migration \
    --engine aurora-postgresql
```

对于 Windows：

```
aws rds create-db-instance ^
    --db-cluster-identifier my-new-aurora-cluster ^
    --db-instance-class db.x2g.16xlarge ^
    --db-instance-identifier rpg-for-migration ^
    --engine aurora-postgresql
```

在下面的列表中，您可以找到有关示例中所示某些选项的更多信息。

- --db-cluster-identifier – 指定在前述步骤中使用 create-db-instance 命令创建的 Aurora PostgreSQL 数据库集群的名称。
- --db-instance-class – 要用于主实例的数据库实例类的名称，例如 db.r4.xlarge、db.t4g.medium、db.x2g.16xlarge 等。有关可用的数据库实例类列表，请参阅数据库实例类类型。

- `--db-instance-identifier` – 指定主实例的名称。
- `--engine aurora-postgresql` – 指定引擎的 `aurora-postgresql`。

RDS API

要从源 RDS for PostgreSQL 数据库实例创建 Aurora 只读副本，请首先使用 RDS API 操作 `CreateDBCluster` 为从源 RDS for PostgreSQL 数据库实例创建的 Aurora 只读副本创建新的 Aurora 数据库集群。当 Aurora PostgreSQL 数据库集群可用时，您可以使用 `CreateDBInstance` 创建 Aurora 数据库集群的主实例。

您不需要指定主用户账户（通常为 `postgres`）、其密码或数据库名称。Aurora 只读副本会自动从使用 `ReplicationSourceIdentifier` 指定的源 RDS for PostgreSQL 数据库实例中获取这些信息。

您需要指定要用于 Aurora PostgreSQL 数据库集群和数据库实例的引擎版本。您指定的版本应与源 RDS for PostgreSQL 数据库实例匹配。如果源 RDS for PostgreSQL 数据库实例已加密，您还需要为 Aurora PostgreSQL 数据库集群主实例指定加密。不支持将已加密的实例迁移到未加密的 Aurora 数据库集群。

要为 Aurora 只读副本创建 Aurora 数据库集群，请使用带以下参数的 RDS API 操作 `CreateDBCluster`：

- `DBClusterIdentifier` – 要创建的数据库集群的名称。
- `DBSubnetGroupName` – 要与该数据库集群关联的数据库子网组的名称。
- `Engine=aurora-postgresql` – 要使用的引擎的名称。
- `ReplicationSourceIdentifier` – 源 PostgreSQL 数据库实例的 Amazon Resource Name (ARN)。有关 Amazon RDS ARN 的更多信息，请参阅《Amazon Web Services 一般参考》中的 Amazon Relational Database Service（Amazon RDS）。如果 `ReplicationSourceIdentifier` 标识加密的源，Amazon RDS 将使用默认 KMS 密钥，除非您使用 `KmsKeyId` 选项指定一个不同的密钥。
- `VpcSecurityGroupIds` – 要与此数据库集群关联的 Amazon EC2 VPC 安全组的列表。
- `StorageEncrypted` – 指示数据库集群已加密。当您使用此参数而不同时指定 `ReplicationSourceIdentifier` 时，Amazon RDS 会使用您的默认 KMS 密钥。
- `KmsKeyId` – 已加密集群的密钥。使用时，您可以通过使用密钥的 ARN、ID、别名 ARN 或其别名来指定用于存储加密的密钥。

有关更多信息，请参阅 Amazon RDS API 参考中的 `CreateDBCluster`。

一旦 Aurora 数据库集群可用，您可以使用带以下参数的 RDS API 操作 `CreateDBInstance` 为其创建主实例：

- `DBClusterIdentifier` – 数据库集群的名称。
- `DBInstanceClass` – 要用于主实例的数据库实例类的名称。
- `DBInstanceIdentifier` – 主实例的名称。
- `Engine=aurora-postgresql` – 要使用的引擎的名称。

有关更多信息，请参阅 Amazon RDS API 参考中的 `CreateDBInstance`。

提升 Aurora 只读副本

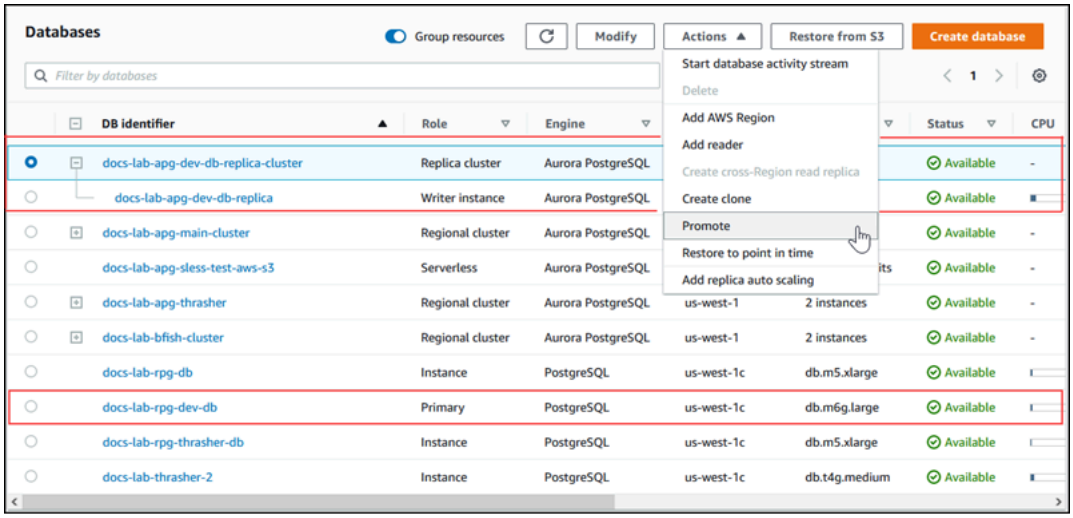
在提升副本集群之前，迁移到 Aurora PostgreSQL 未完成，因此暂时不要删除 RDS for PostgreSQL 源数据库实例。

在提升副本集群之前，请确保 RDS for PostgreSQL 数据库实例没有向数据库写入任何进程中的事务或其他活动。当 Aurora 只读副本上的副本滞后达到零 (0) 时，您可以提升副本集群。有关监控副本滞后的更多信息，请参阅 监控 Aurora PostgreSQL 复制 和 Amazon Aurora 的实例级指标。

控制台

将 Aurora 只读副本提升为 Aurora 数据库集群

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Databases (数据库)。
3. 选择副本集群。



4. 对于操作，请选择提升。这可能需要几分钟时间，并可能导致停机。

该过程完成后，Aurora 副本集群成为区域性 Aurora PostgreSQL 数据库集群，其写入器实例包含来自 RDS for PostgreSQL 数据库实例的数据。

AWS CLI

要将 Aurora 只读副本提升为独立数据库集群，请使用 promote-read-replica-db-cluster AWS CLI 命令。

Example

对于 Linux、macOS 或 Unix：

```
aws rds promote-read-replica-db-cluster \
    --db-cluster-identifier myreadreplicacluster
```

对于 Windows：

```
aws rds promote-read-replica-db-cluster ^
    --db-cluster-identifier myreadreplicacluster
```

RDS API

要将 Aurora 只读副本提升为独立数据库集群，请使用 RDS API 操作 PromoteReadReplicaDBCluster。

提升副本集群后，您可以通过检查事件日志来确认提升已完成，如下所示。

确认 Aurora 副本集群已提升

1. 登录AWS Management Console并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Events (事件)。
3. 在 Events（事件）页面的 Source（源）列表中找到集群的名称。每个事件都有来源、类型、时间和消息。您可以查看在您账户的 AWS 区域 中发生的所有事件。成功提升将生成以下消息。

   ```
   Promoted Read Replica cluster to a stand-alone database cluster.
   ```

提升完成后，源 RDS for PostgreSQL 数据库实例和 Aurora 数据库集群之间的链接会取消。您可以将客户端应用程序引导到 Aurora 只读副本的端点。有关 Aurora 终端节点的更多信息，请参阅Amazon Aurora 端点连接。此时，您可以安全地删除数据库实例。

优化 Aurora PostgreSQL 中的查询性能

优化查询性能至关重要，因为这有助于数据库更快、更高效地运行，同时使用更少的资源，从而改善用户体验并降低运营成本。Amazon Aurora PostgreSQL 提供了多项功能来协助优化 PostgreSQL 工作负载的查询性能。

主题

- 使用 Aurora 优化型读取功能提高 Aurora PostgreSQL 的查询性能
- 优化 Aurora PostgreSQL 中的关联子查询
- 使用自适应联接提高查询性能

使用 Aurora 优化型读取功能提高 Aurora PostgreSQL 的查询性能

使用 Aurora 优化型读取功能，您可以实现更快的 Aurora PostgreSQL 查询处理。使用 Aurora 优化型读取功能的 Aurora PostgreSQL 数据库实例可将查询延迟缩短多达 8 倍，对于具有超出数据库实例内存容量的大型数据集的应用程序，最多可节省 30% 的成本。

主题

- PostgreSQL 中的 Aurora 优化型读取功能概述
- 使用 Aurora 优化型读取
- Aurora 优化型读取的使用案例
- 监控使用 Aurora 优化读取的数据库实例
- Aurora 优化型读取的最佳实践

PostgreSQL 中的 Aurora 优化型读取功能概述

创建使用基于 Graviton 的 R6gd 和基于英特尔的 R6id 实例以及非易失性存储规范（NVMe）存储的数据库集群时，默认可以使用 Aurora 优化型读取功能。可在以下 PostgreSQL 版本中可使用此功能：

- 16.1 及所有更高版本
- 15.4 及更高版本

- 14.9 及更高版本

Aurora 优化型读取支持两种功能：分层缓存和临时对象。

启用优化型读取的分层缓存 - 使用分层缓存，您可以将数据库实例缓存容量最多扩展至实例内存的 5 倍。这样可以自动维护缓存以包含最新的、事务一致的数据，从而使应用程序免除管理基于外部结果集的缓存解决方案数据货币的开销。对于之前从 Aurora 存储中获取数据的查询，查询延迟可缩短高达 8 倍。

在 Aurora 中，默认参数组中 `shared_buffers` 的值通常设置为可用内存的 75% 左右。但是，对于 r6gd 和 r6id 实例类型，Aurora 将减少 4.5% 的 `shared_buffers` 空间，用于托管优化型读取缓存的元数据。

启用优化型读取的临时对象 - 使用临时对象，您可以通过将 PostgreSQL 生成的临时文件放置在本地 NVMe 存储上来实现更快的查询处理。这将减少通过网络流向 Elastic Block Storage（EBS）的流量。对于排序、联接或合并大量数据而不符合数据库实例上可用内存容量范围的高级查询，延迟可缩短高达 2 倍，吞吐量可提升高达 2 倍。

在 Aurora I/O 优化版集群上，优化型读取同时在 NVMe 存储上利用分层缓存和临时对象。借助启用了优化型读取的分层缓存功能，Aurora 为临时对象分配 2 倍的实例内存，为内部操作分配大约 10% 的存储，剩余的存储作为分层缓存。在 Aurora 标准集群上，优化型读取仅使用临时对象。

Aurora I/O 优化型集群可让您在实例级别使用动态参数 `aurora_temp_space_size`，来调整已启用优化型读取功能的临时对象的分配空间大小。可以在以下 PostgreSQL 版本中使用此调整大小功能：

- 16.8 及所有更高版本
- 15.12 及更高的 15 版本
- 14.17 及更高的 14 版本

使用此参数，无需重新启动数据库引擎，即可调整实例内存的容量（范围从 2 倍到最多 6 倍）。当您扩展临时对象空间时，无论并发工作负载如何，更改都会立即生效。但是，当您缩小空间时，只有当临时对象中有足够的未使用空间来容纳新的大小请求之后，调整才会完成。调整启用了优化型读取功能的临时对象的大小后，分层缓存会自动调整以使用任何可用空间。

Engine	集群存储配置	启用优化型读取的临时对象	启用优化型读取的分层缓存	支持的版本
Aurora PostgreSQL 兼容版	Standard	是	否	Aurora PostgreSQL 版本 16.1 及所有更高版本、15.4 及更高版本、版本 14.9 及更高版本
	I/O 优化版	支持	是	

> **Note**
>
> 在基于 NVMe 的数据库实例类上，在 IO 优化型集群和标准集群之间切换会导致数据库引擎立即重启。

在 Aurora PostgreSQL 中，使用 `temp_tablespaces` 参数配置存储临时对象的表空间。

要检查是否配置了临时对象，请使用以下命令：

```
postgres=> show temp_tablespaces;
temp_tablespaces
---------------------
aurora_temp_tablespace
(1 row)
```

`aurora_temp_tablespace` 是由 Aurora 配置的指向 NVMe 本地存储的表空间。您无法修改此参数或切换回 Amazon EBS 存储。

要检查优化型读取缓存是否已开启，请使用以下命令：

```
postgres=> show shared_preload_libraries;
                shared_preload_libraries
-----------------------------------------------------------
rdsutils,pg_stat_statements,aurora_optimized_reads_cache
```

使用 Aurora 优化型读取

当使用其中一个基于 NVMe 的数据库实例预调配 Aurora PostgreSQL 数据库实例时，该数据库实例会自动使用 Aurora 优化型读取功能。

要启用 Aurora 优化读取，请执行以下操作之一：

- 使用其中一个基于 NVMe 的数据库实例类创建 Aurora PostgreSQL 数据库集群。有关更多信息，请参阅 创建 Amazon Aurora 数据库集群。
- 修改现有 Aurora PostgreSQL 数据库集群，以使用其中一个基于 NVMe 的数据库实例类。有关更多信息，请参阅 修改 Amazon Aurora 数据库集群。

在支持其中一个或多个数据库实例类（具有本地 NVMe SSD 存储）的所有 AWS 区域中，均可使用 Aurora 优化型读取功能。有关更多信息，请参阅 Amazon Aurora 数据库实例类。

要切换回未优化读取功能的 Aurora 实例，请将 Aurora 实例的数据库实例类修改为类似的实例类，该实例类对于数据库工作负载没有 NVMe 临时存储。例如，如果当前数据库实例类是 db.r6gd.4xlarge，请选择 db.r6g.4xlarge 以切换回该实例类。有关更多信息，请参阅修改 Aurora 数据库实例。

Aurora 优化型读取的使用案例

启用优化型读取的分层缓存

以下是一些可从优化型读取分层缓存中受益的使用案例：

- Internet 规模的应用程序，例如支付处理、计费、电子商务（具有严格的性能 SLA）。
- 实时报告仪表板，可对指标/数据收集运行数百次单点查询。
- 带有 pgvector 扩展的生成式人工智能应用程序，可在数百万个向量嵌入中搜索精确或最近的邻居。

启用优化型读取的临时对象

以下是一些可从优化型读取临时对象中受益的使用案例：

- 包含公用表表达式（CTE）、派生表和分组操作的分析查询。
- 用于处理应用程序的未优化型查询的只读副本。
- 具有复杂操作（如 GROUP BY 和 ORDER BY）的按需或动态报告查询，这些操作无法始终使用适当的索引。
- 用于排序的 `CREATE INDEX` 或 `REINDEX` 操作。

- 使用内部临时表的其他工作负载。

监控使用 Aurora 优化读取的数据库实例

您可以使用 EXPLAIN 命令监控使用启用了优化型读取的分层缓存的查询，如以下示例所示：

```
Postgres=> EXPLAIN (ANALYZE, BUFFERS) SELECT c FROM sbtest15 WHERE id=100000000


QUERY PLAN
------------------------------------------------------------------------------------------
 Index Scan using sbtest15_pkey on sbtest15  (cost=0.57..8.59 rows=1 width=121) (actual
 time=0.287..0.288 rows=1 loops=1)
   Index Cond: (id = 100000000)
   Buffers: shared hit=3 read=2 aurora_orcache_hit=2
   I/O Timings: shared/local read=0.264
 Planning:
   Buffers: shared hit=33 read=6 aurora_orcache_hit=6
   I/O Timings: shared/local read=0.607
 Planning Time: 0.929 ms
 Execution Time: 0.303 ms
(9 rows)
Time: 2.028 ms
```

> **Note**
>
> 只有在优化型读取功能处于开启状态，并且解释计划的 Buffers 部分中 aurora_orcache_hit 和 aurora_storage_read 字段的值大于零时，才会显示这些字段。读取字段是 aurora_orcache_hit 和 aurora_storage_read 字段的总和。

您可以通过以下 CloudWatch 指标监控使用 Aurora 优化型读取功能的数据库实例：

- AuroraOptimizedReadsCacheHitRatio
- FreeEphemeralStorage
- ReadIOPSEphemeralStorage
- ReadLatencyEphemeralStorage

- ReadThroughputEphemeralStorage
- WriteIOPSEphemeralStorage
- WriteLatencyEphemeralStorage
- WriteThroughputEphemeralStorage

这些指标提供有关可用实例存储的存储空间、IOPS 和吞吐量的数据。有关这些指标的更多信息，请参阅 Amazon Aurora 的实例级指标。

您还可以使用此 pg_proctab 扩展来监控 NVMe 存储。

```
postgres=>select * from pg_diskusage();

major | minor |       devname       | reads_completed | reads_merged | sectors_read |
 readtime | writes_completed | writes_merged | sectors_written | writetime | current_io
 | iotime  | totaliotime
------+-------+---------------------+-----------------+--------------+--------------
+----------+------------------+---------------+-----------------+-----------
+------------+---------+-------------
      |       | rdstemp             |           23264 |            0 |       191450 |
  11670 |          1750892 |             0 |        24540576 |    819350 |          0 |
 3847580 |      831020
      |       | rdsephemeralstorage |           23271 |            0 |       193098 |
   2620 |           114961 |             0 |        13845120 |    130770 |          0 |
  215010 |      133410
(2 rows)
```

Aurora 优化型读取的最佳实践

对于 Aurora 优化型读取使用以下最佳实践：

- 使用 CloudWatch 指标 FreeEphemeralStorage 监控实例存储上的可用存储空间。如果由于数据库实例上的工作负载导致实例存储达到其限制，请调整并发性和大量使用临时对象的查询，或者将其修改为使用更大的数据库实例类。
- 监控 CloudWatch 指标的优化型读取缓存命中率。诸如 VACUUM 之类的操作可以非常快速地修改大量块。这可能会导致命中率暂时下降。pg_prewarm 扩展可用于将数据加载到缓冲区缓存中，从而允许 Aurora 主动将其中一些块写入优化型读取缓存。
- 您可以启用集群缓存管理（CCM），在将用作故障转移目标的 Tier-0 读取器上预热缓冲区缓存和分层缓存。启用 CCM 后，会定期扫描缓冲区缓存，以便在分层缓存中写入符合驱逐条件的页面。有关 CCM 的更多信息，请参阅 通过 Aurora PostgreSQL 的集群缓存管理提供故障转移后的快速恢复。

优化 Aurora PostgreSQL 中的关联子查询

关联子查询引用外部查询中的表列。对于外部查询返回的每一行，关联子查询都会被评估一次。在以下示例中，子查询引用表 ot 中的一列。此表未包含在子查询的 FROM 子句中，但在外部查询的 FROM 子句中引用了该表。如果表 ot 有 100 万行，则需要对子查询评估 100 万次。

```
SELECT ot.a, ot.b FROM ot WHERE ot.b < (SELECT AVG(it.b) FROM it WHERE it.a = ot.a);
```

> **Note**
>
> - 从版本 16.8 开始，Aurora PostgreSQL 中提供了子查询转换和子查询缓存，而适用于 Aurora PostgreSQL 的 Babelfish 从 4.2.0 开始支持这些功能。
> - 从适用于 Aurora PostgreSQL 的 Babelfish 版本 4.6.0 和 5.2.0 开始，以下参数控制这些功能：
>   - babelfishpg_tsql.apg_enable_correlated_scalar_transform
>   - babelfishpg_tsql.apg_enable_subquery_cache
>
>   默认情况下，这两个参数均处于开启状态。

使用子查询转换提高 Aurora PostgreSQL 查询性能

Aurora PostgreSQL 可以通过将关联子查询转换为等效的外部联接来加速这些子查询。此优化适用于以下两种类型的关联子查询：

- 返回单个聚合值并出现在 SELECT 列表中的子查询。

  ```
  SELECT ot.a, ot.b, (SELECT AVG(it.b) FROM it WHERE it.a = ot.a) FROM ot;
  ```

- 返回单个聚合值并出现在 WHERE 子句中的子查询。

  ```
  SELECT ot.a, ot.b FROM ot WHERE ot.b < (SELECT AVG(it.b) FROM it WHERE it.a = ot.a);
  ```

在子查询中启用转换

要支持将关联子查询转换为等效的外部联接，请将 `apg_enable_correlated_scalar_transform` 参数设置为 `ON`。此参数的默认值为 `OFF`。

您可以修改集群或实例参数组来设置参数。要了解更多信息，请参阅Amazon Aurora 的参数组。

或者，可以通过以下命令来仅为当前会话配置此设置：

```
SET apg_enable_correlated_scalar_transform TO ON;
```

验证转换

使用 EXPLAIN 命令来验证关联子查询是否已在查询计划中转换为外部联接。

启用转换后，适用的关联子查询部分将转换为外部联接。例如：

```
postgres=> CREATE TABLE ot (a INT, b INT);
CREATE TABLE
postgres=> CREATE TABLE it (a INT, b INT);
CREATE TABLE

postgres=> SET apg_enable_correlated_scalar_transform TO ON;
SET
postgres=> EXPLAIN (COSTS FALSE) SELECT ot.a, ot.b FROM ot WHERE ot.b < (SELECT
 AVG(it.b) FROM it WHERE it.a = ot.a);

                        QUERY PLAN
-----------------------------------------------------------------
 Hash Join
   Hash Cond: (ot.a = apg_scalar_subquery.scalar_output)
   Join Filter: ((ot.b)::numeric < apg_scalar_subquery.avg)
   ->  Seq Scan on ot
   ->  Hash
         ->  Subquery Scan on apg_scalar_subquery
               ->  HashAggregate
                     Group Key: it.a
                     ->  Seq Scan on it
```

将 GUC 参数设置为 OFF 时，不会转换相同的查询。计划将不会有外部联接，而是有子计划。

```
postgres=> SET apg_enable_correlated_scalar_transform TO OFF;
SET
postgres=> EXPLAIN (COSTS FALSE) SELECT ot.a, ot.b FROM ot WHERE ot.b < (SELECT
 AVG(it.b) FROM it WHERE it.a = ot.a);
                QUERY PLAN
---------------------------------------
```

```
Seq Scan on ot
  Filter: ((b)::numeric < (SubPlan 1))
  SubPlan 1
    ->  Aggregate
          ->  Seq Scan on it
                Filter: (a = ot.a)
```

限制

- 子查询必须位于 SELECT 列表中或 where 子句的其中一个条件中。否则，不会转换此子查询。
- 子查询必须返回一个聚合函数。不支持用户定义的聚合函数进行转换。
- 返回表达式不是简单聚合函数的子查询不会被转换。
- 子查询 WHERE 子句中的关联条件应该是简单的列引用。否则，不会转换此子查询。
- 子查询中的关联条件，其中子句必须是普通的相等谓词。
- 子查询不能包含 HAVING 或 GROUP BY 子句。
- 子查询中的 where 子句可能包含一个或多个使用 AND 组合的谓词。

> **Note**
>
> 转换对性能的影响因您的架构、数据和工作负载而异。随着外部查询生成的行数的增加，将子查询执行与转换关联可以显著提高性能。我们强烈建议您先在非生产环境中使用实际架构、数据和工作负载测试此功能，然后才在生产环境中启用。

使用子查询缓存提高 Aurora PostgreSQL 查询性能

Aurora PostgreSQL 支持子查询缓存来存储关联子查询的结果。当子查询结果已经在缓存中时，此功能会跳过重复的关联子查询执行。

了解子查询缓存

PostgreSQL 的 Memoize 节点是子查询缓存的关键部分。Memoize 节点在本地缓存中维护一个哈希表，用于从输入参数值映射到查询结果行。哈希表的内存限制是 work_mem 和 hash_mem_multiplier 的乘积。要了解更多信息，请参阅 Resource Consumption。

在查询执行期间，子查询缓存使用缓存命中率（CHR）来估计缓存是否提高了查询性能，并在查询运行时决定是否继续使用缓存。CHR 是缓存命中数与请求总数的比率。例如，如果关联子查询需要执行 100 次，并且可以从缓存中检索 70 个执行结果，则 CHR 为 0.7。

对于每个 apg_subquery_cache_check_interval 缓存未命中次数，可通过检查 CHR 是否大于 apg_subquery_cache_hit_rate_threshold 来评估子查询缓存的好处。否则，将从内存中删除缓存，查询执行将返回到原始、未缓存的子查询重新执行状态。

控制子查询缓存行为的参数

下表列出了控制子查询缓存行为的参数。

参数	描述	默认	允许
apg_enable_subquery_cache	支持对关联标量子查询使用缓存。	关闭	ON、OFF
apg_subquery_cache_check_interval	设置评估子查询缓存命中率的频率（以缓存未命中次数为单位）。	500	0–2147483647
apg_subquery_cache_hit_rate_threshold	设置子查询缓存命中率的阈值。	0.3	0.0–1.0

Note

- 较大的 apg_subquery_cache_check_interval 值可能会提高基于 CHR 的缓存优势估计的准确性，但会增加缓存开销，因为在缓存表具有 apg_subquery_cache_check_interval 行之前不会对 CHR 进行评估。
- 较大的 apg_subquery_cache_hit_rate_threshold 值偏向于放弃子查询缓存，而返回到原始、未缓存子查询重新执行状态。

您可以修改集群或实例参数组来设置参数。要了解更多信息，请参阅Amazon Aurora 的参数组。

或者，可以通过以下命令来仅为当前会话配置此设置：

```
SET apg_enable_subquery_cache TO ON;
```

在 Aurora PostgreSQL 中开启子查询缓存

启用子查询缓存后，Aurora PostgreSQL 会应用缓存来保存子查询结果。然后，查询计划将在 SubPlan 下有一个 Memoize 节点。

例如，以下命令序列显示了没有子查询缓存的简单关联子查询的估计查询执行计划。

```
postgres=> SET apg_enable_subquery_cache TO OFF;
SET
postgres=> EXPLAIN (COSTS FALSE) SELECT ot.a, ot.b FROM ot WHERE ot.b < (SELECT it.b
 FROM it WHERE it.a = ot.a);

             QUERY PLAN
-----------------------------------
 Seq Scan on ot
   Filter: (b < (SubPlan 1))
   SubPlan 1
     ->  Seq Scan on it
           Filter: (a = ot.a)
```

开启 apg_enable_subquery_cache 后，查询计划将在 SubPlan 节点下包含一个 Memoize 节点，表示子查询正计划使用缓存。

```
postgres=> SET apg_enable_subquery_cache TO ON;
SET
postgres=> EXPLAIN (COSTS FALSE) SELECT ot.a, ot.b FROM ot WHERE ot.b < (SELECT it.b
 FROM it WHERE it.a = ot.a);

             QUERY PLAN
-----------------------------------
 Seq Scan on ot
   Filter: (b < (SubPlan 1))
   SubPlan 1
     ->  Memoize
           Cache Key: ot.a
           Cache Mode: binary
           ->  Seq Scan on it
                 Filter: (a = ot.a)
```

实际的查询执行计划包含子查询缓存的更多详细信息，包括缓存命中和缓存未命中。以下输出显示了在向表中插入一些值后，上述示例查询的实际查询执行计划。

```
postgres=> EXPLAIN (COSTS FALSE, TIMING FALSE, ANALYZE TRUE) SELECT ot.a, ot.b FROM ot
 WHERE ot.b < (SELECT it.b FROM it WHERE it.a = ot.a);
            QUERY PLAN
-------------------------------------------------------------------------------
 Seq Scan on ot (actual rows=2 loops=1)
   Filter: (b < (SubPlan 1))
   Rows Removed by Filter: 8
   SubPlan 1
     ->  Memoize (actual rows=0 loops=10)
           Cache Key: ot.a
           Cache Mode: binary
           Hits: 4  Misses: 6  Evictions: 0  Overflows: 0  Memory Usage: 1kB
           ->  Seq Scan on it (actual rows=0 loops=6)
                 Filter: (a = ot.a)
                 Rows Removed by Filter: 4
```

缓存命中总数为 4，缓存未命中总数为 6。如果命中和未命中总数小于 Memoize 节点中的循环次数，则意味着 CHR 评估未通过，缓存已在某个时候被清理并放弃。然后，子查询执行返回到原始未缓存的重新执行状态。

限制

子查询缓存不支持关联子查询的某些规律。这些类型的查询将在没有缓存的情况下运行，即使子查询缓存已开启也是如此：

- IN/EXISTS/ANY/ALL 关联子查询
- 包含非确定性函数的关联子查询。
- 关联子查询，它们引用其数据类型不支持哈希或相等操作的外部表列。

使用自适应联接提高查询性能

概览

自适应联接是 Aurora PostgreSQL 17.4 中的一项预览功能，有助于提高查询性能。默认情况下，此功能处于禁用状态，但您可以使用全局用户配置（GUC）参数将其启用。由于这是一项预览功能，因此默认参数值可能会发生变化。启用后，自适应联接通过在运行时从嵌套循环联接动态切换到哈希联接，有助于优化查询性能。当 PostgreSQL 优化器由于基数估计值不准确而错误地选择了嵌套循环连接时，就会发生这种切换。

配置自适应联接

可以使用以下三个 GUC 参数控制自适应联接：

自适应联接配置参数

GUC 参数	描述	默认值和配置选项
apg_adaptive_join_crossover_multiplier	此乘数与行交叉点结合使用，以确定何时从嵌套循环切换到哈希联接。行交叉点是 SQL 优化器估计嵌套循环和哈希联接操作具有相等成本的地方。较高的乘数值会降低自适应联接切换到哈希连接的可能性。	控制是否启用自适应联接 • 默认值：-1（禁用） • 有效范围：-1 到 DBL_MAX • 要启用：设置为 >= 1
apg_adaptive_join_cost_threshold	此参数设置最低查询成本阈值。对于低于此阈值的查询，自适应联接会自动禁用。这样可以防止简单查询中的性能开销，在这种查询中，规划自适应联接的成本可能会超过从嵌套循环切换到哈希联接的优势。	设置查询的最低成本阈值 • 默认值：100 • 有效范围：0 到 DBL_MAX

apg_enable_parameterized_adaptive_join	启用后，此参数将自适应联接功能扩展到参数化的嵌套循环联接。默认情况下，自适应联接仅适用于未参数化的嵌套循环联接，因为这些联接更有可能从切换到哈希联接而受益。参数化嵌套循环联接通常性能更好，而使得切换到哈希联接并不那么重要。	控制嵌套循环联接的自适应联接行为
		• 默认值：false
		• 有效值：true/false
		  • 当为 false 时：仅适用于未参数化的嵌套循环联接
		  • 当为 true 时：同时适用于参数化和非参数化的嵌套循环联接
		Note
		要求先启用 apg_adapt

GUC 参数	描述	默认值和配置选项
		ive_join_ crossover _multiplier

在 Aurora PostgreSQL 中使用未记录的表

Amazon Aurora PostgreSQL 支持未记录的表，这些表具有崩溃安全性，即使在出现写入器实例故障或进行失效转移后也能保持数据完整性。在标准 PostgreSQL 中，未记录的表在写入操作期间会绕过预写日志（WAL），从而提高写入速度。但是，这是以降低持久性为代价的，因为未记录的表不具有崩溃安全性，在系统故障或不正常关闭后可能会丢失数据。这些未记录的表会在崩溃或不正常关闭后自动截断。它们的内容和索引也不会复制到备用服务器。

相比之下，Aurora PostgreSQL 由于其分布式存储架构，处理未记录的表的方式有所不同。这是因为 Aurora 的存储系统不依赖于传统的 PostgreSQL WAL 来实现持久性。然而，在标准 PostgreSQL 中通常与未记录的表关联的性能优势在 Aurora 中可能并不那么明显。这归因于 Aurora 的分布式存储架构，与标准 PostgreSQL 中使用的本地存储相比，这种架构可能会带来额外的开销。

在 Aurora PostgreSQL 中使用未记录的表时，请注意以下事项：

- 您只能从 Aurora 数据库集群中的写入器节点访问未记录的表。
- 读取器节点只有在提升到写入器状态时，才能访问未记录的表。
- 当您尝试从读取器节点访问未记录的表时，将导致以下错误：

```
cannot access temporary or unlogged relations during recovery.
```

创建未记录的表

要在 Aurora PostgreSQL 中创建未记录的表，请在 CREATE TABLE 语句中添加 UNLOGGED 关键字：

```
CREATE UNLOGGED TABLE staging_sales_data (
    transaction_id bigint,
    customer_id bigint,
    product_id bigint,
```

```
    transaction_date date,
    amount NUMERIC
);
```

将未记录的表转换为已记录的表

当您需要将未记录的表转换回已记录的表时，可以使用以下命令：

```
ALTER TABLE table_name SET LOGGED;
```

此操作会重写整个表，并在其上放置排他锁，直到操作完成。对于大型表，这可能会导致大量的停机时间。

未记录的表和逻辑复制

逻辑复制中通常不包括未记录的表，因为逻辑复制依赖于 WAL 来捕获和传输更改。默认情况下，对未记录的表进行的更改不会记录 WAL 并被排除在复制流之外，这使得它们不适合需要逻辑复制的用例。但是，Aurora PostgreSQL 提供了一个名为 `rds.logically_replicate_unlogged_tables` 的参数，让您可以控制这种行为：

- 当 `rds.logically_replicate_unlogged_tables` 设置为 0（关闭）时，将从逻辑复制中排除未记录的表。
- 当 `rds.logically_replicate_unlogged_tables` 设置为 1（开启）时，将在逻辑复制中包含未记录的表。

> **Note**
>
> 在 Aurora PostgreSQL 中，`rds.logically_replicate_unlogged_tables` 参数在版本 14 及更早版本中默认设置为 1（开启），而在版本 15 及更高版本中默认设置为 0（关闭）。

在 Amazon Aurora PostgreSQL 上使用 PostgreSQL autovacuum

我们强烈建议您使用 Autovacuum 功能来保持 PostgreSQL 数据库实例正常运行。Autovacuum 自动启动 VACUUM 和 ANALYZE 命令。它会检查包含大量插入的、更新的或删除的元组的表。进行此检查后，它会通过从 PostgreSQL 数据库中删除过时的数据或元组来回收存储。

在使用任何默认 PostgreSQL 数据库参数组创建的 Aurora PostgreSQL 数据库实例上，会默认启用 autovacuum。默认情况下，还会设置与 Autovacuum 功能关联的其他配置参数。这些默认值是通用值，因此可以针对特定工作负载优化与 Autovacuum 功能关联的某些参数。

在下文中，您可以了解有关 autovacuum 功能以及如何为 Aurora PostgreSQL 数据库实例优化其部分参数的更多信息。

主题

- 为 Autovacuum 分配内存
- 减少事务 ID 重叠的可能性
- 确定数据库中的表是否需要 vacuum 操作
- 确定哪些表当前符合 Autovacuum 条件
- 确定 Autovacuum 当前是否正在运行以及运行时长
- 执行手动 vacuum 冻结
- 在 Autovacuum 正在运行时重新为表建立索引
- 使用大型索引管理 autovacuum
- 其他影响 Autovacuum 的参数
- 设置表级别 Autovacuum 参数
- 记录 autovacuum 和 vacuum 活动
- 了解对无效数据库使用 autovacuum 的行为
- 识别并解决 Aurora PostgreSQL 中的积极真空拦截器

为 Autovacuum 分配内存

影响Autovacuum 性能的最重要参数之一是 `autovacuum_work_mem` 参数。在 Aurora PostgreSQL 版本 14 及更早版本中，`autovacuum_work_mem` 参数设置为 -1，表示改用 `maintenance_work_mem` 的设置。对于所有其它版本，`autovacuum_work_mem` 由 GREATEST({DBInstanceClassMemory/32768}, 65536) 决定。

手动 vacuum 操作始终使用 maintenance_work_mem 设置，默认设置为 GREATEST({DBInstanceClassMemory/63963136*1024}, 65536)，也可以使用 SET 命令在会话级别对其进行调整，以进行更有针对性的手动 VACUUM 操作。

autovacuum_work_mem 确定 autovacuum 在对索引执行 vacuum 操作时，用于容纳无效元组 (pg_stat_all_tables.n_dead_tup) 的标识符的内存。

当执行计算以确定 autovacuum_work_mem 参数的值时，请注意以下事项：

- 如果将此参数设得过低，则 vacuum 过程可能必须扫描表多次才能完成其工作。此类多次扫描可能会对性能产生负面影响。对于较大的实例，将 maintenance_work_mem 或 autovacuum_work_mem 设置为至少 1 GB，可以提高对包含大量无效元组的表执行 vacuum 操作的性能。但是，在 PostgreSQL 版本 16 及更早版本中，vacuum 的内存使用量上限为 1 GB，这足以单次处理大约 1.79 亿个无效元组。如果表的无效元组比这更多，vacuum 将需要多次遍历表的索引，从而显著增加所需的时间。从 PostgreSQL 版本 17 开始，没有 1 GB 的限制，自动真空可以使用基数树处理超过 1.79 亿个元组。

  元组标识符的大小为 6 个字节。要估算对表的索引执行 vacuum 操作所需的内存，请查询 pg_stat_all_tables.n_dead_tup 来找到无效元组的数量，然后将该数字乘以 6，以确定单次对索引执行 vacuum 操作所需的内存。您可以使用以下查询：

  ```
  SELECT
      relname AS table_name,
      n_dead_tup,
      pg_size_pretty(n_dead_tup * 6) AS estimated_memory
  FROM
      pg_stat_all_tables
  WHERE
      relname = 'name_of_the_table';
  ```

- autovacuum_work_mem 参数可与 autovacuum_max_workers 参数结合使用。autovacuum_max_workers 中的每个工作进程均可使用您分配的内存。如果您有多个小型表，请分配更多的 autovacuum_max_workers 和更少的 autovacuum_work_mem。如果您拥有大型表（大于 100 GB），请分配更多内存和更少的工作进程。您需要分配有足够的内存才能对最大的表成功完成操作。因此，请确保工作进程和内存的组合等于要分配的总内存。

减少事务 ID 重叠的可能性

在一些情况下，与 Autovacuum 相关的参数组设置可能不够积极，无法阻止事务 ID 重叠。为解决此问题，Aurora PostgreSQL 提供了自动调整 Autovacuum 参数值的机制。适应性 autovacuum 是 Aurora PostgreSQL 的一项功能。在 PostgreSQL 文档中找到了 TransactionID 重叠的详细说明。

默认情况下，动态参数 `rds.adaptive_autovacuum` 设置为 ON 的 Aurora PostgreSQL 实例启用适应性 autovacuum。强烈建议您保持启用此选项。不过，要关闭适应性 Autovacuum 参数优化，请将 `rds.adaptive_autovacuum` 参数设置为 0 或 OFF。

即使在 Aurora 优化 Autovacuum 参数时，仍可能出现事务 ID 重叠。鼓励您为事务 ID 重叠实施 Amazon CloudWatch 警报。有关更多信息，请参阅 AWS 数据库博客上的贴子在 RDS for PostgreSQL 中为事务 ID 重叠实施预警系统。

启用适应性 autovacuum 参数优化后，当 CloudWatch 指标 `MaximumUsedTransactionIDs` 达到 `autovacuum_freeze_max_age` 参数值或 500000000 中的较大值时，Amazon RDS 将开始调整 autovacuum 参数。

如果表继续倾向于事务 ID 重叠，则 Amazon RDS 将继续调整 autovacuum 的参数。其中每次调整都会将更多资源专用于 Autovacuum 以避免重叠。Amazon RDS 更新以下与 Autovacuum 相关的参数：

- Autovacuum_vacuum_cost_delay
- Autovacuum_vacuum_cost_limit
- `autovacuum_work_mem`
- Autovacuum_naptime

仅当新值使 Autovacuum 更积极时，RDS 才会修改这些参数。在数据库实例上的内存中修改参数。不会更改参数组中的值。要查看当前内存中的设置，请使用 PostgreSQL SHOW SQL 命令。

当 Amazon RDS 修改其中任何 autovacuum 参数时，它会为受影响的数据库实例生成事件。此事件在 AWS Management Console 上和通过 Amazon RDS API 显示。在 `MaximumUsedTransactionIDs` CloudWatch 指标返回的值低于阈值后，Amazon RDS 会将内存中与 autovacuum 相关的参数重置回参数组中指定的值。然后，它会生成另一个与此更改对应的事件。

确定数据库中的表是否需要 vacuum 操作

您可以使用以下查询显示数据库中解冻事务的数目。数据库的 `datfrozenxid` 行的 `pg_database` 列是显示在该数据库中的正常事务 ID 的下限。此列是数据库中每个表的 `relfrozenxid` 值的最小值。

```
SELECT datname, age(datfrozenxid) FROM pg_database ORDER BY age(datfrozenxid) desc
 limit 20;
```

例如，运行上述查询的结果可能如下所示。

```
datname    | age
mydb       | 1771757888
template0  | 1721757888
template1  | 1721757888
rdsadmin   | 1694008527
postgres   | 1693881061
(5 rows)
```

当数据库的期限达到 20 亿个事务 ID 时，事务 ID (XID) 重叠将出现，并且数据库将变成只读状态。您可以使用此查询来生成指标，并且一天可运行几次。默认情况下，将设置 Autovacuum 以确保事务期限不超过 200000000 ()。autovacuum_freeze_max_age

示例监控策略可能类似于：

- 将 autovacuum_freeze_max_age 值设置为 2 亿个事务。
- 如果表中的解冻事务达到 5 亿个，则会触发低严重性警报。这不是一个不合理的值，但它可能指示 Autovacuum 未保持同步。
- 如果表期限为 10 亿，这应被视为要采取操作的警报。通常，您出于性能原因，需要使期限更接近 autovacuum_freeze_max_age。建议您使用以下建议进行调查。
- 如果表达到 15 亿个未执行 vacuum 操作的事务，则这会触发高严重性警报。根据数据库使用事务 ID 的频率，此警报将指示系统运行 Autovacuum 的时间不多了。在这种情况下，建议您立即解决此问题。

如果表持续违反这些阈值，请进一步修改 autovacuum 参数。默认情况下，手动使用 VACUUM（已禁用基于成本的延迟）比使用默认的 Autovacuum 更积极，但对整个系统来说也更具侵入性。

我们建议执行下列操作：

- 了解和启用监控机制，以便您了解最早的事务的期限。

  有关创建提醒您事务 ID 重叠的过程的信息，请参阅 AWS 数据库博客帖子 Implement an early warning system for transaction ID wraparound in Amazon RDS for PostgreSQL。

- 对于更复杂的表，在维护时段内定期执行手动 vacuum 冻结操作，并依赖 Autovacuum。有关执行手动 vacuum 冻结的信息，请参阅执行手动 vacuum 冻结。

确定哪些表当前符合 Autovacuum 条件

通常，它是需要执行 vacuum 操作的一个或两个表。其 `relfrozenxid` 值大于 `autovacuum_freeze_max_age` 中的事务数的表始终是 Autovacuum 的目标。否则，如果元组数因上一个 VACUUM 超出 vacuum 阈值而变得过时，则对表执行 vacuum 操作。

Autovacuum 阈值的定义如下：

```
Vacuum-threshold = vacuum-base-threshold + vacuum-scale-factor * number-of-tuples
```

其中，`vacuum base threshold` 为 `autovacuum_vacuum_threshold`，`vacuum scale factor` 为 `autovacuum_vacuum_scale_factor`，而 `number of tuples` 为 `pg_class.reltuples`。

在连接到数据库时，运行以下查询可查看 autovacuum 认为有资格执行 vacuum 操作的表的列表。

```
WITH vbt AS (SELECT setting AS autovacuum_vacuum_threshold FROM
pg_settings WHERE name = 'autovacuum_vacuum_threshold'),
vsf AS (SELECT setting AS autovacuum_vacuum_scale_factor FROM
pg_settings WHERE name = 'autovacuum_vacuum_scale_factor'),
fma AS (SELECT setting AS autovacuum_freeze_max_age FROM pg_settings WHERE name =
 'autovacuum_freeze_max_age'),
sto AS (select opt_oid, split_part(setting, '=', 1) as param,
split_part(setting, '=', 2) as value from (select oid opt_oid, unnest(reloptions)
 setting from pg_class) opt)
SELECT '"'||ns.nspname||'"."'||c.relname||'"' as relation,
pg_size_pretty(pg_table_size(c.oid)) as table_size,
age(relfrozenxid) as xid_age,
coalesce(cfma.value::float, autovacuum_freeze_max_age::float)
 autovacuum_freeze_max_age,
(coalesce(cvbt.value::float, autovacuum_vacuum_threshold::float) +
coalesce(cvsf.value::float,autovacuum_vacuum_scale_factor::float) * c.reltuples)
AS autovacuum_vacuum_tuples, n_dead_tup as dead_tuples FROM
pg_class c join pg_namespace ns on ns.oid = c.relnamespace
join pg_stat_all_tables stat on stat.relid = c.oid join vbt on (1=1) join vsf on (1=1)
 join fma on (1=1)
left join sto cvbt on cvbt.param = 'autovacuum_vacuum_threshold' and c.oid =
 cvbt.opt_oid
```

```
left join sto cvsf on cvsf.param = 'autovacuum_vacuum_scale_factor' and c.oid =
 cvsf.opt_oid
left join sto cfma on cfma.param = 'autovacuum_freeze_max_age' and c.oid = cfma.opt_oid
WHERE c.relkind = 'r' and nspname <> 'pg_catalog'
AND (age(relfrozenxid) >= coalesce(cfma.value::float, autovacuum_freeze_max_age::float)
OR coalesce(cvbt.value::float, autovacuum_vacuum_threshold::float) +
coalesce(cvsf.value::float,autovacuum_vacuum_scale_factor::float) *
c.reltuples <= n_dead_tup)
ORDER BY age(relfrozenxid) DESC LIMIT 50;
```

确定 Autovacuum 当前是否正在运行以及运行时长

如果需要手动对表执行 vacuum 操作，确保确定 autovacuum 当前是否正在运行。如果它当前正在运行，则您可能需要调整参数以使其更高效地运行，或者暂时禁用 autovacuum 以便手动运行 VACUUM。

使用以下查询来确定 Autovacuum 是否正在运行、它已运行的时长以及它是否正在等待其他会话。

```
SELECT datname, usename, pid, state, wait_event, current_timestamp - xact_start AS
 xact_runtime, query
FROM pg_stat_activity
WHERE upper(query) LIKE '%VACUUM%'
ORDER BY xact_start;
```

在运行查询后，您应看到类似以下内容的输出。

```
 datname | usename  |  pid  | state  | wait_event |      xact_runtime       | query
 --------+----------+-------+--------+------------+-------------------------
+-------------------------------------------------------------------------------------------
 mydb    | rdsadmin | 16473 | active |            | 33 days 16:32:11.600656 |
 autovacuum: VACUUM ANALYZE public.mytable1 (to prevent wraparound)
 mydb    | rdsadmin | 22553 | active |            | 14 days 09:15:34.073141 |
 autovacuum: VACUUM ANALYZE public.mytable2 (to prevent wraparound)
 mydb    | rdsadmin | 41909 | active |            | 3 days 02:43:54.203349  |
 autovacuum: VACUUM ANALYZE public.mytable3
 mydb    | rdsadmin |   618 | active |            | 00:00:00                |
 SELECT datname, usename, pid, state, wait_event, current_timestamp - xact_start AS
 xact_runtime, query+
         |          |       |        |            |                         | FROM
 pg_stat_activity
           +
```

```
        |          |       |        |            |                          | WHERE
query like '%VACUUM%'
         +
        |          |       |        |            |                          | ORDER BY
xact_start;
      +
```

有多个问题可能会导致长时间运行 Autovacuum 会话（即，多天）。最常见的问题是，对于表的大小或更新速率来说，设置的 maintenance_work_mem 参数值太小。

建议您使用以下公式来设置 maintenance_work_mem 参数值。

```
GREATEST({DBInstanceClassMemory/63963136*1024},65536)
```

短时间运行的 Autovacuum 会话还可以指示以下问题：

- 它可以指示，对于工作负载而言，autovacuum_max_workers 不足。在这种情况下，您将需要指示工作线程数。
- 它可以指示存在索引损坏（autovacuum 将发生崩溃并在同一关系上重新启动，但毫无进展）。在这种情况下，运行手动 vacuum freeze verbose *table* 以查看准确原因。

执行手动 vacuum 冻结

您可能需要对已具有正在运行的 vacuum 进程的表执行手动 vacuum 操作。如果您已使用接近 20 亿个事务（或高于您监控的任何阈值）的期限标识表，则这会很有用。

以下步骤是指导原则，此过程存在几种变化。例如，在测试期间，假设您发现设定的 maintenance_work_mem 参数值过小，并且您需要立即对表采取措施。不过，可能您不希望此时恢复实例。通过使用前几节中的查询，您可以确定哪个表存在问题，并找到长时间运行的 Autovacuum 会话。您知道您需要更改 maintenance_work_mem 参数设置，但您还需要立即采取行动，对有问题的表执行 vacuum 操作。以下过程说明了在此情况下应采取的措施。

手动执行 vacuum 冻结

1. 打开针对包含要执行 vacuum 操作的表的数据库的两个会话。对于第二个会话，使用“screen”或其他维护会话的实用工具 (如果您的连接已中断)。
2. 在第一个会话中，获取正在表上运行的 autovacuum 会话的进程 ID (PID)。

   运行以下查询可获取 Autovacuum 会话的 PID。

```
SELECT datname, usename, pid, current_timestamp - xact_start
AS xact_runtime, query
FROM pg_stat_activity WHERE upper(query) LIKE '%VACUUM%' ORDER BY
xact_start;
```

3. 在第二个会话中，计算该操作所需的内存量。在此示例中，我们确定自己最多可以为该操作使用 2GB 的内存，因此，我们将当前会话的 maintenance_work_mem 设置为 2 GB。

```
SET maintenance_work_mem='2 GB';
SET
```

4. 在第二个会话中，为表发出 vacuum freeze verbose 命令。详细设置很有用，因为虽然 PostgreSQL 中当前没有进度报告，但您可以查看活动。

```
\timing on
Timing is on.
vacuum freeze verbose pgbench_branches;
```

```
INFO:  vacuuming "public.pgbench_branches"
INFO:  index "pgbench_branches_pkey" now contains 50 row versions in 2 pages
DETAIL:  0 index row versions were removed.
0 index pages have been deleted, 0 are currently reusable.
CPU 0.00s/0.00u sec elapsed 0.00 sec.
INFO:  index "pgbench_branches_test_index" now contains 50 row versions in 2 pages
DETAIL:  0 index row versions were removed.
0 index pages have been deleted, 0 are currently reusable.
CPU 0.00s/0.00u sec elapsed 0.00 sec.
INFO:  "pgbench_branches": found 0 removable, 50 nonremovable row versions
     in 43 out of 43 pages
DETAIL:  0 dead row versions cannot be removed yet.
There were 9347 unused item pointers.
0 pages are entirely empty.
CPU 0.00s/0.00u sec elapsed 0.00 sec.
VACUUM
Time: 2.765 ms
```

5. 在第一个会话中，如果 autovacuum 阻止 vacuum 会话，pg_stat_activity 显示 vacuum 会话的等待为 T。在此情况下，您需要终止 autovacuum 进程，如下所示。

```
SELECT pg_terminate_backend('the_pid');
```

> **Note**
>
> 某些较低版本的 Amazon Aurora 无法使用上述命令终止 autovacuum 进程，并且会失败，显示以下错误：`ERROR: 42501: must be a superuser to terminate superuser process LOCATION: pg_terminate_backend, signalfuncs.c:227`。要查找已修补的 PostgreSQL 版本，请在 Amazon Aurora PostgreSQL updates中搜索以下公告：
>
> ```
> Allow rds_superuser to terminate backends which are not explicitly
>  associated with a role
> ```

此时，您的会话将开始。由于此表可能位于其工作列表中的最高位置，因此 Autovacuum 将立即重新启动。

6. 在第二个会话中启动您的 `vacuum freeze verbose` 命令，然后终止第一个会话中的 autovacuum 过程。

在 Autovacuum 正在运行时重新为表建立索引

如果索引已损坏，Autovacuum 将继续处理表并失败。在此情况下，如果您尝试执行手动 vacuum 操作，您将收到一条与以下内容类似的错误消息。

```
postgres=>  vacuum freeze pgbench_branches;
ERROR: index "pgbench_branches_test_index" contains unexpected
   zero page at block 30521
HINT: Please REINDEX it.
```

当索引损坏并且 autovacuum 尝试在表上运行时，您将处理已经正在运行的 autovacuum 会话。在您发出 REINDEX 命令时，将删除表上的排他锁。将阻止写入操作以及使用该特定索引的读取操作。

在对表运行 Autovacuum 时重新为表建立索引

1. 打开针对包含要执行 vacuum 操作的表的数据库的两个会话。对于第二个会话，使用“screen”或其他维护会话的实用工具 (如果您的连接已中断)。
2. 在第一个会话中，获取正在表上运行的 Autovacuum 会话的 PID。

   运行以下查询可获取 Autovacuum 会话的 PID。

```
SELECT datname, usename, pid, current_timestamp - xact_start
AS xact_runtime, query
FROM pg_stat_activity WHERE upper(query) like '%VACUUM%' ORDER BY
xact_start;
```

3. 在第二个会话中，发出 reindex 命令。

```
\timing on
Timing is on.
reindex index pgbench_branches_test_index;
REINDEX
  Time: 9.966 ms
```

4. 在第一个会话中，如果 autovacuum 阻止该过程，您将在 pg_stat_activity 中看到 vacuum 会话的等待为“T”。在此情况下，您将终止 autovacuum 过程。

```
SELECT pg_terminate_backend('the_pid');
```

此时，您的会话将开始。由于此表可能位于其工作列表中的最高位置，因此，了解 Autovacuum 将立即重新启动很重要。

5. 在第二个会话中启动您的命令，然后终止第一个会话中的 autovacuum 过程。

使用大型索引管理 autovacuum

作为其操作的一部分，在对表运行 autovacuum 时会执行多个 vacuum 阶段。在清理表之前，首先对所有索引执行 vacuum 操作。删除多个大型索引时，此阶段会消耗大量的时间和资源。因此，作为最佳实践，请务必控制表上的索引数量并消除未使用的索引。

对于此过程，请先检查总体索引大小。然后，确定是否存在可以删除的潜在未使用索引，如以下示例所示。

检查表及其索引的大小

```
postgres=> select pg_size_pretty(pg_relation_size('pgbench_accounts'));
pg_size_pretty
6404 MB
(1 row)
```

```
postgres=> select pg_size_pretty(pg_indexes_size('pgbench_accounts'));
```

```
pg_size_pretty
11 GB
(1 row)
```

在此示例中，索引的大小大于表的大小。这种差异可能会导致性能问题，因为索引膨胀或未使用，这会影响 autovacuum 以及插入操作。

检查是否有未使用的索引

使用 pg_stat_user_indexes 视图，您可以检查 idx_scan 列使用索引的频率。在以下示例中，未使用索引的 idx_scan 值为 0。

```
postgres=> select * from pg_stat_user_indexes where relname = 'pgbench_accounts' order
 by idx_scan desc;

relid  | indexrelid | schemaname | relname          | indexrelname          | idx_scan
 | idx_tup_read | idx_tup_fetch
-------+------------+------------+------------------+-----------------------+----------
+--------------+---------------
16433  | 16454      | public     | pgbench_accounts | index_f               | 6
 | 6            | 0
16433  | 16450      | public     | pgbench_accounts | index_b               | 3
 | 199999       | 0
16433  | 16447      | public     | pgbench_accounts | pgbench_accounts_pkey | 0
 | 0            | 0
16433  | 16452      | public     | pgbench_accounts | index_d               | 0
 | 0            | 0
16433  | 16453      | public     | pgbench_accounts | index_e               | 0
 | 0            | 0
16433  | 16451      | public     | pgbench_accounts | index_c               | 0
 | 0            | 0
16433  | 16449      | public     | pgbench_accounts | index_a               | 0
 | 0            | 0
(7 rows)
```

```
postgres=> select schemaname, relname, indexrelname, idx_scan from pg_stat_user_indexes
 where relname = 'pgbench_accounts' order by idx_scan desc;

schemaname  | relname          | indexrelname          | idx_scan
------------+------------------+-----------------------+----------
public      | pgbench_accounts | index_f               | 6
public      | pgbench_accounts | index_b               | 3
```

```
public      | pgbench_accounts | pgbench_accounts_pkey | 0
public      | pgbench_accounts | index_d               | 0
public      | pgbench_accounts | index_e               | 0
public      | pgbench_accounts | index_c               | 0
public      | pgbench_accounts | index_a               | 0
(7 rows)
```

> **Note**
>
> 这些统计数据自统计数据重置之时起开始递增。假设您的索引仅用于业务季度末或仅用于特定报告。自从统计数据重置后，此索引可能就没有使用过。有关更多信息，请参阅统计数据函数。用于强制唯一性的索引不会执行扫描，也不应被标识为未使用的索引。要识别未使用的索引，您应该对应用程序及其查询有深入的了解。

要检查数据库上次重置统计数据的时间，请使用 pg_stat_database

```
postgres=> select datname, stats_reset from pg_stat_database where datname =
 'postgres';

datname   | stats_reset
----------+-------------------------------
postgres  | 2022-11-17 08:58:11.427224+00
(1 row)
```

尽快对表执行 vacuum 操作

RDS for PostgreSQL 12 及更高版本

如果大型表中的索引过多，则您的数据库实例可能接近事务 ID 重叠（XID），也就是 XID 计数器变为零时。如果不进行检查，这种情况可能导致数据丢失。但是，您可以在不清理索引的情况下快速对表执行 vacuum 操作。在 RDS for PostgreSQL 12 及更高版本中，可以将 VACUUM 与 INDEX_CLEANUP 子句结合使用。

```
postgres=> VACUUM (INDEX_CLEANUP FALSE, VERBOSE TRUE) pgbench_accounts;

INFO: vacuuming "public.pgbench_accounts"
INFO: table "pgbench_accounts": found 0 removable, 8 nonremovable row versions in 1 out
 of 819673 pages
```

```
DETAIL: 0 dead row versions cannot be removed yet, oldest xmin: 7517
Skipped 0 pages due to buffer pins, 0 frozen pages.
CPU: user: 0.01 s, system: 0.00 s, elapsed: 0.01 s.
```

如果 autovacuum 会话已在运行，则必须将其终止才能开始手动 VACUUM。有关执行手动 vacuum 冻结的信息，请参阅执行手动 vacuum 冻结。

> **Note**
>
> 定期跳过索引清理会导致索引膨胀，这会降低扫描性能。索引保留死行，而表保留死行指针。因此，pg_stat_all_tables.n_dead_tup 会一直增加，直到 autovacuum 或带有索引清理功能的手动 VACUUM 运行。作为最佳实践，请仅使用此过程来防止事务 ID 重叠。

RDS for PostgreSQL 11 及更低版本

但是，在 RDS for PostgreSQL 11 及更低版本中，让 vacuum 过程更快地完成的唯一方法是减少表上的索引数量。删除索引可能会影响查询计划。我们建议您先删除未使用的索引，然后在 XID 重叠非常接近时删除索引。vacuum 过程完成后，您可以重新创建这些索引。

其他影响 Autovacuum 的参数

以下查询将显示直接影响 Autovacuum 及其行为的一些参数的值。PostgreSQL 文档中完整介绍了 Autovacuum 参数。

```
SELECT name, setting, unit, short_desc
FROM pg_settings
WHERE name IN (
'autovacuum_max_workers',
'autovacuum_analyze_scale_factor',
'autovacuum_naptime',
'autovacuum_analyze_threshold',
'autovacuum_analyze_scale_factor',
'autovacuum_vacuum_threshold',
'autovacuum_vacuum_scale_factor',
'autovacuum_vacuum_threshold',
'autovacuum_vacuum_cost_delay',
'autovacuum_vacuum_cost_limit',
'vacuum_cost_limit',
'autovacuum_freeze_max_age',
```

```
'maintenance_work_mem',
'vacuum_freeze_min_age');
```

所有这些参数都会影响 Autovacuum，其中一些最重要的参数为：

- maintenance_work_mem
- Autovacuum_freeze_max_age
- Autovacuum_max_workers
- Autovacuum_vacuum_cost_delay
- Autovacuum_vacuum_cost_limit

设置表级别 Autovacuum 参数

您可以在表级别设置与 autovacuum 相关的存储参数，这可能优于更改整个数据库的行为。对于大型表，您可能需要设置主动设置，并且可能不希望 Autovacuum 对所有表的行为都相同。

以下查询将显示哪些表当前拥有表级别选项。

```
SELECT relname, reloptions
FROM pg_class
WHERE reloptions IS NOT null;
```

例如，对于比您的其他表大得多的表，这可能会很有用。假设您具有一个 300GB 表和另外 30 个小于 1GB 的表。在这种情况下，可以为大型表设置一些特定的参数，这样便无需更改整个系统的行为。

```
ALTER TABLE mytable set (autovacuum_vacuum_cost_delay=0);
```

这样便可通过使用系统中的更多资源来禁用此表的基于成本的 autovacuum 延迟。通常，每次达到 `autovacuum_cost_limit` 时，autovacuum 会暂停 `autovacuum_vacuum_cost_delay`。有关更多详细信息，请参阅 PostgreSQL 文档中有关基于成本的 vacuum 操作的内容。

记录 autovacuum 和 vacuum 活动

根据 `rds.force_autovacuum_logging_level` 参数中指定的级别，有关 autovacuum 活动的信息将发送到 `postgresql.log`。以下是此参数允许的值以及该值为默认设置的 PostgreSQL 版本：

- `disabled`（PostgreSQL 10、PostgreSQL 9.6）
- `debug5`, `debug4`, `debug3`, `debug2`, `debug1`

- info（PostgreSQL 12、PostgreSQL 11）
- notice
- warning（PostgreSQL 13 及更高版本）
- error、日志、fatal、panic

rds.force_autovacuum_logging_level 与 log_autovacuum_min_duration 参数结合使用。log_autovacuum_min_duration 参数的值为阈值（以毫秒为单位），超出该值后将记录 autovacuum 操作。设置为 -1 不会记录任何内容，而设置 0 将记录所有操作。和 rds.force_autovacuum_logging_level 一样，log_autovacuum_min_duration 的默认值取决于版本，如下所示：

- 10000 ms - PostgreSQL 14、PostgreSQL 13、PostgreSQL 12 和 PostgreSQL 11
- (empty) - PostgreSQL 10 和 PostgreSQL 9.6 没有默认值

建议您将 rds.force_autovacuum_logging_level 设置为 WARNING。我们还建议您将 log_autovacuum_min_duration 设置为 1000 到 5000 之间的值。设置为 5000 的记录活动耗时将超过 5000 毫秒。如果由于冲突锁定或并行删除关系跳过了 autovacuum 操作，则并非 -1 的任何其他设置也会记录消息。有关更多信息，请参阅 PostgreSQL 文档中的 Automatic Vacuuming。

要对问题进行故障排除，可以将 rds.force_autovacuum_logging_level 参数更改为调试级别之一，从 debug1 到最高 debug5 以获取最详细的信息。我们建议您在短时间内使用调试设置，并且仅用于故障排除目的。要了解更多信息，请参阅 PostgreSQL 文档中的何时记录。

> **Note**
>
> PostgreSQL 允许 rds_superuser 账户查看 pg_stat_activity 中的 Autovacuum 会话。例如，您可识别并结束阻止命令运行或运行速度慢于手动发出的 vacuum 命令的 Autovacuum 会话。

了解对无效数据库使用 autovacuum 的行为

在 pg_database 目录的 datconnlimit 列中引入了一个新值 -2，来指示在 DROP DATABASE 操作过程中已被中断的数据库为无效的。

以下 Aurora PostgreSQL 版本中提供了这个新值：

- 15.4 及所有更高版本
- 14.9 及更高版本
- 13.12 及更高版本
- 12.16 及更高版本
- 11.21 及更高版本

无效的数据库不会影响 autovacuum 冻结有效数据库的功能的能力。Autovacuum 会忽略无效的数据库。因此，对于 PostgreSQL 环境中的所有有效数据库，常规 vacuum 操作将继续正常且高效地运行。

主题

- 监控事务 ID
- 调整监控查询
- 解决无效的数据库问题

监控事务 ID

`age(datfrozenxid)` 函数常用于监控数据库的事务 ID（XID）期限，以防止事务 ID 重叠。

由于无效数据库被排除在 autovacuum 之外，因此其事务 ID（XID）计数器可以达到最大值 2 billion，重叠到 `- 2 billion`，并无限期地继续此循环。监控事务 ID 重叠的典型查询可能如下所示：

```
SELECT max(age(datfrozenxid)) FROM pg_database;
```

但是，在为 `datconnlimit` 引入 -2 值后，无效的数据库可能会扭曲此查询的结果。由于这些数据库无效，而不应作为定期维护检查的一部分，因此它们可能会导致误报，致使您认为 `age(datfrozenxid)` 高于实际值。

调整监控查询

为确保准确的监控，应调整监控查询来排除无效的数据库。请遵循这一建议的查询：

```
SELECT
    max(age(datfrozenxid))
FROM
```

```
    pg_database
WHERE
    datconnlimit <> -2;
```

此查询可确保在 age(datfrozenxid) 计算中只考虑有效的数据库，从而真实地反映整个 PostgreSQL 环境中的事务 ID 期限。

解决无效的数据库问题

当尝试连接到无效的数据库时，可能会遇到类似于以下内容的错误消息：

```
postgres=> \c db1
connection to server at "mydb.xxxxxxxxxx.us-west-2.rds.amazonaws.com" (xx.xx.xx.xxx),
 port xxxx failed: FATAL:  cannot connect to invalid database "db1"
HINT:  Use DROP DATABASE to drop invalid databases.
Previous connection kept
```

此外，如果 log_min_messages 参数设置为 DEBUG2 或更高，您可能会注意到以下日志条目，表明 autovacuum 进程正在跳过无效的数据库：

```
2024-07-30 05:59:00 UTC::@:[32000]:DEBUG:  autovacuum: skipping invalid database "db6"
2024-07-30 05:59:00 UTC::@:[32000]:DEBUG:  autovacuum: skipping invalid database "db1"
```

要解决此问题，请按照尝试连接期间提供的 HINT 进行操作。使用 RDS 主账户或具有 rds_superuser 角色的数据库账户连接到任何有效的数据库，然后删除无效的数据库。

```
SELECT
    'DROP DATABASE ' || quote_ident(datname) || ';'
FROM
    pg_database
WHERE
    datconnlimit = -2 \gexec
```

识别并解决 Aurora PostgreSQL 中的积极真空拦截器

在 PostgreSQL 中，清理对于确保数据库良好运行状况至关重要，因为它可以回收存储并防止出现事务 ID 重叠问题。但是，有时可能会阻止真空清理操作按预期运行，这样可能会导致性能下降、存储膨胀，甚至会因事务 ID 重叠而影响您的数据库实例的可用性。因此，识别和解决这些问题对于实现最佳数据库性能和可用性具有至关重要的意义。阅读了解 Amazon RDS for PostgreSQL 环境中的自动真空功能，以了解有关自动真空的更多信息。

`postgres_get_av_diag()` 函数可用于识别会阻止或延迟积极 vacuum 进度的问题。提供了建议，其中可能包括用于解决可识别问题的命令，或者在无法识别问题时提供进一步诊断的指导。当龄期超过 RDS 的 5 亿个事务 ID 的自适应自动真空阈值时，将报告积极真空拦截器。

事务 ID 的龄期是多少？

事务 ID 的 `age()` 函数会计算自数据库 (`pg_database.datfrozenxid`) 或表 (`pg_class.relfrozenxid`) 的最早解冻事务 ID 以来发生的事务数。此值表示自上次积极真空操作以来的数据库活动，并突出显示即将到来的真空进程可能出现的工作负载。

什么是积极真空？

积极真空操作会对表中的所有页面进行全面扫描，包括在常规真空期间通常跳过的页面。这种彻底的扫描旨在“冻结”接近其最大龄期的事务 ID，从而有效地防止出现被称为事务 ID 重叠的情况。

要使 `postgres_get_av_diag()` 报告拦截器，拦截器必须至少有 5 亿个事务。

主题

- 在 Aurora PostgreSQL 中安装 autovacuum 监控和诊断工具
- Aurora PostgreSQL 中的 postgres_get_av_diag() 函数
- 在 Aurora PostgreSQL 中解决可识别的真空拦截器
- 在 Aurora PostgreSQL 中解除无法识别的真空拦截器
- 解决 Aurora PostgreSQL 中的真空性能问题
- Aurora PostgreSQL 中的通知消息说明

在 Aurora PostgreSQL 中安装 autovacuum 监控和诊断工具

`postgres_get_av_diag()` 函数目前在以下 Aurora PostgreSQL 版本中可用：

- 17.4 及更高的 17 版本
- 16.7 及更高的 16 版本

- 15.11 及更高的 15 版本
- 14.16 及更高的 14 版本
- 13.19 及更高的 13 版本

要使用 postgres_get_av_diag()，请创建 rds_tools 扩展。

```
postgres=> CREATE EXTENSION rds_tools ;
CREATE EXTENSION
```

验证是否已安装扩展。

```
postgres=> \dx rds_tools
             List of installed extensions
   Name    | Version |  Schema   |                    Description
 ----------+---------+-----------
+-------------------------------------------------------------
 rds_tools |   1.9   | rds_tools | miscellaneous administrative functions for RDS
 PostgreSQL
 1 row
```

验证是否已创建函数。

```
postgres=> SELECT
    proname function_name,
    pronamespace::regnamespace function_schema,
    proowner::regrole function_owner
FROM
    pg_proc
WHERE
    proname = 'postgres_get_av_diag';
    function_name     | function_schema | function_owner
----------------------+-----------------+----------------
 postgres_get_av_diag | rds_tools       | rds_superuser
(1 row)
```

Aurora PostgreSQL 中的 postgres_get_av_diag() 函数

postgres_get_av_diag() 函数会检索有关 Aurora PostgreSQL 数据库中阻止或滞后的 autovacuum 进程的诊断信息。为了获得准确的结果，需要在具有最早事务 ID 的数据库中执行查询。有关使用具有最早事务 ID 的数据库的更多信息，请参阅未连接到具有最早事务 ID 龄期的数据库

```
SELECT
    blocker,
    DATABASE,
    blocker_identifier,
    wait_event,
    TO_CHAR(autovacuum_lagging_by, 'FM9,999,999,999') AS autovacuum_lagging_by,
    suggestion,
    suggested_action
FROM (
    SELECT
        *
    FROM
        rds_tools.postgres_get_av_diag ()
    ORDER BY
        autovacuum_lagging_by DESC) q;
```

`postgres_get_av_diag()` 函数将返回一个包含以下信息的表：

blocker

指定阻止真空的数据库活动的类别。

- 活动语句
- 事务中空闲
- 准备好的事务
- 逻辑复制槽
- 读取器实例
- 临时表

database

在适用和支持的情况下指定数据库的名称。这是其中活动正在进行并正在阻止或将阻止自动真空的数据库。这是您需要连接并执行操作的数据库。

blocker_identifier

指定正在阻止或将阻止自动真空的活动的标识符。该标识符可以是进程 ID 以及 SQL 语句、准备好的事务、只读副本的 IP 地址以及复制槽的名称（逻辑或物理）。

wait_event

指定阻止会话的等待事件，适用于以下拦截器：

- 活动语句

- 事务中空闲

autovacum_lagging_by

指定自动真空在每个类别的积压工作中滞后的事务数。

suggestion

指定解除拦截器的建议。这些说明包括活动所在的数据库的名称（如果适用）、会话的进程 ID（PID）（如果适用）以及要执行的操作。

suggested_action

建议需要执行哪些操作来解除拦截器。

在 Aurora PostgreSQL 中解决可识别的真空拦截器

自动真空会执行积极真空操作，并将事务 ID 的龄期降低到低于由 RDS 实例的 `autovacuum_freeze_max_age` 参数指定的阈值。您可以使用 Amazon CloudWatch 指标 `MaximumUsedTransactionIDs` 跟踪此龄期。

要查找 Amazon RDS 实例的 `autovacuum_freeze_max_age` 设置（默认为 2 亿个事务 ID），您可以使用以下查询：

```
SELECT
    TO_CHAR(setting::bigint, 'FM9,999,999,999') autovacuum_freeze_max_age
FROM
    pg_settings
WHERE
    name = 'autovacuum_freeze_max_age';
```

请注意，`postgres_get_av_diag()` 仅在龄期超过 Amazon RDS 的 5 亿个事务 ID 的自适应自动真空阈值时检查积极真空拦截器。要使 `postgres_get_av_diag()` 检测拦截器，拦截器必须至少有 5 亿个事务。

`postgres_get_av_diag()` 函数可识别以下类型的拦截器：

主题

- 活动语句
- 事务中空闲
- 准备好的事务

- 逻辑复制槽
- 读取器实例
- 临时表

活动语句

在 PostgreSQL 中，活动语句是数据库当前正在执行的 SQL 语句。这包括查询、事务或任何正在进行的操作。通过 pg_stat_activity 进行监控时，状态列指示具有相应 PID 的进程处于活动状态。

postgres_get_av_diag() 函数在识别系活动语句的语句时会显示类似于以下内容的输出。

```
blocker               | Active statement
database              | my_database
blocker_identifier    | SELECT pg_sleep(20000);
wait_event            | Timeout:PgSleep
autovacuum_lagging_by | 568,600,871
suggestion            | Connect to database "my_database", review carefully and you
 may consider terminating the process using suggested_action. For more information, see
 Working with PostgreSQL autovacuum in the Amazon RDS User Guide.
suggested_action      | {"SELECT pg_terminate_backend (29621);"}
```

建议采取的措施

按照 suggestion 列中的指导，用户可以连接到存在活动语句的数据库，并按照 suggested_action 列中的指定，建议仔细查看终止会话的选项。如果终止是安全的，则可以使用 pg_terminate_backend() 函数终止会话。此操作可由管理员（如 RDS 主账户）或具有所需 pg_terminate_backend() 权限的用户执行。

> **Warning**
>
> 已终止的会话将撤消 (ROLLBACK) 其所做的更改。根据您的要求，您可能需要重新运行相应语句。但是，建议只有在自动真空过程完成其积极真空操作之后才这样做。

事务中空闲

事务中空闲语句是指已打开显式事务（例如通过发出 BEGIN 语句）、执行了一些工作，并且现在正在等待客户端传递更多工作或通过发出 COMMIT、ROLLBACK 或 END（这将导致隐式 COMMIT）来发出事务结束信号的任何会话。

postgres_get_av_diag() 函数在将 idle in transaction 语句识别为拦截器时，显示类似于以下内容的输出。

```
blocker               | idle in transaction
database              | my_database
blocker_identifier    | INSERT INTO tt SELECT * FROM tt;
wait_event            | Client:ClientRead
autovacuum_lagging_by | 1,237,201,759
suggestion            | Connect to database "my_database", review carefully and you
 may consider terminating the process using suggested_action. For more information, see
 Working with PostgreSQL autovacuum in the Amazon RDS User Guide.
suggested_action      | {"SELECT pg_terminate_backend (28438);"}
```

建议采取的措施

如 suggestion 列中所示，您可以连接到存在事务中空闲会话的数据库，并使用 pg_terminate_backend() 函数终止会话。该用户可以是您的管理员（RDS 主账户）用户或具有 pg_terminate_backend() 权限的用户。

> **Warning**
>
> 已终止的会话将撤消 (ROLLBACK) 其所做的更改。根据您的要求，您可能需要重新运行相应语句。但是，建议只有在自动真空过程完成其积极真空操作之后才这样做。

准备好的事务

PostgreSQL 允许属于称为准备好的事务的两阶段提交策略一部分的事务。通过将 max_prepared_transactions 参数设置为非零值即可启用这些功能。准备好的事务旨在确保事务的持久性，即使在数据库崩溃、重启或客户端断开连接后仍可用。与定期事务一样，它们会被分配一个事务 ID，并且可能会影响自动真空。如果保持已准备状态，则自动真空将无法执行冻结，这样可能会导致事务 ID 重叠。

当事务被无限期地准备而没有被事务管理器解决时，它们将成为孤立的已准备事务。解决此问题的唯一方法是分别使用 COMMIT PREPARED 或 ROLLBACK PREPARED 命令提交或回滚事务。

> **Note**
>
> 请注意，在准备事务期间进行的备份在还原后仍将包含该事务。有关如何查找和关闭此类事务的信息，请参阅以下信息。

postgres_get_av_diag() 函数在识别出系已准备事务的拦截器时显示以下输出。

```
blocker               | Prepared transaction
database              | my_database
blocker_identifier    | myptx
wait_event            | Not applicable
autovacuum_lagging_by | 1,805,802,632
suggestion            | Connect to database "my_database" and consider either COMMIT
 or ROLLBACK the prepared transaction using suggested_action. For more information, see
 Working with PostgreSQL autovacuum in the Amazon RDS User Guide.
suggested_action      | {"COMMIT PREPARED 'myptx';",[OR],"ROLLBACK PREPARED 'myptx';"}
```

建议采取的措施

如建议列中所述，连接到已准备事务所在的数据库。根据 suggested_action 列，仔细查看是执行 COMMIT 还是 ROLLBACK，并执行适当的操作。

为了总体上监控已准备事务，PostgreSQL 提供了一个名为 pg_prepared_xacts 的目录视图。您可以使用以下查询来查找已准备事务。

```
SELECT
    gid,
    prepared,
    owner,
    database,
    transaction AS oldest_xmin
FROM
    pg_prepared_xacts
ORDER BY
    age(transaction) DESC;
```

逻辑复制槽

复制槽的用途是保存未使用的更改，直到它们被复制到目标服务器。有关更多信息，请参阅 PostgreSQL 的 Logical replication。

有两种类型的逻辑复制槽。

非活动逻辑复制槽

复制终止后，无法删除未使用的事务日志，且复制槽变为非活动状态。尽管订阅用户当前未使用非活动逻辑复制槽，但它仍保留在服务器上，从而保留 WAL 文件并防止删除旧的事务日志。这样会增加磁盘

使用量，特别是阻止自动真空清理内部目录表，因为系统必须保护 LSN 信息不被覆盖。如果不加以解决，可能会导致目录臃肿、性能下降和重叠真空风险增加，从而可能导致事务停机。

活动但速度较慢的逻辑复制槽

有时，由于逻辑复制的性能下降，目录中无效元组的删除会延迟。这种复制延迟会减慢 `catalog_xmin` 的更新速度，并可能导致目录臃肿和重叠真空。

`postgres_get_av_diag()` 函数在找到逻辑复制槽作为拦截器时，会显示类似于以下内容的输出。

```
blocker               | Logical replication slot
database              | my_database
blocker_identifier    | slot1
wait_event            | Not applicable
autovacuum_lagging_by | 1,940,103,068
suggestion            | Ensure replication is active and resolve any lag for the slot
 if active. If inactive, consider dropping it using the command in suggested_action.
 For more information, see Working with PostgreSQL autovacuum in the Amazon RDS User
 Guide.
suggested_action      | {"SELECT pg_drop_replication_slot('slot1') FROM
 pg_replication_slots WHERE active = 'f';"}
```

建议采取的措施

要解决此问题，请检查复制配置中是否存在可能终止应用进程的目标架构或数据问题。最常见的原因如下：

- 缺少列
- 数据类型不兼容
- 数据不匹配
- 缺少表

如果该问题与基础设施问题有关：

- 网络问题 – 如何解决 Amazon RDS 数据库处于不兼容网络状态的问题？。
- 数据库或数据库实例不可用的原因如下：
  - 副本实例用尽存储空间 – 有关添加存储的信息，请查看 Amazon RDS DB 实例存储空间不足。
  - 不兼容参数 – 有关如何能够解决该问题的更多信息，请查看如何修复一直处于不兼容参数状态的 Amazon RDS 数据库实例问题？。

如果您的实例位于 AWS 网络之外或 AWS EC2 上，请咨询您的管理员，了解如何解决可用性或基础设施相关问题。

删除非活动槽

> **Warning**
>
> 注意：在删除复制槽之前，请仔细确保其没有正在进行的复制、处于非活动状态且处于不可恢复状态。过早删除槽可能会中断复制或导致数据丢失。

确认不再需要复制槽后，将其删除以允许自动真空继续运行。条件 `active = 'f'` 可确保仅删除非活动槽。

```
SELECT pg_drop_replication_slot('slot1') WHERE active ='f'
```

读取器实例

在启用 hot_standby_feedback 设置之后，对于读取器实例上正在运行的查询仍然需要的死行，该设置会阻止写入器实例上的 autovacuum 删除这些死行。此行为是必需的，因为在读取器实例（同样适用于 Aurora Global Database 中的读取器实例）上运行的查询要求这些行在写入器实例上保持可用，从而防止出现查询冲突和查询取消。

> **Note**
>
> `hot_standby_feedback` 在 Aurora PostgreSQL 中默认启用且不可修改。

`postgres_get_av_diag()` 函数在找到以物理复制槽作为拦截器的只读副本时，会显示类似于以下内容的输出。

```
blocker               | Oldest query running on aurora reader
database              | Not applicable
blocker_identifier    | my-aurora-reader-2
wait_event            | Not applicable
autovacuum_lagging_by | 540,122,859
suggestion            | Run the following query on the reader "my-aurora-reader-2" to
 find the long running query:
```

```
                      | SELECT * FROM pg_catalog.pg_stat_activity WHERE
 backend_xmin::text::bigint = 523476310;

                      | Review carefully and you may consider terminating the query on
 reader using suggested_action.
suggested_action      | {"SELECT pg_terminate_backend(pid) FROM
 pg_catalog.pg_stat_activity WHERE backend_xmin::text::bigint = 523476310;","

                      | [OR]


                      | ","Delete the reader if not needed"}
```

按照 suggested_action 列中的建议，仔细查看用于解除阻止自动真空的这些选项。

- 终止查询 – 按照建议列中的指导，您可以连接到只读副本，如 suggested_action 列中指定，建议仔细查看终止会话的选项。如果终止被视为是安全的，则可以使用 pg_terminate_backend( ) 函数终止会话。此操作可由管理员（如 RDS 主账户）或具有所需 pg_terminate_backend() 权限的用户执行。

  您可以在只读副本上运行以下 SQL 命令来终止用于阻止主数据库上的真空清理旧行的查询。在函数的输出中报告 backend_xmin 的值：

  ```
  SELECT
      pg_terminate_backend(pid)
  FROM
      pg_catalog.pg_stat_activity
  WHERE
      backend_xmin::text::bigint = backend_xmin;
  ```

- 如果不需要读取器实例，请将其删除 – 如果不再需要读取器实例，则可以将其删除。这将消除相关的复制开销，并允许主数据库在不受实例阻碍的情况下回收事务日志。

临时表

使用 TEMPORARY 关键字创建的临时表驻留在临时架构中，例如 pg_temp_xxx，并且只能由创建它们的会话访问。会话结束时会删除临时表。但是，这些表对于 PostgreSQL 的自动真空流程是不可见的，必须由创建它们的会话手动清理。尝试从另一个会话中清空临时表是无效的。

在特殊情况下，存在一个临时表，但没有活动会话拥有该表。如果所属会话由于致命崩溃、网络问题或类似事件而意外结束，则可能无法清理临时表，从而会将其留作为“孤立”表。当 PostgreSQL 自动真空进程检测到孤立的临时表时，它会记录以下消息：

```
LOG: autovacuum: found orphan temp table \"%s\".\"%s\" in database \"%s\"
```

postgres_get_av_diag() 函数在将临时表识别为拦截器时，会显示类似于以下内容的输出。要使该函数正确显示与临时表相关的输出，需要在存在这些表的同一数据库中执行该函数。

```
blocker               | Temporary table
database              | my_database
blocker_identifier    | pg_temp_14.ttemp
wait_event            | Not applicable
autovacuum_lagging_by | 1,805,802,632
suggestion            | Connect to database "my_database". Review carefully, you
 may consider dropping temporary table using command in suggested_action. For more
 information, see Working with PostgreSQL autovacuum in the Amazon RDS User Guide.
suggested_action      | {"DROP TABLE ttemp;"}
```

建议采取的措施

按照输出的 suggestion 列中提供的说明识别并删除阻止自动真空运行的临时表。使用以下命令删除由 postgres_get_av_diag() 报告的临时表。根据 postgres_get_av_diag() 函数提供的输出替换表名称。

```
DROP TABLE my_temp_schema.my_temp_table;
```

以下查询可用于识别临时表：

```
SELECT
    oid,
    relname,
    relnamespace::regnamespace,
    age(relfrozenxid)
FROM
    pg_class
WHERE
relpersistence = 't'
ORDER BY
    age(relfrozenxid) DESC;
```

在 Aurora PostgreSQL 中解除无法识别的真空拦截器

本节探讨了可能阻碍清理取得进展的其他原因。postgres_get_av_diag() 函数目前无法直接识别这些问题。

主题

- 索引不一致
- 事务速率异常高

索引不一致

逻辑上不一致的索引可能会阻碍自动真空取得进展。在索引的真空阶段或 SQL 语句访问索引时，将记录以下错误或类似错误。

```
ERROR: right sibling's left-link doesn't match:block 5 links to 10 instead of expected
 2 in index ix_name
```

```
ERROR: failed to re-find parent key in index "XXXXXXXXXX" for deletion target page XXX
CONTEXT:  while vacuuming index index_name of relation schema.table
```

指南

在手动 VACUUM FREEZE 上使用 INDEX_CLEANUP 重建索引或跳过索引。

- 使用 CONCURRENTLY 选项 – 在 PostgreSQL 版本 12 之前，重建索引需要独占的表锁定，从而限制对表的访问。在 PostgreSQL 版本 12 及更高版本中，CONCURRENTLY 选项允许行级别锁定，从而显著提高表的可用性。命令如下：

  ```
  REINDEX INDEX ix_name CONCURRENTLY;
  ```

  虽然 CONCURRENTLY 的破坏性较小，但在繁忙的表中其速度可能会变慢。如果可能，可以考虑在流量较低的时段构建索引。有关更多信息，请参阅 PostgreSQL 文档中的 REINDEX。
- 使用 INDEX_CLEANUP FALSE 选项 – 如果索引很大并且估计需要大量时间才能完成，则您可以通过在排除索引的同时执行手动 VACUUM FREEZE 来解除阻止 autovacuum。PostgreSQL 版本 12 及更高版本中提供了此功能。

  绕过索引将允许您跳过索引不一致的真空过程并缓解重叠问题。但是，这样并不能解决潜在的无效页面问题。要完全应对和解决无效页面问题，您仍然需要重建索引。

事务速率异常高

在 PostgreSQL 中，事务速率高会显著影响自动真空的性能，从而导致无效元组的清理速度变慢，并增加事务 ID 重叠的风险。您可以通过衡量两个时间段之间的 `max(age(datfrozenxid))` 差异（通常为每秒）来监控事务速率。此外，您还可以使用 RDS 性能详情中的以下计数器指标来衡量事务速率（xact_commit 与 xact_rollback 之和），即事务总数。

计数器	类型	单位	指标
xact_commit	事务	每秒提交数	db.Transactions.xact_commit
xact_rollback	事务	每秒回滚数	db.Transactions.xact_rollback

快速增加表示高事务负载，这可能会使自动真空不堪重负，从而导致膨胀、锁争用和潜在的性能问题。这样可能会以多种方式对自动真空进程产生负面影响：

- 表活动：正在经历清理操作的特定表可能会遇到大量事务，从而导致延迟。
- 系统资源：整个系统可能会过载，使自动真空难以访问必要的资源以便高效运行。

请考虑以下策略，以允许自动真空更有效地运行并跟上其任务：

1. 如果可能，请降低事务速率。请考虑在可行的情况下对类似的事务进行批处理或分组。
2. 在非高峰时段，每晚、每周或每两周通过手动 `VACUUM FREEZE` 操作来定位频繁更新的表。
3. 请考虑扩展您的实例类以分配更多系统资源来处理高事务量和自动真空操作。

解决 Aurora PostgreSQL 中的真空性能问题

本节讨论经常导致真空性能变慢的因素以及如何解决这些问题。

主题

- 清理大型索引
- 需要清理的表或数据库太多
- 正在运行积极真空（以防止重叠）

清理大型索引

VACUUM 按以下阶段顺序运行：初始化、堆扫描、索引和堆真空清理、索引清理、堆截断和最终清理。在堆扫描期间，该过程修剪页面、整理碎片并冻结它们。完成堆扫描后，VACUUM 将清理索引，将空页返回到操作系统，并执行最终的清理任务，例如对空闲空间映射表执行真空清理和更新统计信息。

对索引执行真空清理操作时，如果 `maintenance_work_mem`（或 `autovacuum_work_mem`）不足以处理索引，则可能需要多次扫描。在 PostgreSQL 16 及更早版本中，用于存储无效元组 ID 存在 1 GB 内存限制，对于大型索引通常需要多次扫描。PostgreSQL 17 引入了 `TidStore`，这会动态分配内存，而不是使用单一分配数组。此功能消除了 1 GB 限制，可以更高效地利用内存，并减少了每次索引操作中进行多次索引扫描的需求。

在 PostgreSQL 17 中，对于大型索引，如果可用内存无法一次性容纳整个索引的处理需求，那么仍可能需要多次扫描。通常，更大的索引往往包含更多需要多次扫描的无效元组。

检测缓慢的 vacuum 操作

`postgres_get_av_diag()` 函数可以检测 vacuum 操作何时因内存不足而运行缓慢。有关此函数的更多信息，请参阅在 Aurora PostgreSQL 中安装 autovacuum 监控和诊断工具。

当可用内存不足以一次性完成索引 vacuum 时，`postgres_get_av_diag()` 函数会发出以下通知。

`rds_tools` 1.9

```
NOTICE: Your database is currently running aggressive vacuum to prevent wraparound and
 it might be slow.
```

```
NOTICE: The current setting of autovacuum_work_mem is XX might not be sufficient.
 Consider increasing the setting to XXX, and if necessary, scaling up the RDS instance
 class for more
       memory. The suggested value is an estimate based on the current number of dead
 tuples for the table being vacuumed, which might not fully reflect the latest state.
 Additionally, review the possibility of manual
       vacuum with exclusion of indexes using (VACUUM (INDEX_CLEANUP FALSE, VERBOSE
 TRUE) table_name;). For more information, see

       Working with PostgreSQL autovacuum in the Amazon Aurora User Guide.
```

> **Note**
>
> postgres_get_av_diag() 函数依靠 pg_stat_all_tables.n_dead_tup 来估计索引清理所需的内存量。

当 postgres_get_av_diag() 函数识别出由于 autovacuum_work_mem 不足而需要多次索引扫描的慢速 vacuum 操作时，它将生成以下消息：

```
NOTICE: Your vacuum is performing multiple index scans due to insufficient
 autovacuum_work_mem:XXX for index vacuuming.
       For more information, see Working with PostgreSQL autovacuum in the Amazon
 Amazon RDS User Guide.
```

指南

您可以使用手动 VACUUM FREEZE 应用以下解决方法来加快表的冻结速度。

增加用于清理的内存

正如 postgres_get_av_diag() 函数所建议的那样，建议增加 autovacuum_work_mem 参数以解决实例级别的潜在内存限制问题。虽然 autovacuum_work_mem 是一个动态参数，但请务必注意，要使新的内存设置生效，自动真空进程守护程序需要重新启动其工作线程。要实现这一点，请执行以下操作：

1. 确认新设置已落实到位。
2. 终止当前运行自动真空的进程。

这种方法可确保将调整后的内存分配应用于新的自动真空操作。

为了获得更直接的结果，请考虑在会话中手动执行 VACUUM FREEZE 操作，并增加 maintenance_work_mem 设置：

```
SET maintenance_work_mem TO '1GB';
VACUUM FREEZE VERBOSE table_name;
```

如果您使用的是 Amazon RDS，并且发现需要额外的内存来支持更高的 maintenance_work_mem 或 autovacuum_work_mem 值，请考虑升级到具有更多内存的实例类。这样可以提供必要的资源来增强手动和自动清理操作，从而提高整体真空和数据库性能。

禁用 INDEX_CLEANUP

PostgreSQL 版本 12 及更高版本中的手动 VACUUM 允许跳过索引清理阶段，而 PostgreSQL 版本 14 及更高版本中的紧急自动真空会根据 vacuum_failsafe_age 参数自动执行此操作。

> **Warning**
>
> 跳过索引清理会导致索引膨胀并对查询性能产生负面影响。为了缓解这种情况，可以考虑在维护时段内对受影响的索引重建索引或清理受影响的索引。

有关处理大型索引的更多指导，请参阅使用大型索引管理 autovacuum 中的文档。

并行索引清理

从 PostgreSQL 13 开始，默认情况下，可以使用手动 VACUUM 并行对索引进行清理，并为每个索引分配一个真空工件进程。但是，要使 PostgreSQL 确定真空操作是否符合并行执行的条件，必须满足特定标准：

- 必须至少有两个索引。
- max_parallel_maintenance_workers 参数应至少设置为 2。
- 索引大小必须超过 min_parallel_index_scan_size 限制，默认为 512 KB。

您可以根据 Amazon RDS 实例上可用的 vCPU 数量和表上的索引数量来调整 max_parallel_maintenance_workers 设置，以优化清理周转时间。

有关更多信息，请参阅 Parallel vacuuming in Amazon RDS for PostgreSQL and Amazon Aurora PostgreSQL。

需要清理的表或数据库太多

如 PostgreSQL 的自动真空进程守护程序文档中所述，自动真空进程守护程序通过多个进程运行。这包括一个持久性自动真空启动程序，负责为系统中的每个数据库启动自动真空工件进程。该启动程序安排这些工件大约每个数据库每 autovacuum_naptime 秒启动一次。

对于“N”个数据库，大约每 [autovacuum_naptime/N 秒] 就会有一个新工件开始工作。但是，并发工件的总数受 autovacuum_max_workers 设置的限制。如果需要清理的数据库或表的数量超过此限制，则一旦有工件可用，就会立即处理下一个数据库或表。

当许多大型表或数据库需要同时执行清理操作时，所有可用的自动真空工件可能会长时间被占用，从而会延迟其他表和数据库的维护。在事务速率较高的环境中，此瓶颈可能会迅速升级，并可能导致 Amazon RDS 实例中出现重叠真空问题。

当 postgres_get_av_diag() 检测到大量表或数据库时，它会提供以下建议：

```
NOTICE: Your database is currently running aggressive vacuum to prevent wraparound and
 it might be slow.
```

```
NOTICE: The current setting of autovacuum_max_workers:3 might not be sufficient.
 Consider increasing the setting and, if necessary, consider scaling up the Amazon RDS
 instance class for more workers.
```

指南

增加 autovacuum_max_workers

为了加快清理速度，我们建议调整 autovacuum_max_workers 参数以允许更多并发自动真空工件。如果性能瓶颈仍然存在，请考虑将 Amazon RDS 实例扩展到具有更多 vCPU 的类，这样可以进一步提高并行处理能力。

正在运行积极真空（以防止重叠）

PostgreSQL 中数据库（MaximumUsedTransactionIDs）的龄期仅会在成功完成积极真空（以防止重叠）时减少。在此真空结束之前，龄期将继续增加，具体取决于事务速率。

postgres_get_av_diag() 函数在检测到积极真空时会生成以下 NOTICE。但是，只有在真空处于活动状态至少两分钟后，它才会触发此输出。

```
NOTICE: Your database is currently running aggressive vacuum to prevent wraparound,
 monitor autovacuum performance.
```

有关积极真空的更多信息，请参阅当积极真空已经在运行时。

您可以使用以下查询验证是否正在进行主动清理：

```
SELECT
    a.xact_start AS start_time,
    v.datname "database",
    a.query,
```

```
    a.wait_event,
    v.pid,
    v.phase,
    v.relid::regclass,
    pg_size_pretty(pg_relation_size(v.relid)) AS heap_size,
    (
        SELECT
            string_agg(pg_size_pretty(pg_relation_size(i.indexrelid)) || ':' ||
 i.indexrelid::regclass || chr(10), ', ')
        FROM
            pg_index i
        WHERE
            i.indrelid = v.relid
    ) AS index_sizes,
    trunc(v.heap_blks_scanned * 100 / NULLIF(v.heap_blks_total, 0)) AS step1_scan_pct,
    v.index_vacuum_count || '/' || (
        SELECT
            count(*)
        FROM
            pg_index i
        WHERE
            i.indrelid = v.relid
    ) AS step2_vacuum_indexes,
    trunc(v.heap_blks_vacuumed * 100 / NULLIF(v.heap_blks_total, 0)) AS
 step3_vacuum_pct,
    age(CURRENT_TIMESTAMP, a.xact_start) AS total_time_spent_sofar
FROM
    pg_stat_activity a
    INNER JOIN pg_stat_progress_vacuum v ON v.pid = a.pid;
```

您可以通过检查输出中的查询列来确定这是否为积极真空（以防止重叠）。“防止重叠”一词表示这是一个积极真空。

```
query                 | autovacuum: VACUUM public.t3 (to prevent wraparound)
```

例如，假设您有一个事务龄期为 10 亿的拦截器，还有一个需要积极真空的表，以防止在相同的事务龄期出现重叠。此外，在事务龄期为 7.5 亿时，还有另一个拦截器。在事务龄期为 10 亿时清除拦截器后，事务龄期不会立即下降到 7.5 亿。它将保持较高的状态，直到需要积极真空或任何龄期超过 7.5 亿的事务完成为止。在此期间，PostgreSQL 集群的事务龄期将持续提高。真空过程完成后，事务龄期将降至 7.5 亿，但在进一步的清理完成之前，事务龄期将再次开始增加。只要这些条件仍然存在，此循环就会继续，直到事务龄期最终下降到为您的 Amazon RDS 实例配置的级别（由 `autovacuum_freeze_max_age` 指定）。

Aurora PostgreSQL 中的通知消息说明

postgres_get_av_diag() 函数提供以下通知消息：

当龄期尚未达到监控阈值时

默认情况下，postgres_get_av_diag() 识别拦截器的监控阈值为 5 亿个事务。如果 postgres_get_av_diag() 生成以下通知，则表示事务龄期尚未达到此阈值。

```
NOTICE: postgres_get_av_diag() checks for blockers that prevent aggressive vacuums
 only, it does so only after exceeding dvb_threshold which is 500,000,000 and age of
 this PostgreSQL cluster is currently at 2.
```

未连接到具有最早事务 ID 龄期的数据库

当连接到具有最早事务 ID 龄期的数据库时，postgres_get_av_diag() 函数可提供最准确的输出。postgres_get_av_diag() 报告的具有最早事务 ID 龄期的数据库将不同于您的案例中的“my_database”。如果您未连接到正确的数据库，则会生成以下通知：

```
NOTICE: You are not connected to the database with the age of oldest transaction
 ID. Connect to my_database database and run postgres_get_av_diag() for accurate
 reporting.
```

出于以下原因，连接到具有最早事务龄期的数据库非常重要：

- 识别临时表拦截器：由于临时表的元数据特定于每个数据库，因此它们通常位于在其中创建临时表的数据库中。但是，如果临时表恰好是最重要的拦截器，并且位于具有最早事务的数据库中，则可能会产生误导。连接到正确的数据库可确保准确识别临时表拦截器。
- 诊断缓慢真空：索引元数据和表计数信息是特定于数据库的，是诊断缓慢真空问题所必需的。

按龄期计算具有最早事务的数据库位于 rdsadmin 或 template0 数据库中

在某些情况下，rdsadmin 或 template0 数据库可能会被标识为具有最早事务 ID 龄期的数据库。如果发生这种情况，postgres_get_av_diag() 将发出以下通知：

```
NOTICE: The database with the age of oldest transaction ID is rdsadmin or template0,
 reach out to support if the reported blocker is in rdsadmin or template0.
```

验证列出的拦截器是否不是来自这两个数据库中的任何一个。如果据报告 rdsadmin 或 template0 中存在拦截器，请联系支持部门，因为这些数据库对用户不可访问，需要干预。

rdsadmin 或 template0 数据库包含最重要拦截器的可能性极小。

当积极真空已经在运行时

postgres_get_av_diag() 函数旨在报告何时正在运行积极真空进程，但它仅在真空处于活动状态至少 1 分钟后触发此输出。这种有意的延迟有助于减少误报的可能性。通过等待，该函数可确保仅报告有效、重要的真空，从而可更准确、更可靠地监控真空活动。

postgres_get_av_diag() 函数在检测到一个或多个正在进行的积极真空时会生成以下通知。

```
NOTICE: Your database is currently running aggressive vacuum to prevent wraparound,
 monitor autovacuum performance.
```

如通知中所述，请继续监控真空的性能。有关积极真空的更多信息，请参阅正在运行积极真空（以防止重叠）

当自动真空关闭时

如果数据库实例上禁用了自动真空，则 postgres_get_av_diag() 函数会生成以下通知：

```
NOTICE: Autovacuum is OFF, we strongly recommend to enable it, no restart is
 necessary.
```

Autovacuum 是 Aurora PostgreSQL 数据库实例的一项关键功能，可确保数据库顺畅运行。它会自动删除旧的行版本，回收存储空间，并防止表膨胀，从而帮助保持表和索引的高效性，以实现最佳性能。此外，它还可以防止发生事务 ID 重叠，这可能会停止 Amazon RDS 实例上的事务。禁用自动真空会导致数据库性能和稳定性的长期下降。建议您始终将其保持开启状态。有关更多信息，请参阅 Understanding autovacuum in Aurora PostgreSQL environments。

> **Note**
>
> 关闭自动真空并不会阻止积极真空。一旦您的表达到 autovacuum_freeze_max_age 阈值，仍会发生这种情况。

剩余的事务数严重不足

postgres_get_av_diag() 函数会在即将发生重叠真空时生成以下通知。当您的 Amazon RDS 实例距离可能拒绝新事务还有 1 亿个事务时，将发出此通知。

```
WARNING: Number of transactions remaining is critically low, resolve issues with
 autovacuum or perform manual VACUUM FREEZE before your instance stops accepting
 transactions.
```

需要立即采取行动以免发生数据库停机。您应该密切监控真空操作，并考虑在受影响的数据库上手动启动 VACUUM FREEZE 以防止事务失败。

使用 Babelfish for Aurora PostgreSQL

Babelfish for Aurora PostgreSQL 扩展了您的 Aurora PostgreSQL 数据库集群，以能够接受来自 SQL Server 客户端的数据库连接。使用 Babelfish，最初为 SQL Server 构建的应用程序可以直接与 Aurora PostgreSQL 协作，与传统迁移相比，代码更改很少，而且无需更改数据库驱动程序。有关迁移的更多信息，请参阅 将 SQL Server 数据库迁移到 Babelfish for Aurora PostgreSQL。

Babelfish 为 Aurora PostgreSQL 数据库集群提供了一个额外的端点，使其能够了解 SQL Server 线路级协议和常用的 SQL Server 语句。使用表格式数据流 (TDS) 线路协议的客户端应用程序可以通过本机方式连接到 Aurora PostgreSQL 上的 TDS 侦听器端口。要了解有关 TDS 的更多信息，请参阅 Microsoft 网站上的 [MS-TDS]：表格式数据流协议。

> **Note**
>
> Babelfish for Aurora PostgreSQL 支持 TDS 版本 7.1 到 7.4。

Babelfish 还可以使用 PostgreSQL 连接访问数据。原定设置情况下，Babelfish 支持的两种 SQL 方言都可以通过以下端口处的本机线路协议获得：

- SQL Server 方言 (T-SQL)，客户端连接到端口 1433。
- PostgreSQL 方言 (PL/pgSQL)，客户端连接到端口 5432。

Babelfish 运行 Transact-SQL (T-SQL) 语言，但有一些区别。有关更多信息，请参阅 适用于 Aurora PostgreSQL 的 Babelfish 与 SQL Server 之间的区别。

在下列部分中，您可找到有关设置和使用 Babelfish for Aurora PostgreSQL 数据库集群的信息。

主题

- Babelfish 限制
- 了解 Babelfish 架构和配置
- 创建 Babelfish for Aurora PostgreSQL 数据库集群
- 将 SQL Server 数据库迁移到 Babelfish for Aurora PostgreSQL
- 适用于 Aurora PostgreSQL 的 Babelfish 的数据库身份验证
- 连接到 Babelfish 数据库集群
- 使用 Babelfish

- Babelfish 问题排查
- 关闭 Babelfish
- 管理适用于 Aurora PostgreSQL 的 Babelfish 版本更新
- Babelfish for Aurora PostgreSQL 参考

Babelfish 限制

以下限制目前可用于适用于 Aurora PostgreSQL 的 Babelfish：

- 升级时，您可能会注意到同一架构中对象所有权的差异。升级前的对象归当前用户拥有，而升级后的对象则归架构所有者拥有，架构所有者可能是不同的用户。为了解决这个问题，我们引入了 `sys.generate_alter_ownership_statements()` 函数。

  要修复对象所有权不匹配的问题，请使用 PostgreSQL 端点连接到集群，运行 sys.generate_alter_ownership_statements() 函数，然后执行生成的 SQL 语句。

  请考虑以下与对象所有权更改有关的重要限制：

  - 通过 PostgreSQL 端点获得 CREATE 权限的用户无法通过 TDS 端点创建对象。建议不要通过 PostgreSQL 端点更改对 T-SQL 对象的权限，因为这可能会导致错误的 T-SQL 行为。
  - 对象访问权限可能会发生变化。例如，如果“sch_own”拥有的架构包含在升级前由“dbo”拥有的对象，则用户执行诸如 SELECT 或 INSERT 之类的操作的能力可能在升级前对象（由“dbo”拥有）和升级后对象（由“sch_own”拥有）之间有所不同。
- Babelfish 目前不支持以下 Aurora 功能：
  - AWS Identity and Access Management
  - 数据库活动流（DAS）
  - RDS 数据 API 与 Aurora PostgreSQL Serverless v2 和预调配
  - 适用于 RDS for SQL Server 的 RDS 代理
  - 加盐质询响应身份验证机制（SCRAM）
  - 查询编辑器
- Babelfish 不提供以下客户端驱动程序 API 支持：
  - 不支持连接属性与 Microsoft 分布式事务处理协调器（MSDTC）相关的 API 请求。其中包括由 SQL Server JDBC 驱动程序中的 SQLServerXAResource 类执行的 XA 调用。
- Babelfish 目前不支持以下 Aurora PostgreSQL 扩展：
  - `bloom`
  - `btree_gin`
  - `btree_gist`
  - `citext`
  - `cube`
  - `hstore`

- hypopg
- 使用 pglogical 的逻辑复制
- ltree
- pgcrypto
- 使用 apg_plan_mgmt 的查询计划管理

要了解有关 PostgreSQL 扩展的更多信息，请参阅使用扩展和外部数据包装器。

- 不支持作为 Microsoft JDBC 驱动程序的替代方案而设计的开源 jTDS 驱动程序。

了解 Babelfish 架构和配置

您管理运行 Babelfish 的 Aurora PostgreSQL 兼容版数据库集群，就像管理任何 Aurora 数据库集群一样。也就是说，您可以从 Aurora 数据库集群提供的可扩展性、具有失效转移支持的高可用性以及内置复制中受益。要了解有关这些功能的更多信息，请参阅管理 Aurora 数据库集群的性能和扩展、Amazon Aurora 的高可用性和使用 Amazon Aurora 进行复制。您还可以访问许多其他 AWS 工具和实用程序，包括以下各项：

- Amazon CloudWatch 是一项监控和可观察性服务，为您提供数据和可操作的洞察。有关更多信息，请参阅 使用 Amazon CloudWatch 监控 Amazon Aurora 指标。
- 性能详情是一项数据库性能优化和监控功能，可帮助您快速评估数据库的负载。要了解更多信息，请参阅 在 Amazon Aurora 上使用性能详情监控数据库负载。
- Aurora 全局数据库跨越多个 AWS 区域，可实现低延迟的全局读取，并可从可能影响整个 AWS 区域的罕见停机事件中快速恢复。有关更多信息，请参阅 使用 Amazon Aurora Global Database。
- 自动软件修补使用可用的最新安全和功能补丁来保持数据库的最新状态。
- Amazon RDS 事件通过电子邮件或 SMS 消息通知您重要的数据库事件，例如自动失效转移。有关更多信息，请参阅 监控 Amazon Aurora 事件。

接下来，您可以了解 Babelfish 架构以及 Babelfish 如何处理您迁移的 SQL Server 数据库。创建 Babelfish 数据库集群时，您需要事先就单个数据库或多个数据库、排序规则和其他详细信息做出一些决策。

主题

- Babelfish 架构
- Babelfish 的数据库集群参数组设置

- 了解适用于 Aurora PostgreSQL 的 Babelfish 中的排序规则
- 使用转义孵化管理 Babelfish 错误处理

Babelfish 架构

当您在打开 Babelfish 的情况下创建 Aurora PostgreSQL 集群时，Aurora 会为该集群预置名为 `babelfish_db` 的 PostgreSQL 数据库。此数据库是所有迁移的 SQL Server 对象和结构所在的位置。

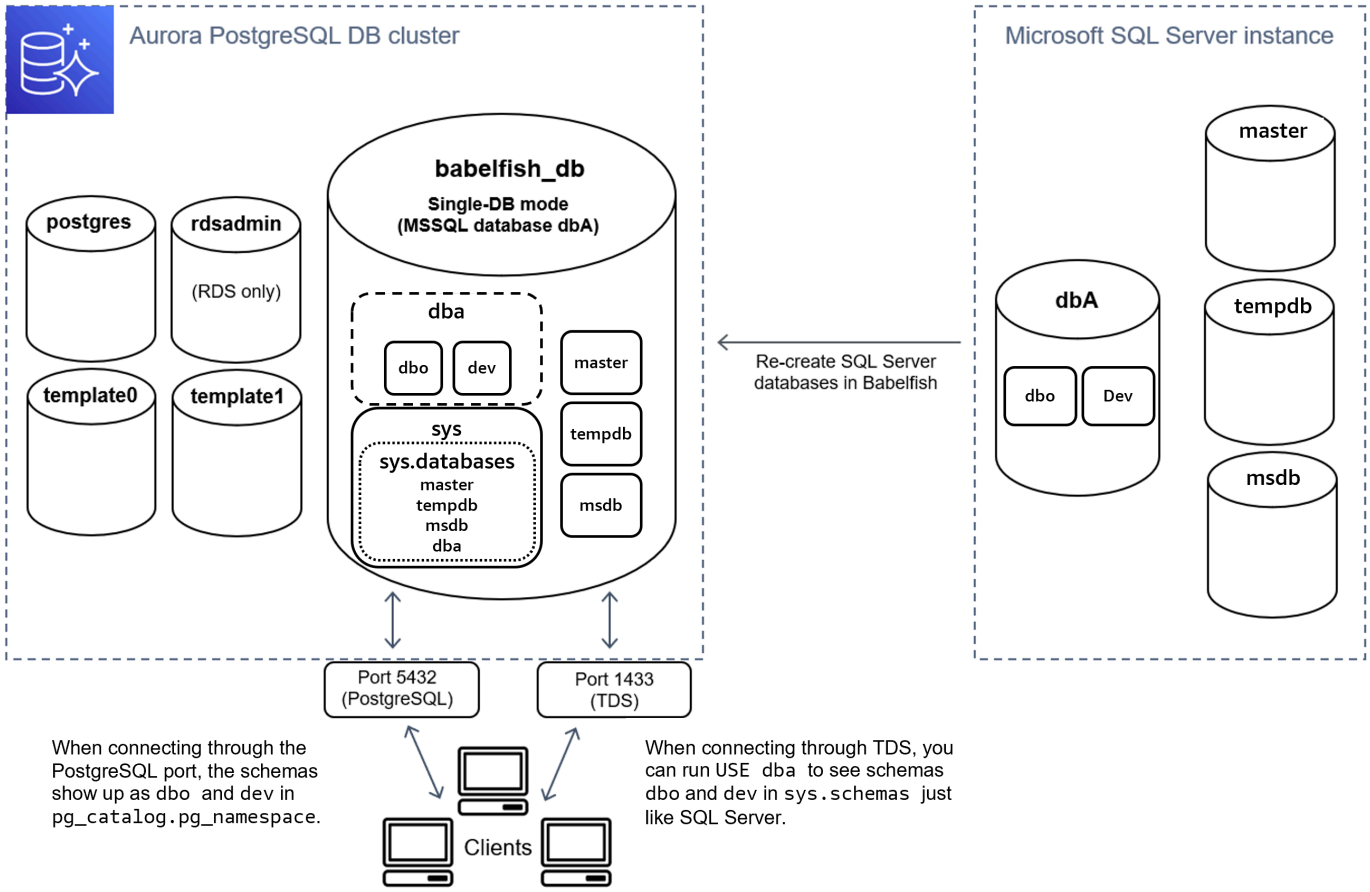
> **Note**
>
> 在 Aurora PostgreSQL 集群中，系统会为 Babelfish 预留 `babelfish_db` 数据库名称。在 Babelfish 数据库集群上自行创建“babelfish_db”数据库会阻止 Aurora 成功预调配 Babelfish。

当您连接到 TDS 端口时，会话将被放置在 `babelfish_db` 数据库中。从 T-SQL 来看，该结构看起来类似于连接到 SQL Server 实例。您可以看到 `master`、`msdb` 和 `tempdb` 数据库以及 `sys.databases` 目录。您可以创建其他用户数据库并使用 USE 语句在数据库之间切换。当您创建 SQL Server 用户数据库时，该数据库将被展平到 `babelfish_db` PostgreSQL 数据库中。数据库保留的跨数据库语法和语义等于或类似于 SQL Server 提供的语法和语义。

将 Babelfish 与单个数据库或多个数据库结合使用

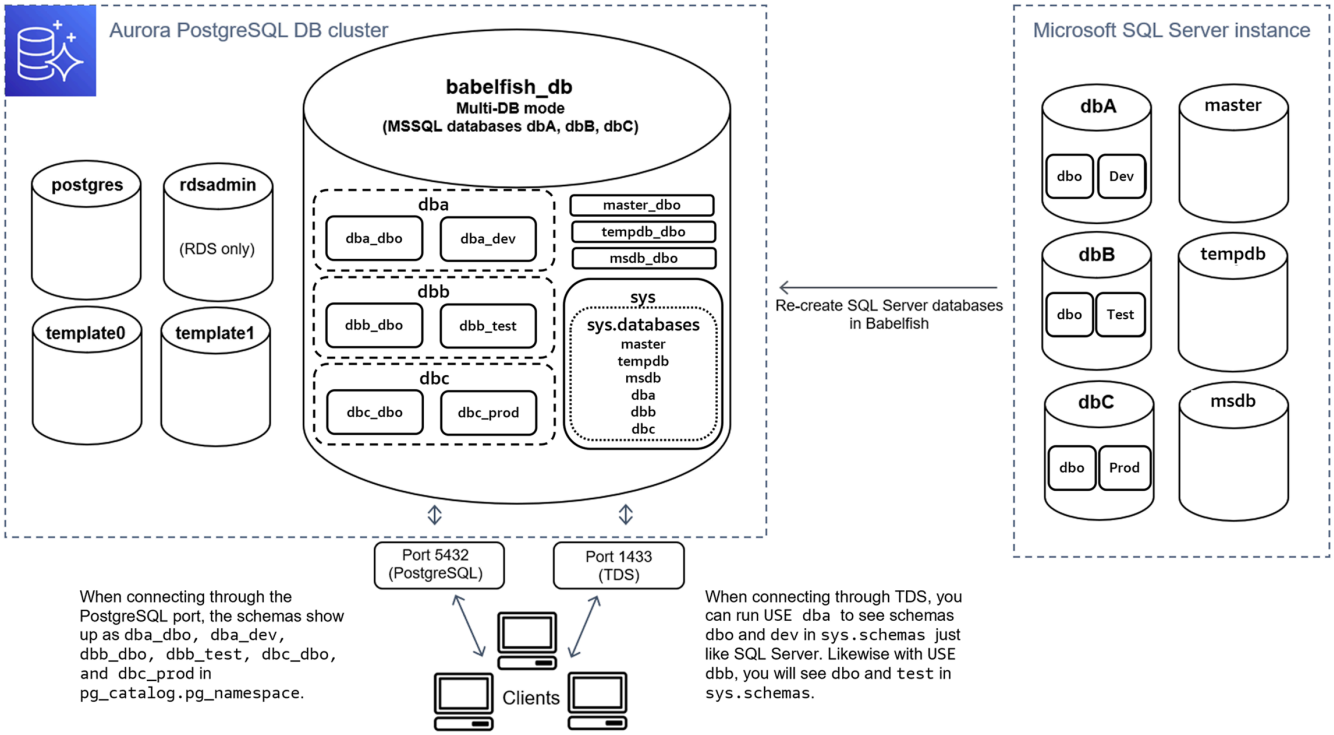
当您创建 Aurora PostgreSQL 集群以与 Babelfish 一起使用时，您可以选择单独使用单个 SQL Server 数据库或者将多个 SQL Server 数据库一起使用。您的选择会影响 `babelfish_db` 数据库内的 SQL Server 架构的名称显示在 Aurora PostgreSQL 中的方式。迁移模式存储在 `migration_mode` 参数中。创建集群后不得更改此参数，因为您可能会丢失对所有先前创建的 SQL 对象的访问权限。

在单数据库模式下，SQL Server 数据库的模式名称与 PostgreSQL 的 `babelfish_db` 数据库中保持一致。如果您选择仅迁移单个数据库，则可以在 PostgreSQL 中引用迁移的用户数据库的模式名称，其名称与 SQL Server 中使用的名称相同。例如，`dbo` 和 `smith` 架构驻留在 `dbA` 数据库内。



通过 TDS 进行连接时，您可以运行 USE dba 以查看来自 T-SQL 的架构 dbo 和 dev，就像在 SQL Server 中一样。不变的架构名称也可以从 PostgreSQL 中看到。

在多数据库模式下，当从 PostgreSQL 中访问时，用户数据库的模式名称变为 dbname_schemaname。从 T-SQL 中访问时，模式名称保持不变。



如图中所示，当通过 TDS 端口连接并使用 T-SQL 时，多数据库模式和单数据库模式与 SQL Server 相同。例如，USE dbA 列出架构 dbo 和 dev，就像在 SQL Server 中一样。映射的架构名称，例如 dba_dbo 和 dba_dev，在 PostgreSQL 中可见。

每个数据库仍包含您的架构。每个数据库的名称都加在 SQL Server 架构的名称之前，使用下划线作为分隔符，例如：

- dba 包含 dba_dbo 和 dba_dev。
- dbb 包含 dbb_dbo 和 dbb_test。
- dbc 包含 dbc_dbo 和 dbc_prod。

在 babelfish_db 数据库内，T-SQL 用户仍然需要运行 USE dbname 以更改数据库上下文，因此外观和感觉与 SQL Server 相似。

选择迁移模式

每种迁移模式都有优点和缺点。根据您拥有的用户数据库数量和迁移计划选择迁移模式。创建用于 Babelfish 的集群后，您不得更改迁移模式，因为您可能无法访问之前创建的所有 SQL 对象。选择迁移模式时，请考虑用户数据库和客户端的要求。

当您创建用于 Babelfish 的集群时，Aurora PostgreSQL 会创建系统数据库 `master` 和 `tempdb`。如果您在系统数据库中创建或修改了对象（`master` 或 `tempdb`），请确保在新集群中重新创建这些对象。与 SQL Server 不同，Babelfish 不会在集群重启后重新初始化 `tempdb`。

在以下情况下使用单个数据库迁移模式：

- 如果您要迁移单个 SQL Server 数据库。在单个数据库模式下，当从 PostgreSQL 访问时迁移的模式名称与原始的 SQL Server 模式名称相同。如果您想优化现有 SQL 查询以通过 PostgreSQL 连接运行，则这可以减少对现有 SQL 查询的代码更改。
- 如果您的最终目标是完全迁移到原生 Aurora PostgreSQL。在迁移之前，将您的架构整合到单个架构中 (dbo)，然后迁移到单个集群中以减少所需的更改。

在以下情况下使用多数据库迁移模式：

- 如果您想在同一个实例中使用多个用户数据库获得原定设置 SQL Server 体验。
- 如果需要一起迁移多个用户数据库。

Babelfish 的数据库集群参数组设置

当您创建 Aurora PostgreSQL 数据库集群并选择 Turn on Babelfish（开启 Babelfish）时，如果您选择 Create new（新建），将自动为您创建数据库集群参数组。此数据库集群参数组基于为安装选择的 Aurora PostgreSQL 版本（例如 Aurora PostgreSQL 版本 14）的 Aurora PostgreSQL 数据库集群参数组。它使用以下一般模式命名：

```
custom-aurora-postgresql14-babelfish-compat-3
```

您可以在集群创建过程中更改以下设置，但其中一些设置一旦存储在自定义参数组中，就无法更改，因此请仔细选择：

- 单个数据库或多个数据库
- 原定设置排序规则区域设置
- 排序规则名称
- 数据库参数组

要使用现有 Aurora PostgreSQL 数据库集群版本 13 或更高版本的参数组，请编辑该组并将 `babelfish_status` 参数设置为 on。在创建 Aurora PostgreSQL 集群之前，请指定任何 Babelfish 选项。要了解更多信息，请参阅Amazon Aurora 的参数组。

以下参数控制 Babelfish 首选项。除非说明中另有规定，否则参数是可修改的。说明中包含原定设置值。要查看任何参数的允许值，请执行以下操作：

> **Note**
>
> 将新数据库参数组与数据库实例关联时，修改后的静态和动态参数仅在数据库实例重新启动后得到应用。但是，如果在将数据库参数组与数据库实例关联之后修改数据库参数组中的动态参数，这些更改将立即得到应用，而无需重启。

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 从导航窗格中，选择 Parameter groups（参数组）。
3. 从列表中选择 `default.aurora-postgresql14` 数据库集群参数组。
4. 在搜索字段中输入参数的名称。例如，在搜索字段中输入 `babelfishpg_tsql.default_locale`，以显示此参数及其原定设置值和允许的设置。

> **Note**
>
> 只有在辅助区域中开启以下参数后，适用于 Aurora PostgreSQL 的 Babelfish 全球数据库才能在这些区域中运行。

参数	描述	应用类型	是否可修改
babelfishpg_tsql.apg_enable_correlated_scalar_transform	使计划程序能够在 Babelfish 中转换相关的标量子查询。（默认值：打开）（支持的值：打开、关闭）	dynamic	true
babelfishpg_tsql.apg_enable_subquery_cache	支持在 Babelfish 中对关联标量子查询使用缓存。（默认值：打开）（支持的值：打开、关闭）	dynamic	true
babelfishpg_tds.tds_default_numeric_scale	如果引擎未指定一个，则设置要在 TDS 列元数据中发送的数字类型的原定设置扩展。（原定设置值：8）（允许的值：0–38）	dynamic	true
babelfishpg_tds.tds_default_numeric_precision	一个整数，用于设置要在 TDS 列元数据中发送的数字类型的原定设置精度（如果引擎未指定此精度）。（原定设置值：38）（允许的值：1–38）	dynamic	true

参数	描述	应用类型	是否可修改
babelfishpg_tds.tds_default_packet_size	一个整数，设置用于连接 SQL Server 客户端的原定设置数据包大小。（原定设置值：4096）（允许的值：512–32767）	dynamic	true
babelfishpg_tds.tds_default_protocol_version	一个整数，设置用于连接客户端的原定设置 TDS 协议版本。（原定设置值：DEFAULT）（允许的值：TDSv7.0、TDSv7.1、TDSv7.1.1、TDSv7.2、TDSv7.3A、TDSv7.3B、TDSv7.4、DEFAULT）	dynamic	true
babelfishpg_tds.default_server_name	一个字符串，用于标识 Babelfish 服务器的原定设置名称。（原定设置值：Microsoft SQL Server）（允许的值：null）	dynamic	true
babelfishpg_tds.tds_debug_log_level	一个整数，用于在 TDS 中设置日志记录级别；0 表示关闭日志记录。（原定设置值：1）（允许的值：0、1、2、3）	dynamic	true

参数	描述	应用类型	是否可修改
babelfishpg_tds.listen_addresses	一个字符串，用于设置要在其上监听 TDS 的主机名或 IP 地址。创建 Babelfish 数据库集群后将无法修改此参数。（原定设置值：*）（允许的值：null）	–	false
babelfishpg_tds.port	一个整数，指定在 SQL Server 语法中用于请求的 TCP 端口。（原定设置值：1433）（允许的值：1–65535）	静态	true
babelfishpg_tds.tds_ssl_crypt	一个布尔值，用于为遍历 TDS 侦听器端口的数据打开 (0) 或关闭 (1) 加密。有关使用 SSL 进行客户端连接的详细信息，请参阅 Babelfish SSL 设置和客户端连接。（原定设置值：0）（允许的值：0、1）	dynamic	true
babelfishpg_tds.tds_ssl_max_protocol_version	一个字符串，指定要用于 TDS 会话的最高 SSL/TLS 协议版本。（原定设置值：“TLSv1.2”）（允许的值：“TLSv1”、“TLSv1.1”、“TLSv1.2”）	dynamic	true

参数	描述	应用类型	是否可修改
babelfishpg_tds.tds_ssl_min_protocol_version	一个字符串，指定要用于 TDS 会话的最低 SSL/TLS 协议版本。（默认值：在 Aurora PostgreSQL 版本 16 中，为“TLSv1.2”；对于比 Aurora PostgreSQL 版本 16 更早的版本，为“TLSv1”）（允许的值：“TLSv1”、“TLSv1.1”、“TLSv1.2”）	dynamic	true
babelfishpg_tds.unix_socket_directories	一个字符串，用于标识 TDS 服务器 Unix 套接字目录。创建 Babelfish 数据库集群后将无法修改此参数。（原定设置值：/tmp）（允许的值：null）	–	false
babelfishpg_tds.unix_socket_group	一个字符串，用于标识 TDS 服务器 Unix 套接字组。创建 Babelfish 数据库集群后将无法修改此参数。（原定设置值：rdsdb）（允许的值：null）	–	false

参数	描述	应用类型	是否可修改
babelfishpg_tsql.default_locale	一个字符串，指定用于 Babelfish 排序规则的原定设置区域设置。原定设置区域设置只是区域设置，不包括任何限定符。 在预置 Babelfish 数据库集群时设置此参数。预调配数据库集群后，将忽略对此参数的更改。（原定设置值：en_US）（允许的值：请参阅表）	静态	true
babelfishpg_tsql.migration_mode	一个不可修改的列表，用于指定对单用户或多用户数据库的支持。在预置 Babelfish 数据库集群时设置此参数。预调配数据库集群后，无法修改此参数的值。（默认值：在 Aurora PostgreSQL 版本 16 中，为 multi-db；对于比 Aurora PostgreSQL 版本 16 更早的版本，为 single-db）（允许的值：single-db、multi-db、null）	静态	true

参数	描述	应用类型	是否可修改
babelfishpg_tsql.server_collation_name	一个字符串，指定用于服务器级别操作的排序规则的名称。在预置 Babelfish 数据库集群时设置此参数。预置数据库集群后，请勿修改此参数的值。（原定设置值：bbf_unicode_general_ci_as）（允许的值：请参阅表）	静态	true
babelfishpg_tsql.version	一个字符串，用于设置 @@VERSION 变量的输出。请勿为 Aurora PostgreSQL 数据库集群修改此值。（原定设置值：null）（允许的值：原定设置值）	dynamic	true
rds.babelfish_status	一个字符串，用于设置 Babelfish 功能的状态。如果此参数设置为 `datatypes only` ，Babelfish 将关闭，但 SQL Server 数据类型仍可用。（原定设置值：off）（允许的值：on、off、datatypesonly）	静态	true

参数	描述	应用类型	是否可修改
unix_socket_permissions	设置 TDS 服务器 Unix 套接字权限的整数。创建 Babelfish 数据库集群后将无法修改此参数。（原定设置值：0700）（允许的值：0–511）	–	false

Babelfish SSL 设置和客户端连接

要想要求使用 SSL/TLS 连接到适用于 Aurora PostgreSQL 的 Babelfish 数据库集群，请使用 `rds.force_ssl` 参数。

- 要想要求使用 SSL/TLS 连接，请将 `rds.force_ssl` 参数值设置为 1（开启）。
- 要想不要求使用 SSL/TLS 连接，请将 `rds.force_ssl` 参数值设置为 0（关闭）。

此参数的默认值取决于 Aurora PostgreSQL 版本：

- 对于 Aurora PostgreSQL 版本 17 及更高版本：默认值为 1（开启）。
- 对于 Aurora PostgreSQL 版本 16 及较低版本：默认值为 0（关闭）。

> **Note**
>
> 当您从 Aurora PostgreSQL 版本 16 或更早版本执行主要版本升级，从而升级到版本 17 或更高版本时，参数的默认值将从 0（关闭）更改为 1（开启）。对于未配置 SSL 的应用程序，此更改可能会导致连接故障。您可以将此参数设置为 0（关闭），从而恢复到之前的默认行为。

有关驱动程序特定的详细信息，请参阅连接到 Babelfish 数据库集群。

当客户端连接到 TDS 端口（原定设置为 1433）时，Babelfish 将在客户端握手过程中发送的安全套接字层 (SSL) 设置与 Babelfish SSL 参数设置 (`tds_ssl_encrypt`) 进行比较。然后，Babelfish 确定是否允许连接。如果允许连接，则强制执行加密行为与否具体取决于您的参数设置和客户端提供的加密支持。

下表显示了 Babelfish 对于每个组合的行为。

客户端 SSL 设置	Babelfish SSL 设置	rds.force_ssl	是否允许连接？	返回给客户端的值
ENCRYPT_ON	任何	任何	允许，整个连接都已加密	ENCRYPT_ON
ENCRYPT_OFF	tds_ssl_encrypt=1	任何	允许，整个连接都已加密	ENCRYPT_REQ

客户端 SSL 设置	Babelfish SSL 设置	rds.force_ssl	是否允许连接？	返回给客户端的值
ENCRYPT_OFF	tds_ssl_encrypt=0	rds.force_ssl=0	允许，登录数据包已加密	ENCRYPT_OFF
ENCRYPT_OFF	tds_ssl_encrypt=0	rds.force_ssl=1	否，连接已关闭	ENCRYPT_OFF
ENCRYPT_NOT_SUP	tds_ssl_encrypt=0	rds.force_ssl=0	是	ENCRYPT_NOT_SUP
ENCRYPT_NOT_SUP	tds_ssl_encrypt=1	任何	否，连接已关闭	ENCRYPT_REQ
ENCRYPT_NOT_SUP	tds_ssl_encrypt=0	rds.force_ssl=1	否，连接已关闭	ENCRYPT_NOT_SUP
ENCRYPT_CLIENT_CERT	任何	任何	否，连接已关闭	不支持

了解适用于 Aurora PostgreSQL 的 Babelfish 中的排序规则

使用 Babelfish 创建 Aurora PostgreSQL 数据库集群时，请为数据选择排序规则。排序规则 指定以给定的书面人类语言生成文本或字符的排序顺序和位模式。排序规则包括用于比较一组给定位模式的数据的规则。排序规则与本地化有关。不同的区域设置会影响字符映射、排序顺序等。排序规则属性反映在各种排序规则的名称中。有关属性的信息，请参阅 Babelfish collation attributes table。

Babelfish 将 SQL Server 排序规则映射到 Babelfish 提供的类似排序规则。Babelfish 使用文化敏感的字符串比较和排序顺序预先定义了 Unicode 排序规则。Babelfish 还提供了一种将 SQL Server 数据库中的排序规则转换为最接近匹配的 Babelfish 排序规则的方法。针对不同的语言和区域提供了特定于区域设置的排序规则。

某些排序规则指定了与客户端编码对应的代码页。根据每个输出列的排序规则，Babelfish 会自动从服务器编码转换为客户端编码。

Babelfish 支持 Babelfish supported collations table中列出的排序规则。Babelfish 将 SQL Server 排序规则映射到 Babelfish 提供的类似排序规则。

Babelfish 使用 International Components for Unicode 排序规则库的 153.80 版本。有关 ICU 排序规则的更多信息，请参阅 ICU 文档中的排序规则。要了解有关 PostgreSQL 和排序规则的更多信息，请参阅 PostgreSQL 文档中的排序规则支持。

主题

- 控制排序规则和区域设置的数据库集群参数
- Babelfish 中的确定性和非确定性排序规则
- Babelfish 支持数据库级别的排序规则
- Babelfish 中的服务器和对象排序规则
- Babelfish 中的默认排序规则行为
- 管理排序规则
- 排序规则限制和行为区别

控制排序规则和区域设置的数据库集群参数

以下参数会影响排序规则行为。

babelfishpg_tsql.default_locale

此参数指定排序规则使用的默认区域设置。此参数与 Babelfish collation attributes table中列出的属性结合使用，以便为特定语言和区域自定义排序规则。此参数的默认值为 en-US。

默认区域设置适用于以“BBF”开头的所有 Babelfish 排序规则名称，以及映射到 Babelfish 排序规则的所有 SQL Server 排序规则。在现有 Babelfish 数据库集群上更改此参数的设置不会影响现有排序规则的区域设置。有关排序规则的列表，请参阅 Babelfish supported collations table。

babelfishpg_tsql.server_collation_name

此参数指定服务器（Aurora PostgreSQL 数据库集群实例）和数据库的默认排序规则。默认值为 sql_latin1_general_cp1_ci_as。server_collation_name 必须是一种 CI_AS 排序规则，是因为在 T-SQL 中，服务器排序规则决定了标识符的比较方式。

当您创建 Babelfish 数据库集群时，可以从可供选择的列表中选择 Collation name（排序规则名称）。其中包括 Babelfish supported collations table中列出的排序规则。请勿在 Babelfish 数据库创建之后修改 server_collation_name。

您在创建 Babelfish for Aurora PostgreSQL 数据库集群时选择的设置存储在与此集群关联且与这些参数对应的数据库集群参数组中，并设置其排序规则行为。

Babelfish 中的确定性和非确定性排序规则

Babelfish 支持确定性和非确定性排序规则：

- 确定性排序规则 将具有相同字节序列的字符评估为相等。这意味着 x 和 X 在确定性排序规则中是不相等的。确定性的排序规则可以区分大小写 (CS) 且区分重音 (AS)。
- 非确定性排序规则 不需要完全相同的匹配。非确定性排序规则评估 x 和 X 相等。非确定性排序规则不区分大小写（CI）和/或不区分重音（AI）。

在下表中，您可以找到在使用非确定性排序规则时，Babelfish 和 PostgreSQL 之间的一些行为差异。

Babelfish	PostgreSQL
对于 CI_AS 排序规则支持 LIKE 子句。	对于非确定性排序规则不支持 LIKE 子句。
仅在 Babelfish 版本 4.2.0 的以下 AI 排序规则中才支持 LIKE 子句：	对于非确定性排序规则不支持 LIKE 子句。

Babelfish	PostgreSQL
• bbf_unicode_cp1250_ci_ai • bbf_unicode_cp1250_cs_ai • bbf_unicode_cp1257_ci_ai • bbf_unicode_cp1257_cs_ai • bbf_unicode_cp1_ci_ai • bbf_unicode_cp1_cs_ai • estonian_ci_ai • finnish_swedish_ci_ai • french_ci_ai • latin1_general_ci_ai • latin1_general_cs_ai • modern_spanish_ci_ai • polish_ci_ai • sql_latin1_general_cp1_ci_ai • sql_latin1_general_cp1_cs_ai • traditional_spanish_ci_ai	

有关 Babelfish 与 SQL Server 和 PostgreSQL 相比的其他限制和行为差异的列表，请参阅排序规则限制和行为区别。

Babelfish 和 SQL Server 遵循描述排序规则属性的排序规则命名约定，如下表所示。

属性	描述
AI	不区分重音。
AS	区分重音。
BIN2	BIN2 请求按代码点顺序对数据进行排序。Unicode 代码点顺序与 UTF-8、UTF-16 和 UCS-2 编码的字符顺序相同。代码点顺序是一种快速的确定性排序规则。

属性	描述
CI	不区分大小写。
CS	区分大小写。
PREF	要在小写字母之前对大写字母进行排序，请使用 PREF 排序规则。如果比较不区分大小写，在没有其他区别的情况下，字母的大写版本排序将在小写版本之前。ICU 库支持带有 `colCaseFirst=upper` 的大写字母首选项，但不适用于 CI_AS 排序规则。 PREF 只能应用于 CS_AS 确定性排序规则。

Babelfish 支持数据库级别的排序规则

Babelfish 支持下列数据库级别的排序规则：

- bbf_unicode_bin2
- bbf_unicode_cp1_ci_ai
- bbf_unicode_cp1_ci_as
- bbf_unicode_cp1250_ci_ai
- bbf_unicode_cp1250_ci_as
- bbf_unicode_cp1257_ci_ai
- bbf_unicode_cp1257_ci_as
- estonian_ci_ai
- estonian_ci_as
- finnish_swedish_ci_ai
- finnish_swedish_ci_as
- french_ci_ai
- french_ci_as
- latin1_general_bin2
- latin1_general_ci_ai
- latin1_general_ci_as
- latin1_general_90_bin2

- latin1_general_100_bin2
- latin1_general_140_bin2
- modern_spanish_ci_ai
- modern_spanish_ci_as
- polish_ci_ai
- polish_ci_as
- sql_latin1_general_cp1_ci_ai
- sql_latin1_general_cp1_ci_as
- sql_latin1_general_cp1250_ci_as
- sql_latin1_general_cp1251_ci_as
- sql_latin1_general_cp1257_ci_as
- traditional_spanish_ci_ai
- traditional_spanish_ci_as

> **Note**
>
> 要在数据库级别使用一个不同的排序规则，请确保它与服务器级别排序规则相匹配。有关更多信息，请参阅 Babelfish 中的服务器和对象排序规则

Babelfish 中的服务器和对象排序规则

将以下排序规则用作服务器排序规则或对象排序规则。

排序规则 ID	备注
bbf_unicode_general_ci_as	支持不区分大小写的比较和 LIKE 运算符。
bbf_unicode_cp1_ci_as	非确定性排序规则也被称为 CP1252。
bbf_unicode_CP1250_ci_as	非确定性排序规则用于表示使用拉丁文字的中欧和东欧语言的文本。

排序规则 ID	备注
bbf_unicode_CP1251_ci_as	使用西里尔文字的语言的非确定性排序规则。
bbf_unicode_cp1253_ci_as	用于代表现代希腊语的非确定性排序规则。
bbf_unicode_cp1254_ci_as	支持土耳其语的非确定性排序规则。
bbf_unicode_cp1255_ci_as	支持希伯来语的非确定性排序规则。
bbf_unicode_cp1256_ci_as	用来编写使用阿拉伯文字的语言的非确定性排序规则。
bbf_unicode_cp1257_ci_as	用于支持爱沙尼亚语、拉脱维亚语和立陶宛语的非确定性排序规则。
bbf_unicode_cp1258_ci_as	用于编写越南语字符的非确定性排序规则。
bbf_unicode_cp874_ci_as	用于编写泰语字符的非确定性排序规则。
sql_latin1_general_cp1250_ci_as	用来表示拉丁字符的非确定性单字节字符编码。
sql_latin1_general_cp1251_ci_as	支持拉丁字符的非确定性排序规则。
sql_latin1_general_cp1_ci_as	支持拉丁字符的非确定性排序规则。
sql_latin1_general_cp1253_ci_as	支持拉丁字符的非确定性排序规则。
sql_latin1_general_cp1254_ci_as	支持拉丁字符的非确定性排序规则。

排序规则 ID	备注
sql_latin1_general_cp1255_ci_as	支持拉丁字符的非确定性排序规则。
sql_latin1_general_cp1256_ci_as	支持拉丁字符的非确定性排序规则。
sql_latin1_general_cp1257_ci_as	支持拉丁字符的非确定性排序规则。
sql_latin1_general_cp1258_ci_as	支持拉丁字符的非确定性排序规则。
chinese_prc_ci_as	支持中文 (PRC) 的非确定性排序规则。
cyrillic_general_ci_as	支持西里尔文的非确定性排序规则。
finnish_swedish_ci_as	支持芬兰语的非确定性排序规则。
french_ci_as	支持法语的非确定性排序规则。
japanese_ci_as	支持日语的非确定性排序规则。在 Babelfish 2.1.0 及更高版本中支持。
korean_wansung_ci_as	支持韩语的非确定性排序规则（带字典排序）。
latin1_general_ci_as	支持拉丁字符的非确定性排序规则。
modern_spanish_ci_as	支持现代西班牙语的非确定性排序规则。
polish_ci_as	支持波兰语的非确定性排序规则。

排序规则 ID	备注
thai_ci_as	支持泰语的非确定性排序规则。
traditional_spanish_ci_as	支持西班牙语的非确定性排序规则（传统排序）。
turkish_ci_as	支持土耳其语的非确定性排序规则。
ukrainian_ci_as	支持乌克兰语的非确定性排序规则。
vietnamese_ci_as	支持越南语的非确定性排序规则。

您可以将以下排序规则作为对象排序规则。

方言	确定性选项	非确定性选项
阿拉伯语	Arabic_CS_AS	Arabic_CI_AS Arabic_CI_AI
阿拉伯语文字	BBF_Unicode_CP1256_CS_AS BBF_Unicode_Pref_CP1256_CS_AS	BBF_Unicode_CP1256_CI_AI BBF_Unicode_CP1256_CS_AI
二元	latin1_general_bin2 BBF_Unicode_BIN2	–
使用拉丁字母的中欧和东欧语言	BBF_Unicode_CP1250_CS_AS BBF_Unicode_Pref_CP1250_CS_AS	BBF_Unicode_CP1250_CI_AI BBF_Unicode_CP1250_CS_AI
中文	Chinese_PRC_CS_AS	Chinese_PRC_CI_AS

方言	确定性选项	非确定性选项
		Chinese_PRC_CI_AI
Cyrillic_General	Cyrillic_General_CS_AS	Cyrillic_General_CI_AS Cyrillic_General_CI_AI
西里尔文字	BBF_Unicode_CP1251_CS_AS BBF_Unicode_Pref_CP1251_CS_AS	BBF_Unicode_CP1251_CI_AI BBF_Unicode_CP1251_CS_AI
爱沙尼亚语	Estonian_CS_AS	Estonian_CI_AS Estonian_CI_AI
爱沙尼亚语、拉脱维亚语和立陶宛语	BBF_Unicode_CP1257_CS_AS BBF_Unicode_Pref_CP1257_CS_AS	BBF_Unicode_CP1257_CI_AI BBF_Unicode_CP1257_CS_AI
Finnish_Swedish	Finnish_Swedish_CS_AS	Finnish_Swedish_CI_AS Finnish_Swedish_CI_AI
法语	French_CS_AS	French_CI_AS French_CI_AI
希腊语	Greek_CS_AS	Greek_CI_AS Greek_CI_AI

方言	确定性选项	非确定性选项
希伯来语	BBF_Unicode_CP1255_CS_AS BBF_Unicode_Pref_CP1255_CS_AS Hebrew_CS_AS	BBF_Unicode_CP1255_CI_AI BBF_Unicode_CP1255_CS_AI Hebrew_CI_AS Hebrew_CI_AI
日语（Babelfish 2.1.0 及更高版本）	Japanese_CS_AS	Japanese_CI_AI Japanese_CI_AS
Korean_Wamsung	Korean_Wamsung_CS_AS	Korean_Wamsung_CI_AS Korean_Wamsung_CI_AI
代码页 CP1252 的拉丁字符	latin1_general_cs_as BBF_Unicode_General_CS_AS BBF_Unicode_General_Pref_CS_AS BBF_Unicode_Pref_CP1_CS_AS BBF_Unicode_CP1_CS_AS	latin1_general_ci_as latin1_general_ci_ai latin1_general_cs_ai BBF_Unicode_General_CI_AI BBF_Unicode_General_CS_AI BBF_Unicode_CP1_CI_AI BBF_Unicode_CP1_CS_AI
现代希腊语	BBF_Unicode_CP1253_CS_AS BBF_Unicode_Pref_CP1253_CS_AS	BBF_Unicode_CP1253_CI_AI BBF_Unicode_CP1253_CS_AI

方言	确定性选项	非确定性选项
Modern_Spanish	Modern_Spanish_CS_AS	Modern_Spanish_CI_AS Modern_Spanish_CI_AI
蒙古语	Mongolian_CS_AS	Mongolian_CI_AS Mongolian_CI_AI
波兰语	Polish_CS_AS	Polish_CI_AS Polish_CI_AI
泰语	BBF_Unicode_CP874_CS_AS BBF_Unicode_Pref_CP874_CS_AS Thai_CS_AS	BBF_Unicode_CP874_CI_AI BBF_Unicode_CP874_CS_AI Thai_CI_AS、Thai_CI_AI
Traditional_Spanish	Traditional_Spanish_CS_AS	Traditional_Spanish_CI_AS Traditional_Spanish_CI_AI
土耳其语	BBF_Unicode_CP1254_CS_AS BBF_Unicode_Pref_CP1254_CS_AS Turkish_CS_AS	BBF_Unicode_CP1254_CI_AI BBF_Unicode_CP1254_CS_AI Turkish_CI_AS、Turkish_CI_AI
乌克兰语	Ukranian_CS_AS	Ukranian_CI_AS Ukranian_CI_AI

方言	确定性选项	非确定性选项
越南语	BBF_Unicode_CP1258_CS_AS	BBF_Unicode_CP1258_CI_AI
	BBF_Unicode_Pref_CP1258_CS_AS	BBF_Unicode_CP1258_CS_AI
	Vietnamese_CS_AS	Vietnamese_CI_AS
		Vietnamese_CI_AI

Babelfish 中的默认排序规则行为

以前，可排序的数据类型的默认排序规则是 pg_catalog.default。数据类型和依赖于这些数据类型的对象遵循区分大小写的排序规则。这种情况可能会影响使用不区分大小写的排序规则的数据集的 T-SQL 对象。从 Babelfish 2.3.0 开始，可排序的数据类型（TEXT 和 NTEXT 除外）的默认排序规则与 babelfishpg_tsql.server_collation_name 参数中的排序规则相同。当您升级到 Babelfish 2.3.0 时，将在创建数据库集群时自动选择默认排序规则，这不会产生任何明显的影响。

管理排序规则

ICU 库提供排序规则版本跟踪功能，以确保当新版本的 ICU 可用时，依赖排序规则的索引可以重新建立索引。要查看当前数据库是否有需要刷新的排序规则，可以在使用 psql 或 pgAdmin 进行连接后使用以下查询：

```
SELECT pg_describe_object(refclassid, refobjid,
    refobjsubid) AS "Collation",
    pg_describe_object(classid, objid, objsubid) AS "Object"
    FROM pg_depend d JOIN pg_collation c ON refclassid = 'pg_collation'::regclass
    AND refobjid = c.oid WHERE c.collversion <> pg_collation_actual_version(c.oid)
    ORDER BY 1, 2;
```

此查询返回如下输出：

```
 Collation | Object
-----------+--------
(0 rows)
```

在此示例中，不需要更新排序规则。

要获取 Babelfish 数据库中预定义排序规则的列表，您可以将 psql 或 pgAdmin 用于以下查询：

```
SELECT * FROM pg_collation;
```

预定义的排序规则存储在 sys.fn_helpcollations 表中。您可以使用以下命令显示有关排序规则的信息（例如其 lcid、样式和排序规则标志）。要使用 sqlcmd 获取所有排序规则的列表，请连接到 T-SQL 端口（原定设置为 1433），然后运行以下查询：

```
1> :setvar SQLCMDMAXVARTYPEWIDTH 40
2> :setvar SQLCMDMAXFIXEDTYPEWIDTH 40
3> SELECT * FROM fn_helpcollations()
4> GO
name                                    description
--------------------------------------- ---------------------------------------
arabic_cs_as                            Arabic, case-sensitive, accent-sensitive
arabic_ci_ai                            Arabic, case-insensitive, accent-insensi
arabic_ci_as                            Arabic, case-insensitive, accent-sensiti
bbf_unicode_bin2                        Unicode-General, case-sensitive, accent-
bbf_unicode_cp1250_ci_ai                Default locale, code page 1250, case-ins
bbf_unicode_cp1250_ci_as                Default locale, code page 1250, case-ins
bbf_unicode_cp1250_cs_ai                Default locale, code page 1250, case-sen
bbf_unicode_cp1250_cs_as                Default locale, code page 1250, case-sen
bbf_unicode_pref_cp1250_cs_as           Default locale, code page 1250, case-sen
bbf_unicode_cp1251_ci_ai                Default locale, code page 1251, case-ins
bbf_unicode_cp1251_ci_as                Default locale, code page 1251, case-ins
bbf_unicode_cp1254_ci_ai                Default locale, code page 1254, case-ins
...
(124 rows affected)
```

示例中显示的第 1 行和第 2 行仅出于文档可读性目的而缩小了输出范围。

```
1> SELECT SERVERPROPERTY('COLLATION')
2> GO
serverproperty
------------------------------------------------------------------------------
sql_latin1_general_cp1_ci_as

(1 rows affected)
1>
```

排序规则限制和行为区别

Babelfish 使用 ICU 库进行排序规则支持。PostgreSQL 使用特定版本的 ICU 构建，最多可以匹配一个排序规则版本。不同版本之间的差异是不可避免的，随着语言的变化，随着时间的推移，会产生一些细微的变化。在下面的列表中，您可以找到 Babelfish 排序规则的一些已知限制和行为变体：

- 索引和排序规则类型依赖关系 – 当库版本更改时，依赖于 International Components for Unicode (ICU) 排序规则库（Babelfish 使用的库）的用户定义类型的索引不会失效。
- COLLATIONPROPERTY 函数 – 排序规则属性仅针对支持的 Babelfish BBF 排序规则实现。有关更多信息，请参阅 Babelfish supported collations table。
- Unicode 排序规则差异 – SQL Server 的 SQL 排序规则对 Unicode 编码的数据（`nchar` 和 `nvarchar`）的排序方式不同于非 Unicode 编码的数据（`char` 和 `varchar`）。Babelfish 数据库始终是 UTF-8 编码的，并且始终一致地应用 Unicode 排序规则，而无论数据类型如何，因此针对 `char` 或 `varchar` 的排序顺序与针对 `nchar` 或 `nvarchar` 的排序顺序相同。
- 二级相等排序规则和排序行为 – 原定设置的 ICU Unicode 二级相等 (`CI_AS`) 排序规则将标点符号和其他非字母数字字符排序在数字字符之前，将数字字符排序在字母字符之前。但是，标点符号和其他特殊字符的顺序不同。
- 三级排序规则，ORDER BY 的解决方法 – `SQL_Latin1_General_Pref_CP1_CI_AS` 之类的 SQL 排序规则支持 `TERTIARY_WEIGHTS` 函数并能够对 `CI_AS` 排序规则中进行同等比较的字符串进行排序，大写字母优先排序：ABC、ABc、AbC、Abc、aBC、aBc、abC，最后是 abc。因此，`DENSE_RANK OVER (ORDER BY column)` 分析函数将这些字符串评估为具有相同的排名，但首先在分区中对它们用大写字母进行排序。

  您可以通过添加一个 `COLLATE` 子句到指定 `@colCaseFirst=upper` 的三级 `CS_AS` 排序规则的 `ORDER BY` 子句中来获取与 Babelfish 相似的结果。但是，`colCaseFirst` 修饰符仅适用于三级相等的字符串（而不是 `CI_AS` 排序规则之类的二级相等字符串）。因此，您无法使用单个 ICU 排序规则模拟三级 SQL 排序规则。

  作为解决办法，我们建议您修改使用 `SQL_Latin1_General_Pref_CP1_CI_AS` 排序规则的应用程序，以首先使用 `BBF_SQL_Latin1_General_CP1_CI_AS` 排序规则。然后将 `COLLATE BBF_SQL_Latin1_General_Pref_CP1_CS_AS` 添加到此列的任何 `ORDER BY` 子句中。
- 字符扩展 – 字符扩展将单个字符视为等于主级别的一系列字符。SQL Server 的原定设置 `CI_AS` 排序规则支持字符扩展。ICU 排序规则仅支持对不区分重音的排序规则进行字符扩展。

  当需要扩展字符时，请使用 `AI` 排序规则进行比较。但是，LIKE 运算符目前不支持此类排序规则。

- char 和 varchar 编码 – 当将 SQL 排序规则用于 `char` 或 `varchar` 数据类型时，ASCII 127 之前的字符的排序顺序由该 SQL 排序规则的特定代码页决定。对于 SQL 排序规则，声明为 `char` 或 `varchar` 的字符串可能与声明为 `nchar` 或 `nvarchar` 的字符串的排序方式不同。

  PostgreSQL 使用数据库编码对所有字符串进行编码，因此，所有字符转换为 UTF-8 并使用 Unicode 规则进行排序。

  由于 SQL 排序规则使用 Unicode 规则对 nchar 和 nvarchar 数据类型进行排序，因此，Babelfish 会使用 UTF-8 对服务器上的所有字符串进行编码。Babelfish 使用 Unicode 规则对 nchar 和 nvarchar 字符串进行排序的方式与对 char 和 varchar 字符串进行排序的方式相同。
- 补充字符 – SQL Server 函数 `NCHAR`、`UNICODE` 和 `LEN` 支持 Unicode 基本多语言平面 (BMP) 之外的代码点的字符。相比之下，非 SC 排序规则使用代理对字符来处理补充字符。对于 Unicode 数据类型，SQL Server 可以使用 UCS-2 最多表示 65535 个字符，如果使用补充字符，则表示完整的 Unicode 范围（1114114 个字符）。
- 区分假名 (KS) 排序规则 – 区分假名 (KS) 排序规则是以不同方式处理 `Hiragana` 和 `Katakana` 日语假名字符的排序规则。ICU 支持日语排序规则标准 `JIS X 4061`。现在已弃用的 `colhiraganaQ [on | off]` 区域设置修饰符可能会提供与 KS 排序规则相同的功能。但是，Babelfish 目前不支持与 SQL Server 同名的 KS 排序规则。
- 区分全角和半角 (WS) 排序规则 – 如果单字节字符（半角）和表示为双字节字符（全角）的同一个字符被区别对待，则排序规则被称为区分全角和半角 (WS)。Babelfish 目前不支持与 SQL Server 同名的 WS 排序规则。
- 变体选择器敏感性 (VSS) 排序规则 – 变体选择器敏感性 (VSS) 排序规则区分日语排序规则 `Japanese_Bushu_Kakusu_140` 和 `Japanese_XJIS_140` 中的表意文字变体选择器。变体序列由基本字符加上一个额外的变体选择器组成。如果您不选择 `_VSS` 选项，则在比较中不考虑变体选择器。

  Babelfish 目前不支持 VSS 排序规则。
- BIN 和 BIN2 排序规则 – BIN2 排序规则根据代码点顺序对字符进行排序。UTF-8 的逐字节二进制顺序保留了 Unicode 代码点顺序，因此这也可能是性能最佳的排序规则。如果 Unicode 代码点顺序适用于应用程序，请考虑使用 BIN2 排序规则。但是，使用 BIN2 排序规则可能会导致数据以一种文化上意想不到的顺序在客户端上显示。随着时间的推移，小写字符的新映射将添加到 Unicode 中，因此 `LOWER` 函数在不同版本的 ICU 上可能会有所不同。这是比较一般的排序规则版本控制问题的特殊情况，而不是 BIN2 排序规定的特定情况。

  Babelfish 以 Babelfish 发行版提供 `BBF_Latin1_General_BIN2` 排序规则，以便按 Unicode 代码点顺序进行排序。在 BIN 排序规则中，只有第一个字符被排序为 wchar。剩余字符按字节排序，以有效地根据其编码按代码点顺序排序。这种方法不遵循 Unicode 排序规则，也不受 Babelfish 支持。

- 非确定性排序规则和 CHARINDEX 限制 – 对于早于版本 2.1.0 的 Babelfish 版本，您不能将 CHARINDEX 与非确定性排序规则结合使用。原定设置情况下，Babelfish 使用不区分大小写（不确定性）排序规则。对较旧版本的 Babelfish 使用 CHARINDEX 会引发以下运行时错误：

```
nondeterministic collations are not supported for substring searches
```

> **Note**
>
> 此限制和解决方法仅适用于 Babelfish 版本 1.x（Aurora PostgreSQL 13.x 版本）。Babelfish 2.1.0 及更高版本没有此问题。

您可以通过以下方式之一解决此问题：

- 将表达式显式转换为区分大小写的排序规则，然后通过应用 LOWER 或 UPER 将两个参数转换为大写。例如，`SELECT charindex('x', a) FROM t1` 将变为以下内容：

```
SELECT charindex(LOWER('x'), LOWER(a COLLATE sql_latin1_general_cp1_cs_as)) FROM t1
```

- 创建一个 SQL 函数 f_charindex，然后将 CHARINDEX 调用替换为对以下函数的调用：

```
CREATE function f_charindex(@s1 varchar(max), @s2 varchar(max)) RETURNS int
AS
BEGIN
declare @i int = 1
WHILE len(@s2) >= len(@s1)
BEGIN
  if LOWER(@s1) = LOWER(substring(@s2,1,len(@s1))) return @i
  set @i += 1
  set @s2 = substring(@s2,2,999999999)
END
return 0
END
go
```

使用转义孵化管理 Babelfish 错误处理

在可能的情况下，Babelfish 会针对控制流和事务状态模仿 SQL 行为。当 Babelfish 遇到错误时，它会返回与 SQL Server 错误代码类似的错误代码。如果 Babelfish 无法将错误映射到 SQL Server 代码，它会返回一个固定的错误代码 (33557097)，并根据错误类型采取具体操作，如下所示：

- 对于编译时错误，Babelfish 会回滚事务。
- 对于运行时错误，Babelfish 会结束批处理并回滚事务。
- 对于客户端和服务器之间的协议错误，不会回滚该事务。

如果错误代码无法映射到等效代码并且类似错误的代码可用，则错误代码将映射到替代代码。例如，导致 SQL Server 代码 8143 和 8144 的行为都会映射到 8143。

无法映射的错误不尊重 `TRY... CATCH` 构造。

您可以使用 `@@ERROR` 返回 SQL Server 错误代码，或使用 `@@PGERROR` 函数返回 PostgreSQL 错误代码。您也可以使用 `fn_mapped_system_error_list` 函数返回映射的错误代码列表。有关 PostgreSQL 错误代码的信息，请参阅 PostgreSQL 网站。

修改 Babelfish 转义孵化设置

为了处理可能失败的语句，Babelfish 定义了一些称为转义孵化的选项。转义孵化是一个选项，用于在遇到不受支持的功能或语法时的指定 Babelfish 行为。

您可以使用 `sp_babelfish_configure` 存储过程来控制转义孵化的设置。使用脚本将转义孵化设置为 `ignore` 或 `strict`。如果设置为 `strict`，Babelfish 会返回一个错误，您需要纠正该错误后才能继续。

要将更改应用于当前会话和集群级别，请包含 `server` 关键字。

使用情况如下所示：

- 要列出所有转义孵化及其状态，以及使用信息，请运行 `sp_babelfish_configure`。
- 要列出命名的转义孵化及其值，对于当前会话或集群范围，请运行命令 `sp_babelfish_configure '`*`hatch_name`*`'`，其中 *`hatch_name`* 是一个或多个转义孵化的标识符。*`hatch_name`* 可以使用 SQL 通配符，例如“%”。
- 要将一个或多个转义孵化设置为指定的值，请运行 `sp_babelfish_configure ['`*`hatch_name`*`' [, 'strict'|'ignore' [, 'server']]`。要在集群级别上使设置永久化，请包括 `server` 关键字，如下面所示：

```
EXECUTE sp_babelfish_configure 'escape_hatch_unique_constraint', 'ignore', 'server'
```

要仅为当前会话设置它们，请不要使用 `server`。

- 要将所有转义孵化重置为默认值，请运行 `sp_babelfish_configure 'default'`（Babelfish 1.2.0 及更高版本）。

标识孵化（或多个孵化）的字符串可能包含 SQL 通配符。例如，以下选项为 Aurora PostgreSQL 集群将所有语法转义孵化设置为 `ignore`。

```
EXECUTE sp_babelfish_configure '%', 'ignore', 'server'
```

在下表中，您可以找到 Babelfish 预定义的转义孵化的描述和默认值。

转义孵化	描述	默认
`escape_hatch_checkpoint`	允许在过程代码中使用 CHECKPOINT 语句，但当前尚未实现 CHECKPOINT 语句。	忽略
`escape_hatch_constraint_name_for_default`	控制与默认约束名称相关的 Babelfish 行为。	忽略
`escape_hatch_database_misc_options`	控制与 CREATE DATABASE 上的以下选项相关的 Babelfish 行为：CONTAINMENT、DB_CHAINING、TRUSTWORTHY、PERSISTENT_LOG_BUFFER。	忽略
`escape_hatch_for_replication`	控制创建或更改表时与 [NOT] FOR REPLICATION 子句相关的 Babelfish 行为。	严格
`escape_hatch_fulltext`		

转义孵化	描述	默认
	控制与 FULLTEXT 功能相关的 Babelfish 行为，例如 CREATE/ALTER DATABASE、CREATE FULLTEXT INDEX 中的 DEFAULT_FULLTEXT_LANGUAGE 或 sp_fulltext_database。	忽略
escape_hatch_ignore_dup_key	控制与 CREATE/ALTER TABLE 和 CREATE INDEX 相关的 Babelfish 行为。当 IGNORE_DUP_KEY=ON 时，设置为 strict（默认值）时会引发错误，设置为 ignore 时会忽略错误（Babelfish 版本 1.2.0 及更高版本）。	严格
escape_hatch_index_clustering	控制与索引的 CLUSTERED 或 NONCLUSTERED 关键字相关的 Babelfish 行为以及 PRIMARY KEY 或 UNIQUE 约束。如果忽略 CLUSTERED，则仍会像指定了 NONCLUSTERED 一样创建索引或约束条件。	忽略
escape_hatch_index_columnstore	控制与 COLUMNSTORE 子句相关的 Babelfish 行为。如果您指定 ignore，Babelfish 会创建一个常规的 B 树索引。	严格
escape_hatch_join_hints	控制 JOIN 运算符中关键字的行为：LOOP、HASH、MERGE、REMOTE、REDUCE、REDISTRIBUTE、REPLICATE。	忽略

转义孵化	描述	默认
escape_hatch_language_non_english	控制屏幕上的消息中与英语以外的语言相关的 Babelfish 行为。Babelfish 目前仅对屏幕上的消息支持 us_english 。SET LANGUAGE 可能会使用包含语言名称的变量，因此只能在运行时检测到正在设置的实际语言。	严格
escape_hatch_login_hashed_password	如果被忽略，则会抑制 CREATE LOGIN 和 ALTER LOGIN 的 HASHED 关键字。	严格
escape_hatch_login_misc_options	如果被忽略，则除了 HASHED、MUST_CHANGE 、OLD_PASSWORD 之外，还会抑制其他关键字的错误，并且还会抑制 CREATE LOGIN 和 ALTER LOGIN 的 UNLOCK。	严格
escape_hatch_login_old_password	如果被忽略，则会抑制 CREATE LOGIN 和 ALTER LOGIN 的 OLD_PASSWORD 关键字。	严格
escape_hatch_login_password_must_change	如果被忽略，则会抑制 CREATE LOGIN 和 ALTER LOGIN 的 MUST_CHANGE 关键字。	严格
escape_hatch_login_password_unlock	如果被忽略，则会抑制 CREATE LOGIN 和 ALTER LOGIN 的 UNLOCK 关键字。	严格

转义孵化	描述	默认
escape_hatch_nocheck_add_constraint	控制与 WITH CHECK 或 NOCHECK 子句相关的 Babelfish 行为以获取约束条件。	严格
escape_hatch_nocheck_existing_constraint	控制与 FOREIGN KEY 或 CHECK 约束相关的 Babelfish 行为。	严格
escape_hatch_query_hints	控制与查询提示相关的 Babelfish 行为。当此选项设置为忽略时，服务器将忽略使用 OPTION (...) 子句指定查询处理方面的提示。示例包括 SELECT FROM ... OPTION(MERGE JOIN HASH, MAXRECURSION 10))。	忽略
escape_hatch_rowversion	控制 ROWVERSION 和 TIMESTAMP 数据类型的行为。有关使用信息，请参阅使用具有有限实施的 Babelfish 功能。	严格
escape_hatch_schemabinding_function	控制与 WITH SCHEMABINDING 子句相关的 Babelfish 行为。预设情况下，使用 CREATE 或 ALTER FUNCTION 命令指定时，WITH SCHEMABINDING 子句将被忽略。	忽略

转义孵化	描述	默认
escape_hatch_schemabinding_procedure	控制与 WITH SCHEMABINDING 子句相关的 Babelfish 行为。预设情况下，使用 CREATE 或 ALTER PROCEDURE 命令指定时，WITH SCHEMABINDING 子句将被忽略。	忽略
escape_hatch_rowguidcol_column	控制创建或更改表时与 ROWGUIDCOL 子句相关的 Babelfish 行为。	严格
escape_hatch_schemabinding_trigger	控制与 WITH SCHEMABINDING 子句相关的 Babelfish 行为。预设情况下，使用 CREATE 或 ALTER TRIGGER 命令指定时，WITH SCHEMABINDING 子句将被忽略。	忽略
escape_hatch_schemabinding_view	控制与 WITH SCHEMABINDING 子句相关的 Babelfish 行为。预设情况下，使用 CREATE 或 ALTER VIEW 命令指定时，WITH SCHEMABINDING 子句将被忽略。	忽略
escape_hatch_session_settings	控制 Babelfish 针对不受支持的会话级别 SET 语句的行为。	忽略

转义孵化	描述	默认
escape_hatch_showplan_all	控制与 SET SHOWPLAN_ALL 和 SET STATISTICS PROFILE 相关的 Babelfish 行为。当设置为 ignore 时，它们的行为与 SET BABELFISH_SHOWPLAN_ALL 和 SET BABELFISH_STATISTICS PROFILE 类似；设置为 strict 时，它们将被无提示忽略。	严格
escape_hatch_storage_on_partition	在定义分区时控制与 ON partition_scheme column 子句相关的 Babelfish 行为。Babelfish 目前没有实施分区。	严格
escape_hatch_storage_options	对 CREATE、ALTER DATABASE、TABLE、INDEX 中使用的任何存储选项的转义孵化。这包括为表、索引和约束以及为数据库定义存储位置（分区、文件组）的子句 (LOG) ON、TEXTIMAGE_ON、FILESTREAM_ON。此转义孵化设置适用于所有这些子句（包括 ON [PRIMARY] 和 ON“DEFAULT”）。例外情况是，使用 ON partition_scheme（列）为表或索引指定分区时。	忽略
escape_hatch_table_hints	控制使用 WITH (...) 子句指定的表提示的行为。	忽略

转义孵化	描述	默认
escape_hatch_unique_constraint	当设置为 strict 时，SQL Server 和 PostgreSQL 在处理索引列上的 NULL 值方面的模糊语义差异可能会引发错误。只有在不切实际的使用案例中才会出现语义差异，因此您可以将此转义孵化设置为“ignore”以避免看到错误。 在以下版本中已弃用：3.6.0 及更高版本、4.2.0 及更高版本	严格

创建 Babelfish for Aurora PostgreSQL 数据库集群

Aurora PostgreSQL 版本 13.4 及更高版本支持 Babelfish for Aurora PostgreSQL。

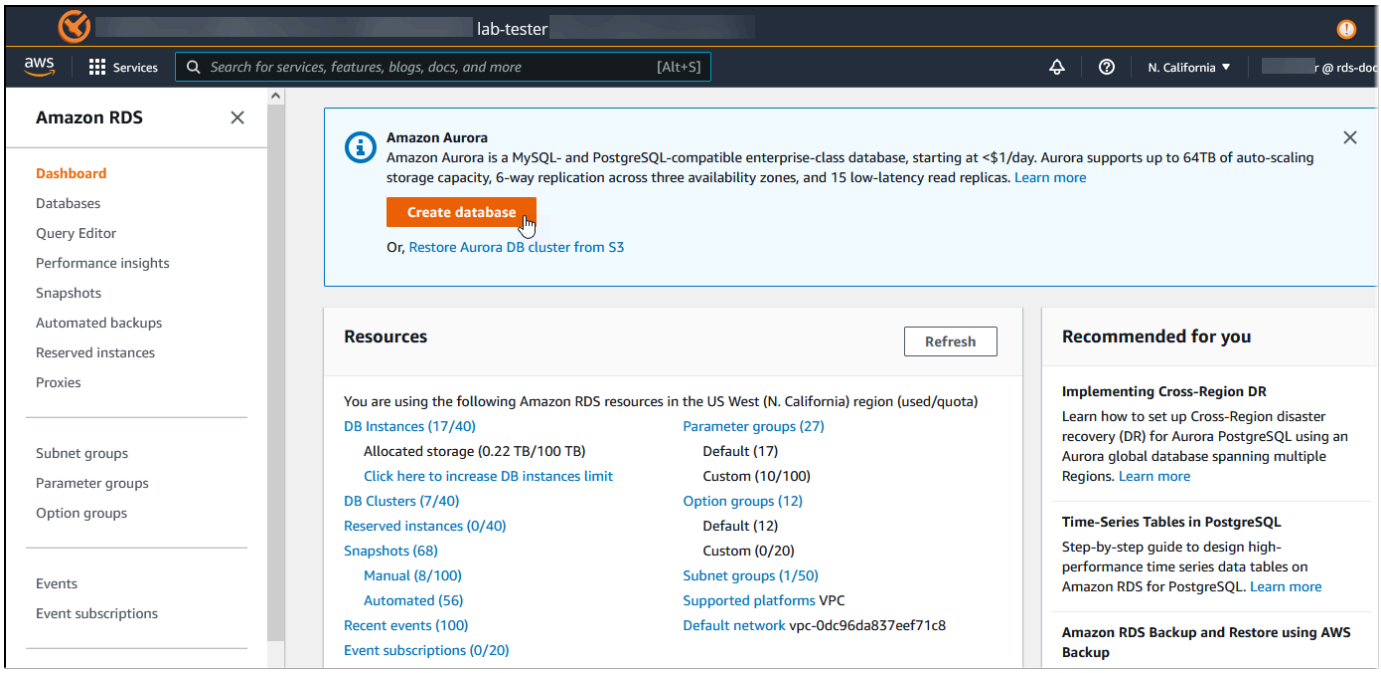
您可以通过 AWS Management Console 或 AWS CLI，使用 Babelfish 创建 Aurora PostgreSQL 集群。

> **Note**
>
> 在 Aurora PostgreSQL 集群中，系统会为 Babelfish 预留 `babelfish_db` 数据库名称。在 Babelfish for Aurora PostgreSQL 上自行创建名为 babelfish_db 的数据库会阻止 Aurora 成功预置 Babelfish。

控制台

在使用 AWS Management Console 运行 Babelfish 的情况下创建一个集群

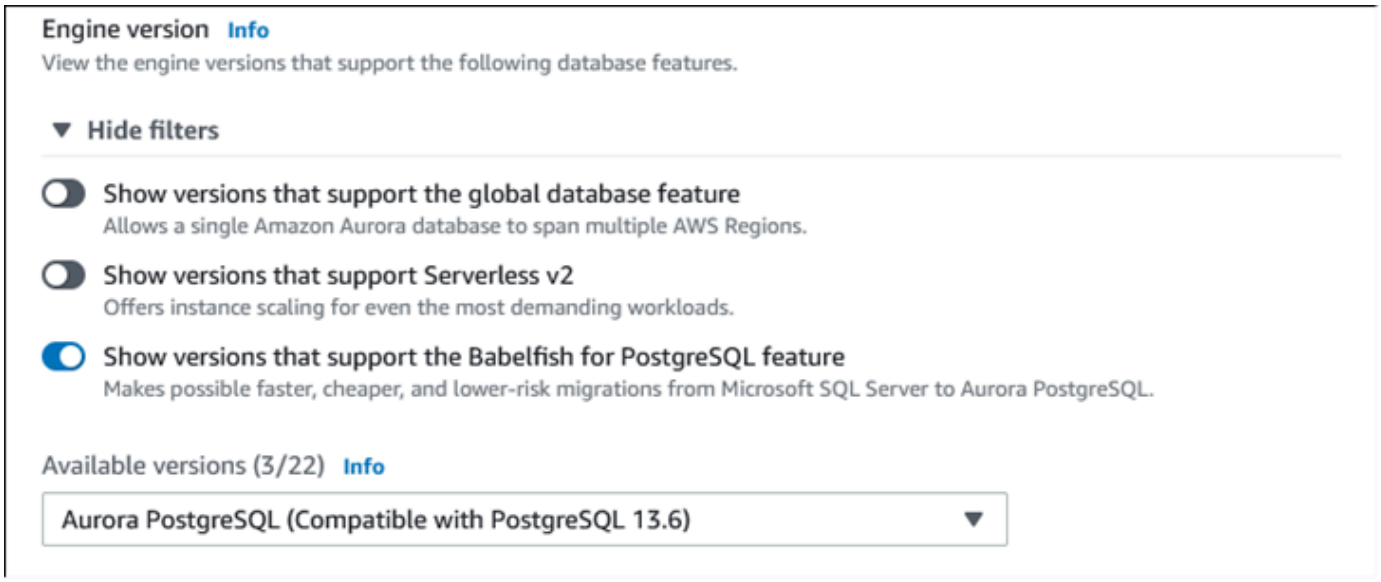
1. 从 https://console.aws.amazon.com/rds/ 打开 Amazon RDS 控制台，然后选择 Create database（创建数据库）。



2. 对于选择数据库创建方法，请执行以下操作之一：

   - 要指定详细的引擎选项，请选择 Standard create（标准创建）。
   - 要使用支持 Aurora 集群最佳实践的预配置选项，请选择 Easy create（轻松创建）。

3. 对于引擎类型，选择 Aurora（PostgreSQL 兼容）。
4. 选择 Show filters（显示筛选条件），然后选择 Show versions that support the Babelfish for PostgreSQL feature（显示支持 PostgreSQL 的 Babelfish 功能的版本）列出支持 Babelfish 的引擎类型。Aurora PostgreSQL 13.4 及更高版本目前支持 Babelfish。
5. 对于可用的版本，选择一个 Aurora PostgreSQL 版本。要获得最新的 Babelfish 功能，请选择最高的 Aurora PostgreSQL 主要版本。



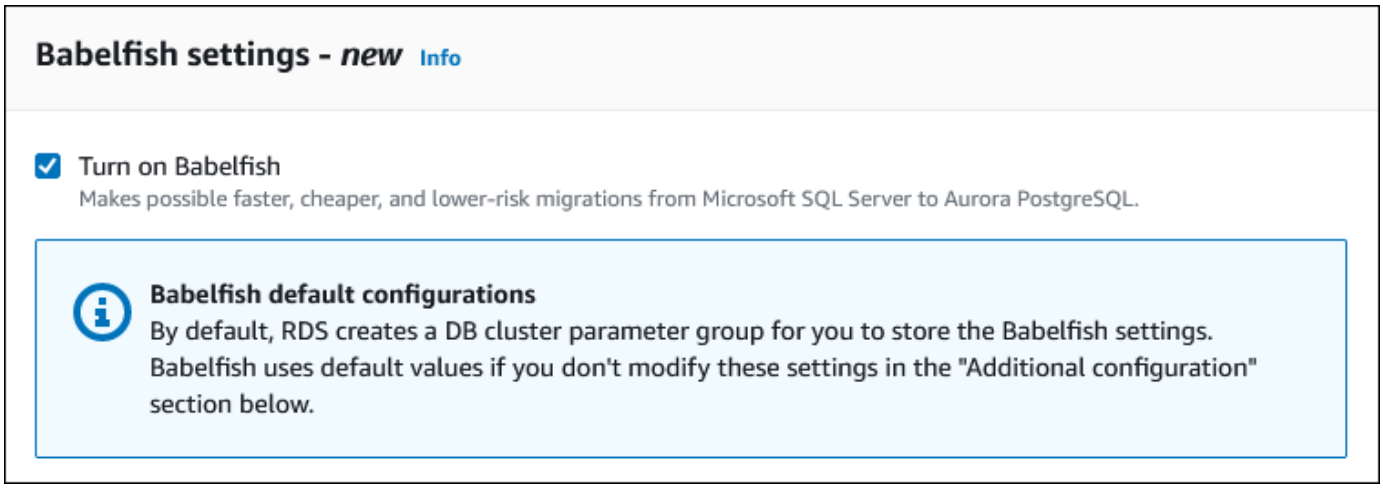
6. 对于 Templates（模板），选择与您的使用案例匹配的模板。
7. 对于数据库集群标识符，输入稍后可以在数据库集群列表中轻松找到的名称。
8. 对于主用户名中，输入管理员用户名。Aurora PostgreSQL 的原定设置值为 `postgres`。您可以接受原定设置，也可以选择其他名称。例如，要遵循 SQL Server 数据库中使用的命名约定，可以为主用户名输入 `sa`（系统管理员）。

   如果您在此时尚未创建名为 `sa` 的用户，您可以稍后用自己选择的客户端创建一个。创建用户后，使用 `ALTER SERVER ROLE` 命令将其添加到集群的 `sysadmin` 组（角色）。

   > **Warning**
   >
   > 主用户名必须始终使用小写字符，否则数据库集群将无法通过 TDS 端口连接到 Babelfish。

9. 对于 Master password（主密码），创建一个强密码并确认密码。
10. 对于接下来的选项，指定数据库集群设置，直到 Babelfish 设置部分。有关每项设置的信息，请参阅 Aurora 数据库集群的设置。

11. 要使 Babelfish 功能可用，请选择 Turn on Babelfish（打开 Babelfish）框。



12. 对于数据库集群参数组，请执行以下操作之一：

    - 选择 Create new（新建）以在打开 Babelfish 的情况下创建新的参数组。
    - 选择 Choose existing（选择现有）以使用现有的参数组。如果您使用现有组，请确保在创建集群之前修改组并为 Babelfish 参数添加值。有关 Babelfish 参数的信息，请参阅 Babelfish 的数据库集群参数组设置。

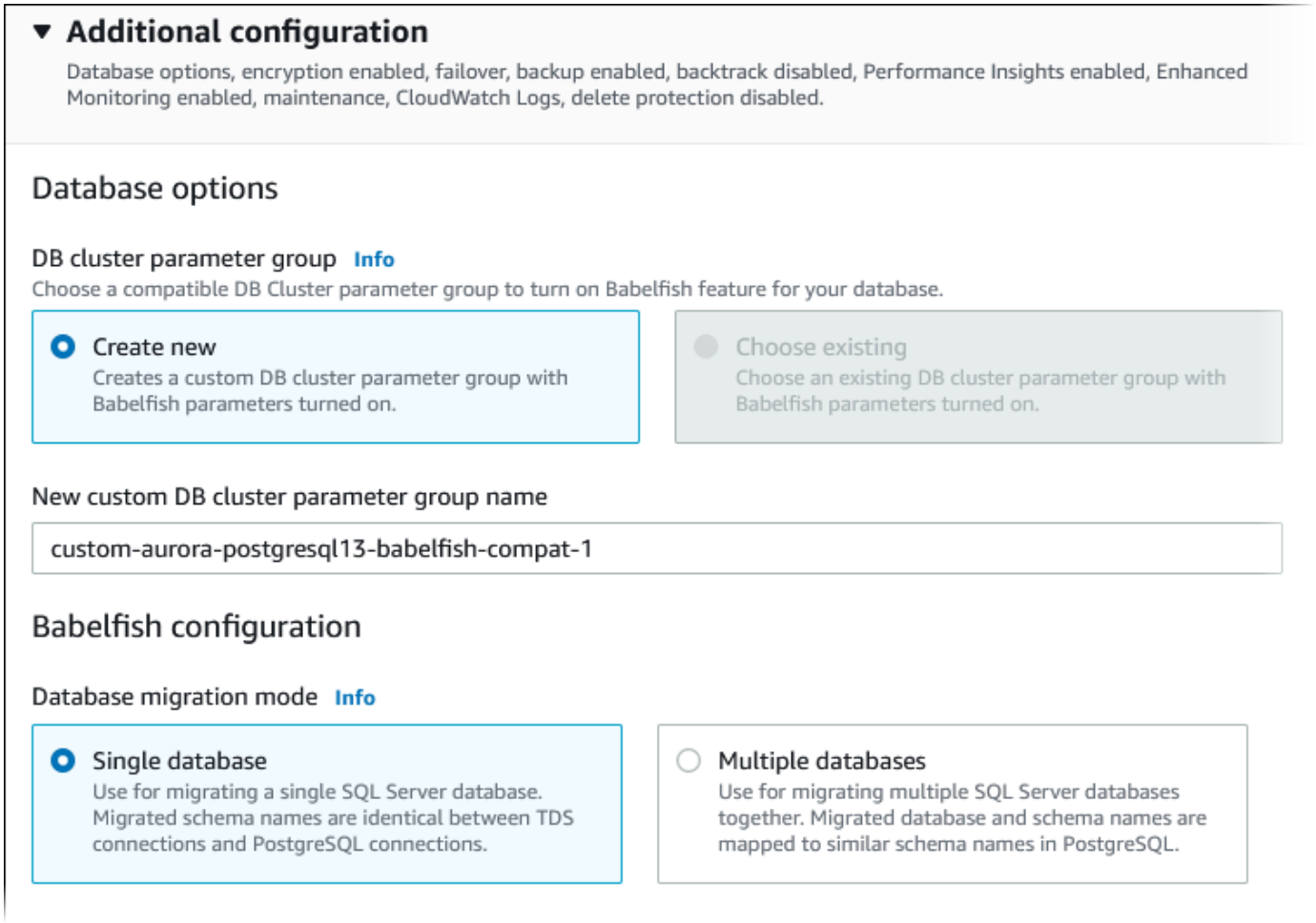
      如果您使用现有组，请在后面的框中提供组名称。

13. 对于 Database migration mode（数据库迁移模式），请选择下列选项之一：

    - 单个数据库，可迁移单个 SQL Server 数据库。

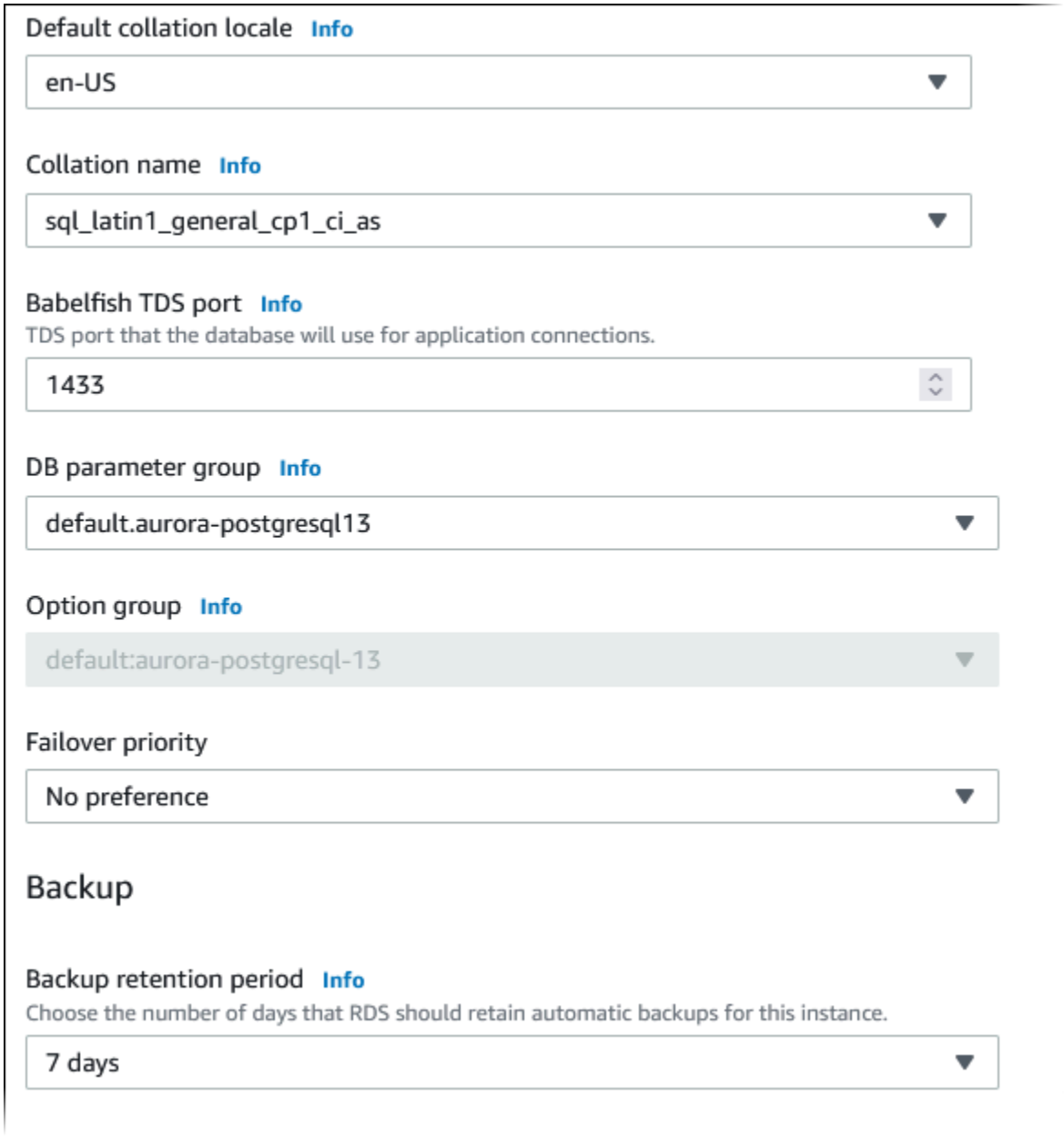
      在某些情况下，您可能会同时迁移多个用户数据库，最终目标是在没有 Babelfish 的情况下完全迁移到本机 Aurora PostgreSQL。如果最终的应用程序需要整合架构（单个 dbo 架构），请确保首先将 SQL Server 数据库整合到单个 SQL Server 数据库中。然后使用单个数据库模式迁移到 Babelfish。
    - 多个数据库，迁移多个 SQL Server 数据库（源自单个 SQL Server 安装）。多数据库模式不会合并不源于单个 SQL Server 安装的多个数据库。有关迁移多个数据库的信息，请参阅 将 Babelfish 与单个数据库或多个数据库结合使用。

> **Note**
>
> 从 Aurora PostgreSQL 16 版本起，默认选择多数据库作为数据库迁移模式。



14. 对于 Default collation locale（原定设置排序规则区域设置），输入您的服务器区域设置。默认值为 `en-US`。有关排序规则的详细信息，请参阅 了解适用于 Aurora PostgreSQL 的 Babelfish 中的排序规则。
15. 对于 Collation name（排序规则名称）字段，输入原定设置排序规则。默认值为 `sql_latin1_general_cp1_ci_as`。有关详细信息，请参阅 了解适用于 Aurora PostgreSQL 的 Babelfish 中的排序规则。
16. 对于 Babelfish TDS 端口，输入原定设置端口 `1433`。目前，Babelfish 仅对于您的数据库集群支持 `1433` 端口。
17. 对于数据库参数组，选择一个参数组或使用原定设置使用 Aurora 为您创建一个新组。
18. 对于 Failover priority（故障转移优先级），选择实例的故障转移优先级。如果您未选择值，则原定设置值为 `tier-1`。此优先级决定在主实例故障恢复时提升 副本的顺序。有关更多信息，请参阅 Aurora 数据库集群的容错能力。

19. 对于 Backup retention period（备份保留期），选择 Aurora 保留数据库的备份副本的时间长度（1–35 天）。您可使用备份副本对数据库执行时间点还原 (PITR)，精度可到秒。原定设置的保留期为七天。



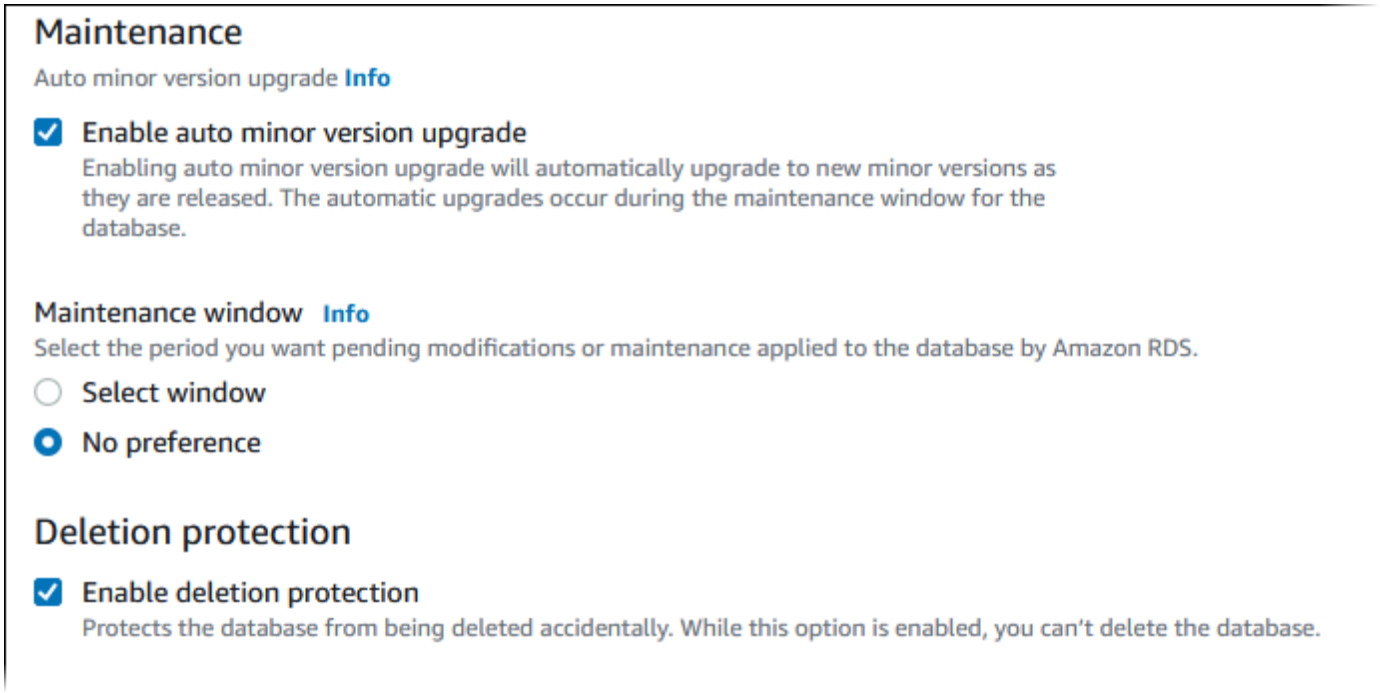
20. 选择 Copy tags to snapshots（将标签复制到快照）以在创建快照时将任何数据库实例标签复制到数据库快照。

> **Note**
>
> 当从快照还原数据库集群时，该集群不会还原为适用于 Aurora PostgreSQL 的 Babelfish 数据库集群。您需要开启用于控制数据库集群参数组中 Babelfish 首选项的参数，才能再

次启用 Babelfish。有关 Babelfish 参数的更多信息，请参阅 Babelfish 的数据库集群参数组设置。

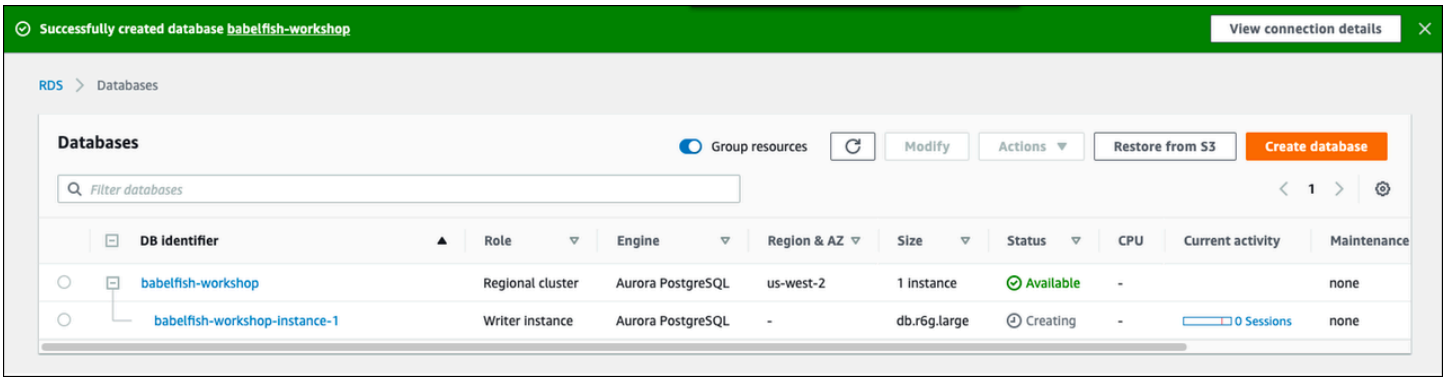
21. 请选择 Enable encryption（启用加密）以对该数据库集群开启静态加密（Aurora 存储加密）。
22. 选择 Enable Performance Insights（启用性能详情）打开 Amazon RDS 性能详情。
23. 选择 Enable Enhanced monitoring（启用增强监控）以开始您的数据库集群在其上运行的操作系统的实时指标收集。
24. 选择 PostgreSQL 日志将日志文件发布到 Amazon CloudWatch Logs。
25. 选择 Enable auto minor version upgrade（启用自动次要版本升级）在次要版本升级可用时自动更新 Aurora 数据库集群。
26. 对于维护时段，请执行以下操作：
    - 要选择 Amazon RDS 进行修改或执行维护的时间，请选择 Select window（选择时段）。
    - 要在计划外的时间执行 Amazon RDS 维护，请选择 No preference（无首选项）。
27. 选择 Enable deletion protection（启用删除保护）框以保护您的数据库免于意外删除。

    如果启用此功能，则无法直接删除数据库。相反，在删除数据库之前，您需要修改数据库集群并关闭此功能。



28. 选择创建数据库。

您可以在 Databases（数据库）列表中找到为 Babelfish 设置的新数据库。Status（状态）列在部署完成时显示 Available（可用）。



AWS CLI

当您使用 AWS CLI 创建 Babelfish for Aurora PostgreSQL 时，您需要向命令传递要用于集群的数据库集群参数组的名称。有关更多信息，请参阅 数据库集群先决条件。

在您可以使用 AWS CLI 创建具有 Babelfish 的 Aurora PostgreSQL 集群之前，请执行以下操作：

- 在 Amazon Aurora 终端节点和配额中，从服务列表中选择端点 URL。
- 创建集群的参数组。有关参数组的更多信息，请参阅 Amazon Aurora 的参数组。
- 修改参数组，添加打开 Babelfish 的参数。

要使用 AWS CLI 创建带有 Babelfish 的 Aurora PostgreSQL 数据库集群

下面的示例使用原定设置的主用户名 `postgres`。根据需要替换您为数据库集群创建的用户名（例如 `sa`），或当您不接受原定设置时选择的任何用户名。

1. 创建参数组。

   对于 Linux、macOS 或 Unix：

```
aws rds create-db-cluster-parameter-group \
--endpoint-url endpoint-url \
--db-cluster-parameter-group-name parameter-group \
--db-parameter-group-family aurora-postgresql14 \
--description "description"
```

   对于 Windows：

```
aws rds create-db-cluster-parameter-group ^
--endpoint-url endpoint-URL ^
--db-cluster-parameter-group-name parameter-group ^
--db-parameter-group-family aurora-postgresql14 ^
--description "description"
```

2. 修改参数组以打开 Babelfish。

   对于 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster-parameter-group \
--endpoint-url endpoint-url \
--db-cluster-parameter-group-name parameter-group \
--parameters
 "ParameterName=rds.babelfish_status,ParameterValue=on,ApplyMethod=pending-reboot"
```

   对于 Windows：

```
aws rds modify-db-cluster-parameter-group ^
--endpoint-url endpoint-url ^
--db-cluster-parameter-group-name paramater-group ^
--parameters
 "ParameterName=rds.babelfish_status,ParameterValue=on,ApplyMethod=pending-reboot"
```

3. 为新的数据库集群指定数据库子网组和 Virtual Private Cloud (VPC) 安全组 ID，然后调用 create-db-cluster 命令。

   对于 Linux、macOS 或 Unix：

```
aws rds create-db-cluster \
--db-cluster-identifier  cluster-name\
--master-username postgres \
--manage-master-user-password \
--engine aurora-postgresql \
--engine-version 14.3            \
--vpc-security-group-ids security-group \
--db-subnet-group-name subnet-group-name \
--db-cluster-parameter-group-name parameter-group
```

   对于 Windows：

```
aws rds create-db-cluster ^
--db-cluster-identifier cluster-name ^
--master-username postgres ^
--manage-master-user-password ^
--engine aurora-postgresql ^
--engine-version 14.3 ^
--vpc-security-group-ids security-group ^
--db-subnet-group-name subnet-group ^
--db-cluster-parameter-group-name parameter-group
```

此示例指定了生成主用户密码并在 Secrets Manager 中对其进行管理的 `--manage-master-user-password` 选项。有关更多信息，请参阅 使用 Amazon Aurora 和 AWS Secrets Manager 管理密码。或者，您可以使用 `--master-password` 选项自行指定和管理密码。

4. 显式为数据库集群主实例。调用 create-db-instance 命令时，将您在步骤 3 中创建的集群名称用于 `--db-cluster-identifier` 参数，如下所示。

对于 Linux、macOS 或 Unix：

```
aws rds create-db-instance \
--db-instance-identifier instance-name \
--db-instance-class db.r6g \
--db-subnet-group-name subnet-group \
--db-cluster-identifier cluster-name \
--engine aurora-postgresql
```

对于 Windows：

```
aws rds create-db-instance ^
--db-instance-identifier instance-name ^
--db-instance-class db.r6g ^
--db-subnet-group-name subnet-group ^
--db-cluster-identifier cluster-name ^
--engine aurora-postgresql
```

将 SQL Server 数据库迁移到 Babelfish for Aurora PostgreSQL

可以使用 Babelfish for Aurora PostgreSQL 将 SQL Server 数据库迁移到 Amazon Aurora PostgreSQL 数据库集群。在迁移之前，请查看 将 Babelfish 与单个数据库或多个数据库结合使用。

主题

- 迁移过程概述
- 评估和处理 SQL Server 和 Babelfish 之间的差异
- 用于从 SQL Server 迁移到 Babelfish 的导入/导出工具

迁移过程概述

以下摘要列出了成功迁移 SQL Server 应用程序和使之与 Babelfish 配合使用所需的步骤：有关可用于导出和导入流程的工具的信息以及更多详细信息，请参阅用于从 SQL Server 迁移到 Babelfish 的导入/导出工具。要加载数据，我们建议将 AWS DMS 与 Aurora PostgreSQL 数据库集群结合用作目标端点。

1. 在开启 Babelfish 的情况下创建新的 Aurora PostgreSQL 数据库集群。要了解如何操作，请参阅创建 Babelfish for Aurora PostgreSQL 数据库集群。

   要导入从 SQL Server 数据库导出的各种 SQL 构件，请使用 SQL Server 工具（例如 sqlcmd）连接到 Babelfish 集群。有关更多信息，请参阅 使用 SQL Server 客户端连接到数据库集群。

2. 在要迁移的 SQL Server 数据库上，导出数据定义语言 (DDL)。DDL 是一种 SQL 代码，用于描述包含用户数据（如表、索引和视图）和用户编写的数据库代码（例如存储过程、用户定义的函数和触发器）的数据库对象。

   有关更多信息，请参阅 使用 SQL Server Management Studio (SSMS) 迁移到 Babelfish。

3. 运行评估工具来评估可能需要进行的任何更改的范围，以便 Babelfish 能够有效地支持在 SQL Server 上运行的应用程序。有关更多信息，请参阅 评估和处理 SQL Server 和 Babelfish 之间的差异。

4. 查看 AWS DMS 目标端点限制并根据需要更新 DDL 脚本。有关更多信息，请参阅 Using Babelfish for Aurora PostgreSQL as a target 中的“Limitations to using a PostgreSQL target endpoint with Babelfish tables”。

5. 在新的 Babelfish 数据库集群上，在指定的 T-SQL 数据库中运行 DDL，以仅创建具有主键约束条件的模式、用户定义的数据类型和表。

6. 使用 AWS DMS 将数据从 SQL Server 迁移到 Babelfish 表。要使用 SQL Server 变更数据捕获或 SQL 复制进行持续复制，请使用 Aurora PostgreSQL 而不是 Babelfish 作为端点。为此，请参阅使用适用于 Aurora PostgreSQL 的 Babelfish 作为 AWS Database Migration Service 的目标。
7. 数据加载完成后，在 Babelfish 集群上创建支持应用程序的所有剩余 T-SQL 对象。
8. 重新配置客户端应用程序以连接到 Babelfish 端点而不是 SQL Server 数据库。有关更多信息，请参阅 连接到 Babelfish 数据库集群。
9. 根据需要修改应用程序并重新测试。有关更多信息，请参阅 适用于 Aurora PostgreSQL 的 Babelfish 与 SQL Server 之间的区别。

您仍然需要评估客户端 SQL 查询。从 SQL Server 实例生成的架构仅转换服务器端 SQL 代码。我们建议您采取以下步骤：

- 通过将 SQL Server Profiler 与 TSQL_Replay 预定义模板结合使用来捕获客户端查询。此模板捕获 T-SQL 语句信息，然后可以重播这些信息以进行迭代优化和测试。您可以从 SQL Server Management Studio 的 Tools（工具）菜单中启动此 Profiler。选择 SQL Server Profiler 以打开此 Profiler 并选择 TSQL_Replay 模板。

  要用于 Babelfish 迁移，请开始跟踪，然后使用功能测试运行应用程序。此 Profiler 捕获 T-SQL 语句。当您完成测试时，停止跟踪。使用客户端查询将结果保存到 XML 文件中 [File（文件）> Save as（另存为）> Trace XML File for Replay（跟踪 XML 文件以便重播）]。

  有关更多信息，请参阅 Microsoft 文档中的 SQL Server Profiler。有关 TSQL_Replay 模板的更多信息，请参阅 SQL Server Profiler 模板。
- 对于具有复杂客户端 SQL 查询的应用程序，我们建议您使用 Babelfish Compass 分析这些查询以了解 Babelfish 兼容性。如果分析表明客户端 SQL 语句包含不受支持的 SQL 功能，请查看客户端应用程序中的 SQL 方面，并根据需要进行修改。
- 您也可以将 SQL 查询捕获为扩展事件（.xel 格式）。为此，请使用 SSMS XEvent Profiler。生成 .xel 文件后，将 SQL 语句提取到 .xml 文件中，然后 Compass 可以对其进行处理。有关更多信息，请参阅 Microsoft 文档中的 使用 SSMS XEvent Profler。

当您对迁移的应用程序所需的所有测试、分析以及任何修改感到满意时，您可以开始将 Babelfish 数据库用于生产环境中。为此，请停止原始数据库，并重新导向实时客户端应用程序以使用 Babelfish TDS 端口。

> **Note**
>
> AWS DMS 现在支持从 Babelfish 复制数据。有关更多信息，请参阅 AWS DMS now supports Babelfish for Aurora PostgreSQL as a source。

评估和处理 SQL Server 和 Babelfish 之间的差异

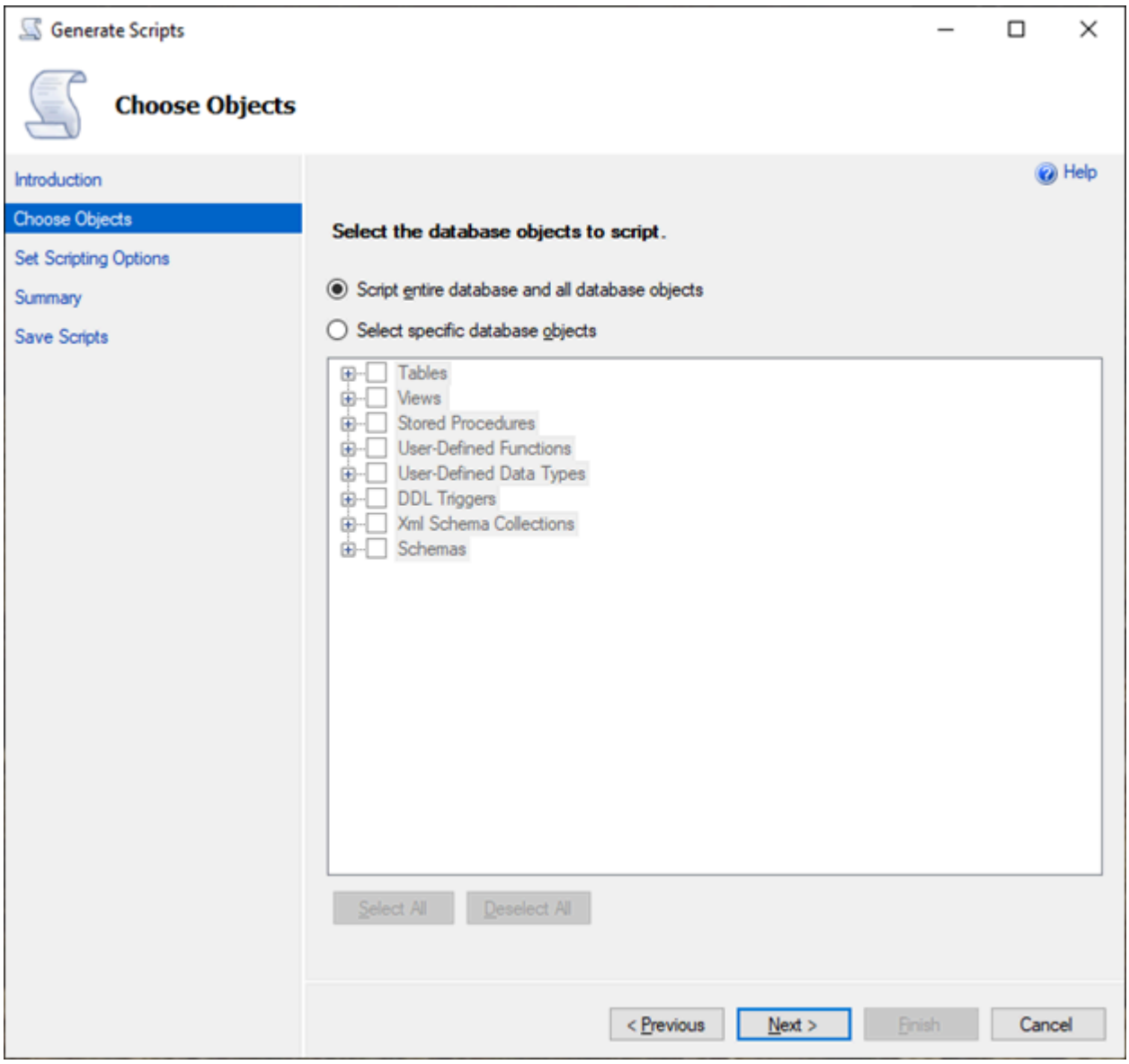
为了获得最佳结果，我们建议您在将 SQL Server 数据库应用程序实际迁移到 Babelfish 之前，先评估生成的 DDL/DML 和客户端查询代码。根据 Babelfish 的版本和应用程序实现的 SQL Server 的特定功能，您可能需要重构应用程序或使用替代方案来获得 Babelfish 中尚未完全支持的功能。

- 要评估 SQL Server 应用程序代码，请在生成的 DDL 上使用 Babelfish Compass 来确定 Babelfish 支持多少个 T-SQL 代码。确定在 Babelfish 上运行之前可能需要修改的 T-SQL 代码。有关此工具的更多信息，请参阅 GitHub 上的 Babelfish Compass 工具。

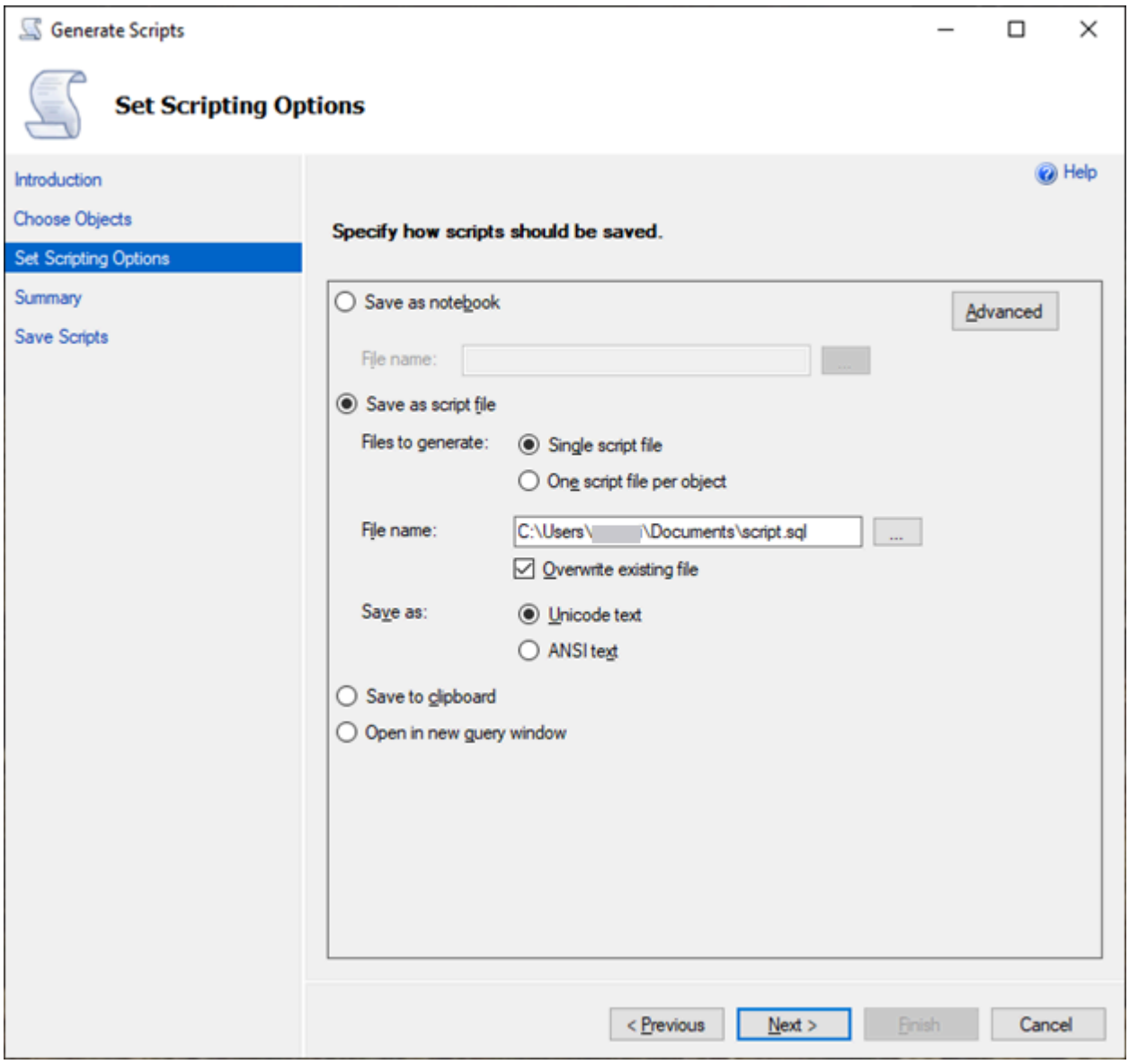
  > **Note**
  >
  > Babelfish Compass 是一种开源工具。请通过 GitHub 而不是 AWS Support 报告 Babelfish Compass 的任何问题。

您可以在 SQL Server Management Studio（SSMS）中使用 Generate Script Wizard（生成脚本向导）来生成由 Babelfish Compass 或 AWS Schema Conversion Tool CLI 评估的 SQL 文件。我们建议采取以下步骤来简化评估。

1. 在 Choose Objects（选择对象）页面上，选择 Script entire database and all database objects（为整个数据库和所有数据库对象编写脚本）。

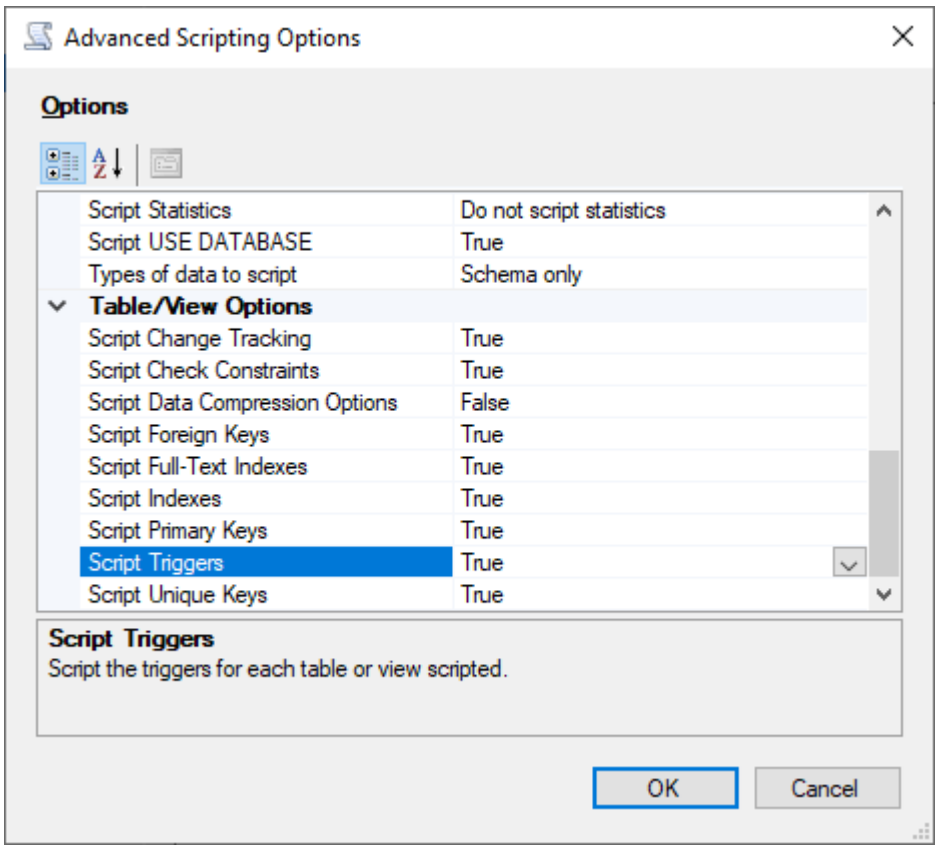


2. 对于 Set Scripting Options（设置脚本选项），选择 Save as script file（另存为脚本文件）作为 Single script file（单个脚本文件）。



3. 选择 Advanced（高级）可更改原定设置脚本选项，以识别在进行全面评估时通常设置为 false 的功能：

   - 脚本将跟踪更改为 True
   - 脚本将全文索引设置为 True
   - 脚本将触发器设置为 True
   - 脚本将登录设置为 True
   - 脚本将拥有者设置为 True
   - 脚本将对象级权限设置为 True
   - 脚本将排序规则设置为 True



4. 执行向导中的剩余步骤来生成文件。

用于从 SQL Server 迁移到 Babelfish 的导入/导出工具

我们建议使用 AWS DMS 作为从 SQL Server 迁移到 Babelfish 的主要工具。但是，Babelfish 支持其他几种使用 SQL Server 工具迁移数据的方式，其中包括以下各项。

- 适用于所有版本的 Babelfish 的 SQL Server Integration Services（SSIS）。有关更多信息，请参阅使用 SSIS 和 Babelfish 从 SQL Server 迁移到 Aurora PostgreSQL。
- 使用适用于 Babelfish 版本 2.1.0 及更高版本的 SSMS 导入/导出向导。此工具可通过 SSMS 获得，但也可作为独立工具使用。有关更多信息，请参阅 Microsoft 文档中的欢迎使用 SQL Server 导入和导出向导。
- Microsoft 批量数据复制程序（bcp）实用程序可让您以您指定的格式将数据从 Microsoft SQL Server 实例复制到数据文件中。有关更多信息，请参阅 Microsoft 文档中的 bcp 实用程序。Babelfish 现在支持使用 BCP 客户端进行数据迁移，而 bcp 实用程序现在支持 `-E` 标志（用于身份列）和 `-b` 标志（用于批量插入）。不支持某些 bcp 选项，包括 `-C`、`-T`、`-G`、`-K`、`-R`、`-V` 和 `-h`。

使用 SQL Server Management Studio (SSMS) 迁移到 Babelfish

我们建议为每种特定的对象类型生成单独的文件。您可以先使用 SSMS 中的 Generate Scripts（生成脚本）向导来处理每组 DDL 语句，然后将对象按组进行修改，以修复评估期间发现的任何问题。

执行这些步骤，使用 AWS DMS 或其他数据迁移方法迁移数据。请先运行这些创建脚本类型，以便更好、更快地在 Aurora PostgreSQL 中的 Babelfish 表上加载数据。

1. 运行 `CREATE SCHEMA` 语句。
2. 运行 `CREATE TYPE` 语句以创建用户定义的数据类型。
3. 使用主键或唯一约束运行基本 `CREATE TABLE` 语句。

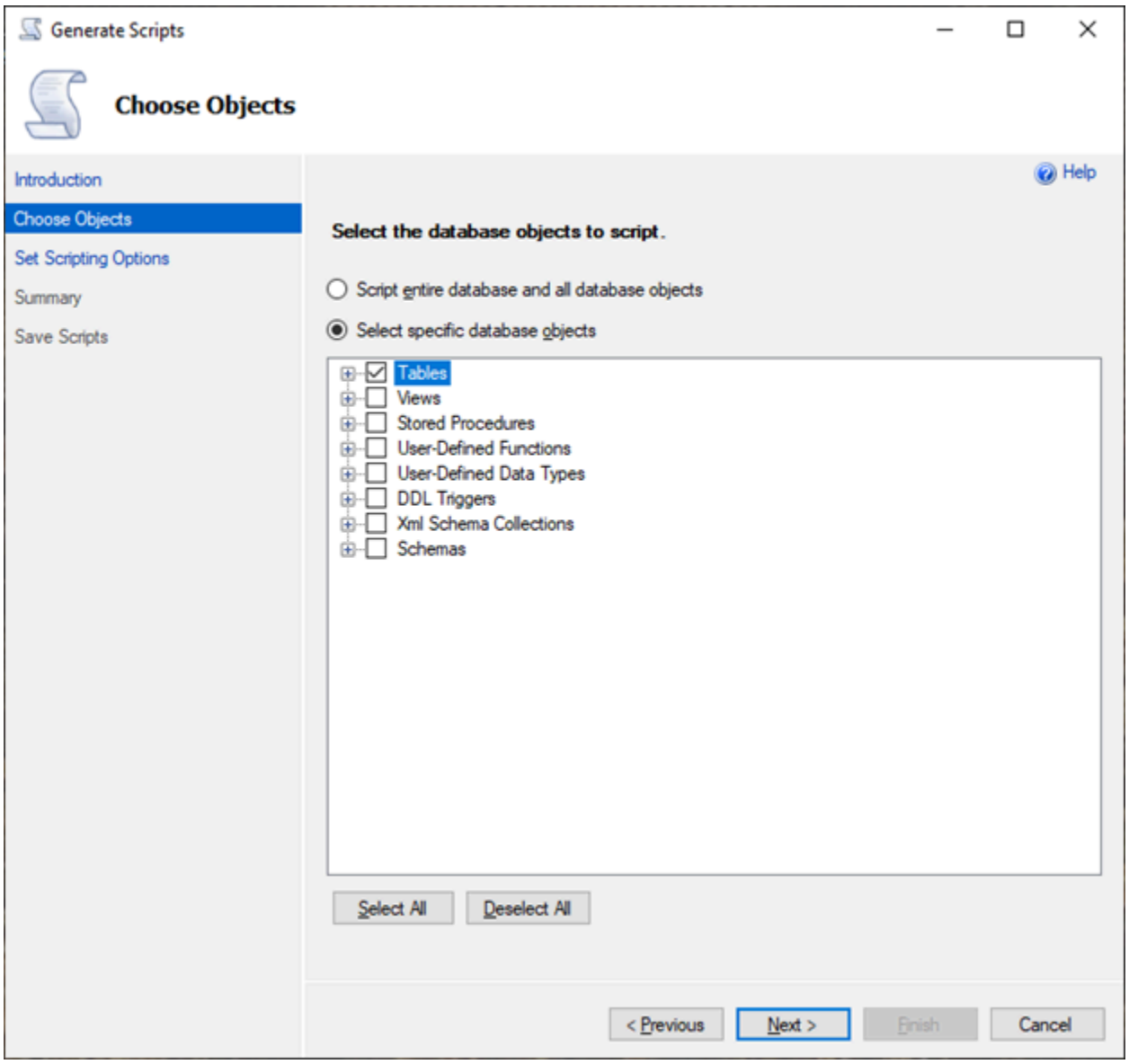
使用建议的导入/导出工具执行数据加载。运行以下步骤的修改后脚本以添加剩余的数据库对象。您需要创建表语句来运行这些脚本，以获取约束、触发器和索引。脚本生成后，删除创建表语句。

1. 针对检查约束、外键约束、原定设置约束运行 `ALTER TABLE` 语句。
2. 运行 `CREATE TRIGGER` 语句。
3. 运行 `CREATE INDEX` 语句。
4. 运行 `CREATE VIEW` 语句。
5. 运行 `CREATE STORED PROCEDURE` 语句。

为每种对象类型生成脚本

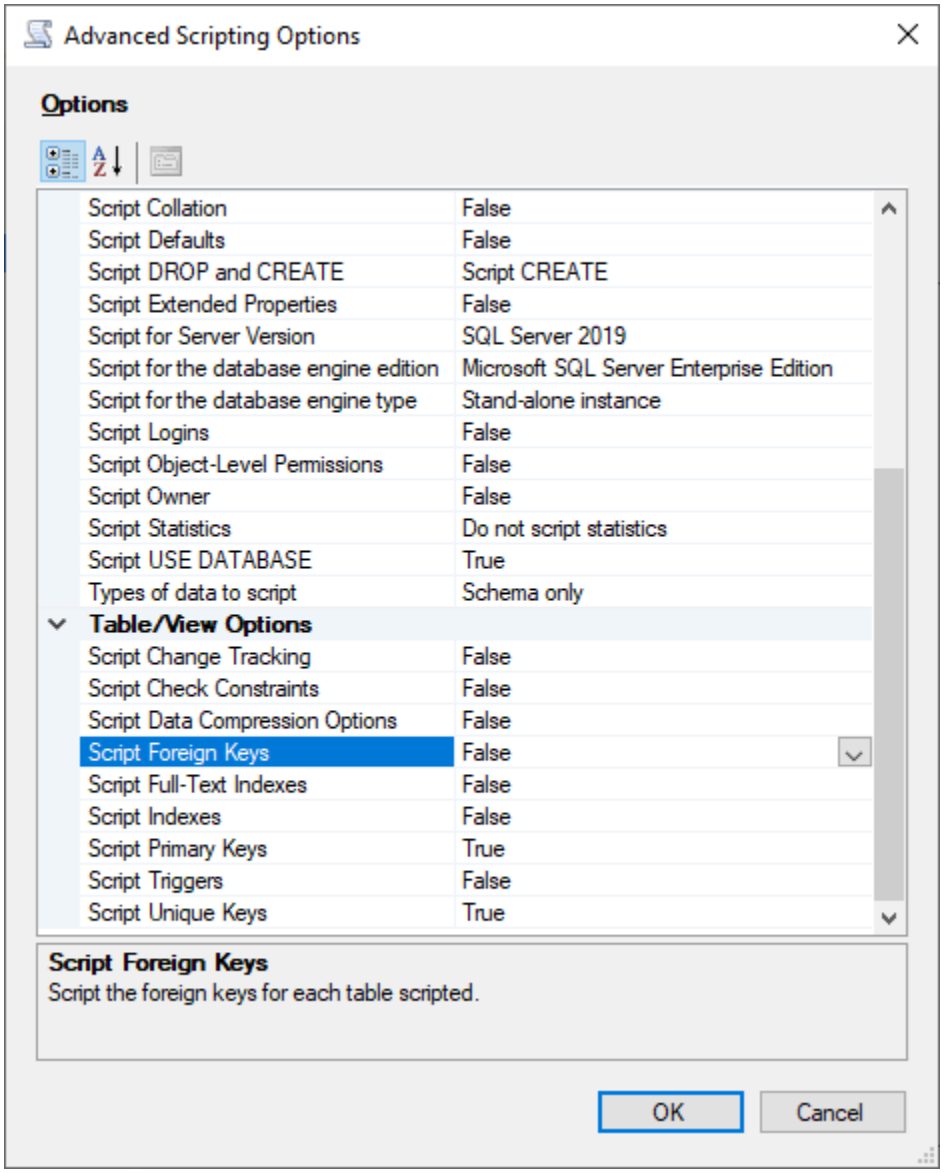
使用以下步骤通过 SSMS 中的生成脚本向导来创建基本的创建表语句。按照同样的步骤为不同的对象类型生成脚本。

1. 连接到您的现有 SQL Server 实例。
2. 打开数据库名称的上下文（右键单击）菜单。
3. 选择 Tasks（任务），然后选择 Generate Scripts...（生成脚本...）。
4. 在 Choose Objects（选择对象）窗格中，选择 Select specific database objects（选择特定数据库对象）。选择 Tables（表），然后选择所有表。选择下一步以继续。

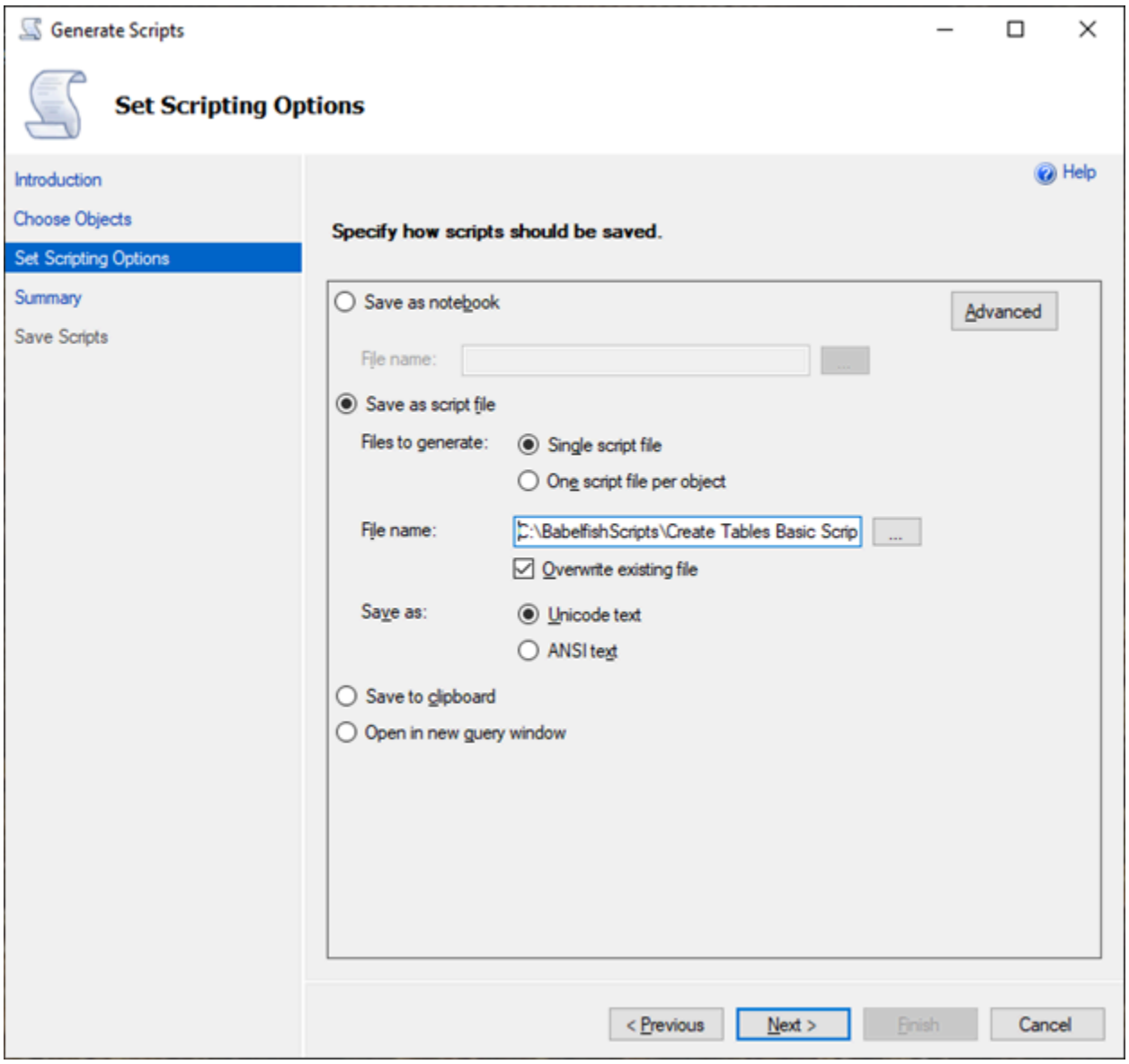


5. 在 Set Scripting Options（设置脚本选项）页面上，选择 Advanced（高级）以打开 Options（选项）设置。要生成基本的创建表语句，请更改以下原定设置值：

   - 脚本将原定设置设为 False。
   - 脚本将扩展属性设置为 False。Babelfish 不支持扩展属性。
   - 脚本将检查约束设置为 False。脚本将外键设置为 False。



6. 选择确定。
7. 在 Set Scripting Options（设置脚本选项）页上，选择 Save as script file（另存为脚本文件），然后选择 Single script file（单个脚本文件）选项。输入 File name（文件名）。



8. 选择 Next（下一步）以查看 Summary wizard（摘要向导）页。
9. 选择 Next（下一步）以启动脚本生成。

   您可以继续在向导中为其他对象类型生成脚本。不是在保存文件后选择 Finish（完成），而是选择 Previous（上一步）按钮三次以返回 Choose Objects（选择对象）页。然后重复向导中的步骤，为其他对象类型生成脚本。

适用于 Aurora PostgreSQL 的 Babelfish 的数据库身份验证

适用于 Aurora PostgreSQL 的 Babelfish 支持两种对数据库用户进行身份验证的方法。原定设置情况下，密码身份验证适用于所有 Babelfish 数据库集群。您还可以为同一个数据库集群添加 Kerberos 身份验证。

主题

- 适用于 Babelfish 的密码身份验证
- 适用于 Babelfish 的 Kerberos 身份验证
- 使用 Active Directory 安全组为 Babelfish 设置 Kerberos 身份验证

适用于 Babelfish 的密码身份验证

Babelfish for Aurora PostgreSQL 支持密码身份验证。密码以加密形式存储在磁盘上。有关 Aurora PostgreSQL 集群上的身份验证的更多信息，请参阅 使用 Amazon Aurora PostgreSQL 实现高安全性。

每次连接到 Babelfish 时，系统可能会提示您输入凭证。迁移到 Aurora PostgreSQL 或在 Aurora PostgreSQL 上创建的任何用户都可以在 SQL Server 端口和 PostgreSQL 端口上使用相同的凭证。Babelfish 不强制执行密码策略，但我们建议您执行以下操作：

- 需要一个复杂密码，该密码长度至少为八 (8) 个字符。
- 强制执行密码过期策略。

要查看数据库用户的完整列表，请使用命令 `SELECT * FROM pg_user;`。

适用于 Babelfish 的 Kerberos 身份验证

适用于 Aurora PostgreSQL 的 Babelfish 15.2 版本支持使用 Kerberos 对数据库集群进行身份验证。当用户连接到 Babelfish 数据库时，这种方法可让您使用 Microsoft Windows 身份验证来验证用户身份。为此，您必须先将数据库集群配置为使用 AWS Directory Service for Microsoft Active Directory 进行 Kerberos 身份验证。有关更多信息，请参阅《AWS Directory Service 管理员指南》中的什么是 AWS Directory Service？。

设置 Kerberos 身份验证

适用于 Aurora PostgreSQL 的 Babelfish 数据库集群可以使用两个不同的端口进行连接，但 Kerberos 身份验证设置是一个一次性过程。因此，您必须先为数据库集群设置 Kerberos 身份验证。有关更多信息，请参阅设置 Kerberos 身份验证。完成设置后，确保您可以使用 Kerberos 连接到 PostgreSQL 客户端。有关更多信息，请参阅使用 Kerberos 身份验证进行连接。

Babelfish 中的登录信息和用户预调配

通过表格式数据流（TDS）端口创建的 Windows 登录信息可以与 TDS 端口或 PostgreSQL 端口一起使用。首先，可以使用 Kerberos 进行身份验证的登录信息必须先从 TDS 端口进行预调配，然后 T-SQL 用户和应用程序才可以使用此登录信息连接到 Babelfish 数据库。创建 Windows 登录信息时，管理员可以使用 DNS 域名或 NetBIOS 域名提供登录信息。通常，NetBIOS 域是 DNS 域名的子域。例如，如果 DNS 域名为 `CORP.EXAMPLE.COM`，则 NetBIOS 域可以是 `CORP`。如果为登录信息提供了 NetBIOS 域名格式，则必须存在与 DNS 域名的映射。

管理 NetBIOS 域名到 DNS 域名的映射

为了管理 NetBIOS 域名和 DNS 域名之间的映射，Babelfish 提供了用于添加、删除和截断映射的系统存储过程。只有具有 `sysadmin` 角色的用户才能运行这些过程。

要在 NetBIOS 和 DNS 域名之间创建映射，请使用 Babelfish 提供的系统存储过程 `babelfish_add_domain_mapping_entry`。这两个参数都必须具有有效值而不是 NULL。

Example

```
EXEC babelfish_add_domain_mapping_entry 'netbios_domain_name',
 'fully_qualified_domain_name'
```

以下示例显示了如何在 NetBIOS 名称 CORP 和 DNS 域名 CORP.EXAMPLE.COM 之间创建映射：

Example

```
EXEC babelfish_add_domain_mapping_entry 'corp', 'corp.example.com'
```

要删除现有的映射条目，请使用系统存储过程 babelfish_remove_domain_mapping_entry。

Example

```
EXEC babelfish_remove_domain_mapping_entry 'netbios_domain_name'
```

以下示例显示如何删除 NetBIOS 名称 CORP 的映射。

Example

```
EXEC babelfish_remove_domain_mapping_entry 'corp'
```

要删除所有现有的映射条目，请使用系统存储过程 babelfish_truncate_domain_mapping_table：

Example

```
EXEC babelfish_truncate_domain_mapping_table
```

要查看 NetBIOS 与 DNS 域名之间的所有映射，请使用以下查询。

Example

```
SELECT netbios_domain_name, fq_domain_name FROM babelfish_domain_mapping;
```

管理登录信息

创建登录信息

使用具有正确权限的登录信息通过 TDS 端点连接到数据库。如果没有为登录信息创建数据库用户，则该登录将映射到访客用户。如果未启用访客用户，则登录尝试将失败。

使用以下查询创建 Windows 登录信息。FROM WINDOWS 选项允许使用 Active Directory 进行身份验证。

```
CREATE LOGIN login_name FROM WINDOWS [WITH DEFAULT_DATABASE=database]
```

Example

以下示例显示了使用原定设置数据库 db1 为 Active Directory 用户 [corp\test1] 创建登录信息。

```
CREATE LOGIN [corp\test1] FROM WINDOWS WITH DEFAULT_DATABASE=db1
```

此示例假设 NetBIOS 域 CORP 和 DNS 域名 CORP.EXAMPLE.COM 之间存在映射。如果没有映射，则必须提供 DNS 域名 [CORP.EXAMPLE.COM\test1]。

> **Note**
>
> Active Directory 用户的登录名仅限于少于 21 个字符的名称。

删除登录信息

要删除登录信息，请使用与任何登录信息相同的语法，如以下示例所示：

```
DROP LOGIN [DNS domain name\login]
```

更改登录信息

要更改登录信息，请使用与任何登录信息相同的语法，如以下示例所示：

```
ALTER LOGIN [DNS domain name\login] { ENABLE|DISABLE|WITH DEFAULT_DATABASE=[master] }
```

ALTER LOGIN 命令对于 Windows 登录信息支持有限的选项，包括以下各项：

- DISABLE – 禁用登录信息。您不能使用禁用的登录信息进行身份验证。
- ENABLE - 启用禁用的登录信息。
- DEFAULT_DATABASE – 更改登录信息的原定设置数据库。

> **Note**
>
> 所有密码管理都是通过 AWS Directory Service 执行的，因此 ALTER LOGIN 命令不允许数据库管理员更改或设置 Windows 登录的密码。

使用 Kerberos 身份验证连接到适用于 Aurora PostgreSQL 的 Babelfish

通常，使用 Kerberos 进行身份验证的数据库用户是从客户端计算机进行身份验证的。这些计算机是 Active Directory 域的成员。这些用户使用客户端应用程序中的 Windows 身份验证在 TDS 端口上访问适用于 Aurora PostgreSQL 的 Babelfish 服务器。

使用 Kerberos 身份验证在 PostgreSQL 端口上连接到适用于 Aurora PostgreSQL 的 Babelfish

您可以将从 TDS 端口创建的登录信息与 TDS 端口或 PostgreSQL 端口一起使用。但是，PostgreSQL 原定设置情况下对用户名使用区分大小写的比较。要让 Aurora PostgreSQL 将 Kerberos 用户名解释为不区分大小写，您必须在自定义 Babelfish 集群参数组中将 `krb_caseins_users` 参数设置为 `true`。原定设置情况下，此参数设置为 `false`。有关更多信息，请参阅配置不区分大小写的用户名。此外，必须从 PostgreSQL 客户端应用程序中以 <登录名@DNS 域名> 格式指定登录用户名。您不能使用 <DNS 域名\登录名> 格式。

经常发生的错误

您无法在本地 Microsoft Active Directory 与 AWS Managed Microsoft AD 之间配置林信任关系。有关更多信息，请参阅创建信任关系。然后，您必须使用专门的域特定端点进行连接，而不是在主机端点中使用 Amazon 域 `rds.amazonaws.com`。如果您没有使用正确的域特定端点，您可能会遇到以下错误：

```
Error: “Authentication method "NTLMSSP" not supported (Microsoft SQL Server, Error:
 514)"
```

当 TDS 客户端无法缓存所提供的端点 URL 的服务票证时，就会出现此错误。有关更多信息，请参阅使用 Kerberos 进行连接。

使用 Active Directory 安全组为 Babelfish 设置 Kerberos 身份验证

从 Babelfish 版本 4.2.0 开始，可以使用 Active Directory 安全组为 Babelfish 设置 Kerberos 身份验证。以下是使用 Active Directory 设置 Kerberos 身份验证需要完成的先决条件：

- 必须按照适用于 Babelfish 的 Kerberos 身份验证中提到的所有步骤进行操作。
- 确保数据库实例与 Active Directory 关联。为验证这一点，您可以在控制台中或者通过运行 describe-db-instances AWS CLI 命令来查看域成员资格的状态。

  数据库实例的状态应为已启用 kerberos。有关了解域成员资格的更多信息，请参阅了解域成员资格。
- 使用以下查询验证 NetBIOS 域名和 DNS 域名之间的映射：

```
SELECT netbios_domain_name, fq_domain_name FROM babelfish_domain_mapping;
```

- 在继续操作之前，请验证使用单独登录名进行的 Kerberos 身份验证是否符合预期。使用以 Active Directory 用户身份进行 Kerberos 身份验证的连接应该会取得成功。如果您遇到任何问题，请参阅经常发生的错误。

设置 pg_ad_mapping 扩展

必须按照设置 pg_ad_mapping 扩展中提到的所有步骤进行操作。要验证扩展是否已安装，请从 TDS 端点运行以下查询：

```
1> SELECT extname, extversion FROM pg_extension where extname like 'pg_ad_mapping';
2> GO
extname       extversion
------------- ----------
pg_ad_mapping 0.1

(1 rows affected)
```

管理组登录名

按照管理登录信息中提到的步骤创建组登录名。我们建议登录名与 Active Directory（AD）安全组名称相同以便于维护，但这不是强制性要求。例如：

```
CREATE LOGIN [corp\accounts-group] FROM WINDOWS [WITH DEFAULT_DATABASE=database]
```

将 T-SQL 组登录名与 AD 安全组进行映射

您需要为每个要求访问数据库服务器的 AD 安全组显式预置 T-SQL Windows 组登录名。属于至少一个预置 AD 安全组的 AD 用户将获得访问数据库服务器的权限。

> **Note**
>
> 此 T-SQL 登录名无法再使用基于密码的身份验证进行身份验证。

例如，accounts-group 是 AD 中的一个安全组，如果您希望在 Babelfish 中预置该安全组，则必须使用 [corp\accounts-group] 格式。

- AD 安全组：accounts-group
- TSQL 登录名：[corp\accounts-group]
- 给定 TSQL 登录名的等效 PG 角色：accounts-group@CORP.EXAMPLE.COM

管理员现在可以通过以下 psql 命令，从 PostgreSQL 端点创建 AD 安全组和 T-SQL 登录名之间的映射。有关函数用法的更多信息，请参阅使用 pg_ad_mapping 扩展中的函数。

> **Note**
>
> 添加映射时，应以 login_name@FQDN 格式指定 T-SQL 登录名。通过 TDS 端点进行连接时，将忽略权重。有关使用权重的更多信息，请参阅通过 PostgreSQL 端口上的 PostgreSQL 端点连接到 Babelfish。

```
postgres=>select pgadmap_set_mapping('accounts-group', 'accounts-
group@CORP.EXAMPLE.COM', <SID>, <Weight>);
```

有关检索 AD 安全组的 SID 的信息，请参阅在 PowerShell 中检索 Active Directory 组 SID。

下表显示了从 AD 安全组到 T-SQL 登录名的示例映射：

AD 安全组	TSQL 登录名	给定 TSQL 登录名的等效 PG 角色	权重
accounts-group	[corp\accounts-group]	accounts-group@CORP.EXAMPLE.COM	7
sales-group	[corp\sales-group]	sales-group@CORP.EXAMPLE.COM	10
dev-group	[corp\dev-group]	dev-group@CORP.EXAMPLE.COM	7

```
postgres=> select admap.ad_sid, admap.ad_grp, lgn.orig_loginname, lgn.rolname,
 admap.weight from pgadmap_read_mapping() as admap, sys.babelfish_authid_login_ext as
 lgn where admap.pg_role = lgn.rolname;
    ad_sid    |     ad_grp     |    orig_loginname   |             rolname
 | weight
--------------+----------------+---------------------+---------------------------------
+--------
 S-1-5-67-890 | accounts-group | corp\accounts-group  | accounts-group@CORP.EXAMPLE.COM
 |     7
 S-1-2-34-560 | sales-group    | corp\sales-group     | sales-group@CORP.EXAMPLE.COM
 |     10
 S-1-8-43-612 | dev-group      | corp\dev-group       | dev-group@CORP.EXAMPLE.COM
 |     7
 (7 rows)
```

通过 TDS 端点连接到 Babelfish

在以下示例中，user1 是 accounts-group 和 sales-group 的成员，user2 是 accounts-group 和 dev-group 的成员。

用户名	AD 安全组成员资格
user1	accounts-group、sales-group
user2	accounts-group、dev-group

使用 sqlcmd 实用程序连接到 Babelfish 数据库服务器。可以按照以下示例验证用户（在本例中为 user1）是否已使用 Kerberos 进行身份验证：

```
1> select principal, gss_authenticated from pg_stat_gssapi where pid =
 pg_backend_pid();
2>  GO
principal              gss_authenticated
---------------------  -----------------
user1@CORP.EXAMPLE.COM  1

((1 rows affected))
1> select suser_name();
```

```
2>  GO
suser_name
----------
corp\user1

(1 rows affected)
```

在此示例中，user1 将继承 accounts-group 和 sales-group 的权限。可以使用 `sys.login_token` 系统视图来验证组成员资格。

```
1> SELECT name, type FROM sys.login_token;
2>  GO
name                type
------------------- ----
corp\accounts-group WINDOWS GROUP
corp\sales-group    WINDOWS GROUP

(2 rows affected)
```

审计和日志记录

要确定 AD 安全主体身份，请使用以下命令：

```
1> select suser_name();
2> GO
suser_name
----------
corp\user1

(1 rows affected)
```

目前，AD 用户身份在日志中不可见。可以开启 `log_connections` 参数来记录数据库会话建立情况。有关更多信息，请参阅 log_connections。输出包括作为主体的 AD 用户身份，如以下示例所示。然后，与此输出关联的后端 PID 有助于将操作归因回实际 AD 用户。

```
bbf_group_ad_login@babelfish_db:[615]:LOG: connection authorized:
 user=bbf_group_ad_login database=babelfish_db application_name=sqlcmd GSS
 (authenticated=yes, encrypted=yes, principal=user1@CORP.EXAMPLE.COM)
```

利用 AD 安全组成员资格的权限

继承服务器级权限

属于给定 AD 安全组成员的 AD 用户将继承向映射的 Windows 组登录名授予的服务器级权限。例如，考虑 accounts-group AD 安全组，该安全组被授予 Babelfish 上 sysadmin 服务器角色的成员资格。可以使用以下命令继承服务器级权限：

```
1> ALTER SERVER ROLE sysadmin ADD MEMBER [corp\accounts-group];
```

因此，属于 accounts-group AD 安全组成员的任何 Active Directory 用户都将继承与 sysadmin 角色关联的服务器级权限。这意味着，像 corp\user1 这样的用户（属于 accounts-group 的成员）现在可以在 Babelfish 中执行服务器级操作。

> **Note**
>
> 要执行服务器级 DDL，单独 AD 用户的 Windows 登录名必须存在。有关更多信息，请参阅 限制。

继承数据库级权限

要授予数据库级权限，必须创建数据库用户并使用 Windows 组登录名进行映射。属于给定 AD 安全组成员的 AD 用户将继承授予该数据库用户的数据库级权限。在以下示例中，可以看到 Windows 组 [corp\accounts-group] 的数据库级权限是如何分配的。

```
1> CREATE DATABASE db1;
2> GO
1> USE db1;
2> GO
Changed database context to 'db1'.
1> CREATE TABLE dbo.t1(a int);
2> GO
```

为 Windows 组登录名 [corp\accounts-group] 创建数据库用户 [corp\sales-group]。要执行此步骤，请使用作为 sysadmin 成员的登录名通过 TDS 端点进行连接。

```
1> CREATE USER [corp\accounts-group] FOR LOGIN [corp\accounts-group];
2> GO
```

现在，以 AD 用户 user1 的身份进行连接，来检查表 t1 的访问权限。由于我们尚未授予数据库级权限，因此这会导致权限被拒绝错误。

```
1> SELECT * FROM dbo.t1;
2> GO
Msg 33557097, Level 16, State 1, Server db-inst, Line 1
permission denied for table t1
```

向数据库用户 [corp\accounts-group] 授予对表 t1 的 SELECT 权限。要执行此步骤，请使用作为 sysadmin 成员的登录名通过 TDS 端点进行连接。

```
1> GRANT SELECT ON dbo.t1 TO [corp\accounts-group];
2> GO
```

以 AD 用户 user1 的身份进行连接，来验证访问权限。

```
1> SELECT * FROM dbo.t1;
2> GO
a
-----------

(0 rows affected)
```

基于默认或显式架构处理 DDL 语句行为

使用经 AD 身份验证的会话时，当前会话的默认架构由以下条件决定：

- 如果存在单独的数据库用户，则该用户的默认架构将被视为当前会话的默认架构。
- 如果存在组数据库用户的默认架构，则该组数据库用户的默认架构将被视为当前会话的默认架构，其主体 ID 为最小。

了解 CREATE DDL 语句行为

如果未在 CREATE DDL 语句中指定显式架构，则将在当前会话的默认架构中创建对象。如果无法确定架构是默认架构还是显式架构，则 DDL 语句将引发以下错误：

```
"Babelfish Unsupported Command : Schema required for CREATE DDLs when connecting with
 Active Directory Group authentication. Assign default schema to group user or specify
 schema in command."
```

Example ：Windows 组用户的默认架构不存在

Windows 组用户 [corp\accounts-group] 具有 NULL 默认架构，AD 用户 user1 正在尝试在没有显式指定架构的情况下执行 DDL。由于 user1 不存在单独的 Windows 登录名和用户，因此它将仅获得 Windows 组用户 [corp\accounts-group] 的数据库级权限。

```
1> create TABLE t2(a int);
2> GO

Msg 33557097, Level 16, State 1, Server db-inst, Line 1
Babelfish Unsupported Command : Schema required for CREATE DDLs when connecting with
 Active Directory Group authentication. Assign default schema to group user or specify
 schema in command.
```

> **Note**
>
> AD 用户 user1 不存在单独的 Windows 登录名和用户

Example ：Windows 组用户存在默认架构

使用 sysadmin 通过默认架构为 [corp\accounts-group] 登录名创建 Windows 组用户。

```
1> CREATE USER [corp\accounts-group] FOR LOGIN [corp\accounts-group] WITH
 DEFAULT_SCHEMA = sch_acc;
```

```
2> GO
1> CREATE SCHEMA sch_acc AUTHORIZATION [gad\accounts-group];
2> GO
1> SELECT name, principal_id, default_schema_name FROM sys.database_principals WHERE
 name = 'corp\accounts-group';
2> GO

name               principal_id default_schema_name
------------------ ------------ ------------------
corp\accounts-group 24162        sch_acc

(1 rows affected)
```

尝试使用 AD 用户 user1 在不显式指定架构的情况下创建对象。表 t2 将在 [corp\accounts-group] Windows 组用户的默认架构中创建。此对象的所有者将与架构 sch_acc 的所有者相同。

```
1> CREATE TABLE t_group(a int);
2> GO
1> SELECT name, schema_name(schema_id) FROM sys.objects WHERE name like 't_group';
2> GO

name    schema_name
------- -----------
t_group sch_acc

(1 rows affected)
```

> **Note**
>
> AD 用户 user1 不存在单独的 Windows 登录名和用户

Example ：AD 用户也存在单独的数据库用户

如果 AD 用户也存在单独的数据库用户，则将始终在与单独的数据库用户关联的架构中创建对象。如果数据库用户不存在该架构，将使用 dbo 架构。为 AD 用户 user1 创建单独的 Windows 登录名和数据库用户。使用 sysadmin 登录名通过 TDS 端点进行连接

```
1> CREATE LOGIN [corp\user1] FROM WINDOWS;
2> GO
1> CREATE USER [corp\user1] FOR LOGIN [corp\user1] WITH DEFAULT_SCHEMA = sch1;
2> GO
1> CREATE SCHEMA sch1 AUTHORIZATION [corp\user1];
2> GO
1> SELECT name, default_schema_name FROM sys.database_principals WHERE name = 'corp
\user1';
2> GO

name      default_schema_name
--------- -------------------
corp\user1 sch1

(1 rows affected)
```

使用 AD 用户 user1 进行连接，然后尝试在不显式指定架构的情况下创建对象。将在架构 sch1 中创建表 t2。另请注意，此对象的所有者将与架构 sch1 的所有者相同。

```
1> CREATE TABLE t2(a int);
2> GO
1> SELECT name, schema_name(schema_id) FROM sys.objects WHERE name like 't2';
2> GO

name schema_name
---- -----------
t2   sch1

(1 rows affected)
```

限制

- 转储/还原实用程序不支持转储 pg_ad_mapping 扩展映射。还原后，您将需要重新创建这些映射。
- 具有 pg_ad_mapping 的 Babelfish 和 Aurora PostgreSQL 实例不支持蓝绿部署。
- 不支持隐式架构创建。不支持需要隐式架构创建的 DDL 语句。

- 当单独的 Windows 登录名不存在而只有组 Windows 登录名存在时，在经组 AD 身份验证的会话中不支持服务器级 DDL ALTER AUTHORIZATION ON DATABASE、CREATE DATABASE、CREATE LOGIN、ALTER LOGIN、ALTER SERVER ROLE、ALTER DATABASE。要解决此限制，建议在经密码验证身份的会话中执行这些操作或创建单独的 Windows 登录名。
- 不支持隐式用户创建。理想的 T-SQL 行为 [Babelfish 尚不支持]；在某些情况下，例如 DDL 和访问控制语句（如 GRANT/REVOKE），此时在命令中指定了 AD 用户的名称，但该用户在数据库中不存在，则会隐式创建名为 AD 用户的数据库用户。
- 对于 PL/pgSQL 过程或函数中的 DDL（它们在经组 AD 身份验证的会话中从 PSQL 端点创建并从 TDS 端点执行）：
  - 将支持 ALTER/DROP 语句。
  - 如果未显式提供架构且当前会话的默认架构为 null，则 CREATE TABLE、CREATE VIEW、CREATE INDEX、CREATE FUNCTION/PROC、CREATE TYPE、CREATE SEQUENCE、CREATE TRIGGER、SELECT INTO、CREATE FULLTEXT INDEX、CREATE UNIQUE INDEX 将引发错误。
  - 对于 PG（不在 T-SQL 中）特定对象 CREATE 订阅、CREATE 表空间、CREATE 策略、CREATE 转换，将不支持 CREATE DATABASE、CREATE EXTENSION 和所有其它 CREATE 语句。
- 经组 AD 身份验证的会话不支持来自 PostgreSQL 端点的 DDL。作为一种解决方法，您始终可以使用主用户或任何使用基于密码的身份验证机制的其它用户进行连接。
- 像 SUSER_SID()、IS_SRVROLEMEMBER()、IS_MEMBER()、sys.dm_exec_sessions 这样的系统对象具有以下限制。
  - 提供 AD 用户或 AD 安全组时，SUSER_SID() 不会返回 SID。
  - 如果当前 AD 用户从任何 Windows 组登录名的服务器角色成员资格继承服务器角色成员资格，则 IS_SRVROLEMEMBER() 将不考虑角色成员资格。
  - 对于任何与 Windows 组相关的查询，IS_MEMBER() 都将返回 false。
  - sys.dm_exec_sessions 不会显示预期值 login_name、nt_user_name 列。

通过 PostgreSQL 端口上的 PostgreSQL 端点连接到 Babelfish

也可以利用从 TDS 端口创建的组登录名，通过 PostgreSQL 端口进行连接。要通过 PostgreSQL 端口进行连接，您需要从 PostgreSQL 客户端应用程序中按照格式 `<ad_username@FQDN>` 指定 AD 用户的名称。不能使用 `<DNS domain name\ad_username>` 格式。

默认情况下，PostgreSQL 对用户名使用区分大小写的比较。要让 Aurora PostgreSQL 将 Kerberos 用户名解释为不区分大小写，必须在自定义 Babelfish 集群参数组中将 krb_caseins_users 参数设置为

true。默认情况下，此参数设置为 false。有关更多信息，请参阅 [将 Aurora PostgreSQL 数据库集群配置为使用不区分大小写的用户名]。

当 AD 用户属于多个组时，T-SQL 和 PostgreSQL 端点之间的行为差异

考虑 AD 用户 user1 是两个 AD 安全组 [corp\accounts-group] 和 [corp\sales-group] 的一部分，数据库管理员已按以下方式设置了用户映射。

```
postgres=> select * from pgadmap_read_mapping();

ad_sid       | pg_role                           | weight | ad_grp
-------------+-----------------------------------+--------+---------------
S-1-5-67-980 | accounts-group@CORP.EXAMPLE.COM | 7      | accounts-group
S-1-2-34-560 | sales-group@CORP.EXAMPLE.COM    | 10     | sales-group
(2 rows)
```

如果用户从 T-SQL 端点进行连接，则在授权期间，该用户将继承所有关联 T-SQL 登录名的权限。在此示例中，user1 将继承来自这两个 T-SQL 组登录名的权限并集，但将忽略权重。这符合标准的 T-SQL 行为。

但是，如果同一个用户从 PostgreSQL 端点进行连接，则它只能从一个权重最高的关联 T-SQL 登录名继承权限。如果为两个 T-SQL 组登录名分配的权重相同，则 AD 用户将继承与最新添加的映射相对应的 T-SQL 登录名的权限。对于 PostgreSQL，建议指定权重，以反映各个数据库角色的相对权限/特权来避免模糊不清。在下面的示例中，user1 通过 PSQL 端点进行了连接，并且仅继承了 sales-groups 权限。

```
babelfish_db=> select session_user, current_user;

   session_user               |   current_user
------------------------------+---------------------------
 sales-group@CORP.EXAMPLE.COM | sales-group@CORP.EXAMPLE.COM
(1 row)


babelfish_db=> select principal, gss_authenticated from pg_stat_gssapi where pid =
 pg_backend_pid();

     principal          | gss_authenticated
```

```
------------------------+--------------------
 user1@CORP.EXAMPLE.COM | t
(1 row)
```

连接到 Babelfish 数据库集群

要连接到 Babelfish，请连接到运行 Babelfish 的 Aurora PostgreSQL 集群的端点。您的客户端可以使用以下符合 TDS 版本 7.1 至 7.4 的客户端驱动程序之一：

- 开放式数据库连接 (ODBC)
- OLE 数据库驱动程序/MSOLEDBSQL
- Java Database Connectivity（JDBC）版本 8.2.2（mssql-jdbc-8.2.2）及更高版本
- 面向 SQL Server 的 Microsoft SqlClient 数据提供程序
- 面向 SQL Server 的 .NET 数据提供程序
- SQL Server 本机客户端 11.0（已弃用）
- OLE DB 提供商/SQLOLEDB（已弃用）

使用 Babelfish，您可以运行以下命令：

- TDS 端口上的 SQL Server 工具、应用程序和语法，原定设置为端口 1433。
- TDS 端口上的 PostgreSQL 工具、应用程序和语法，原定设置为端口 5432。

要概括地了解有关连接到 Aurora PostgreSQL 的更多信息，请参阅连接到 Amazon Aurora PostgreSQL 数据库集群。

> **Note**
>
> 不支持使用 SQL Server OLEDB 提供程序访问元数据的第三方开发人员工具。建议您使用 SQL Server JDBC、ODBC 或 SQL 原生客户端连接来访问这些工具。

从 Babelfish 版本 5.1.0 开始，默认情况下会强制执行端到端连接加密。要确保持续连接：

- 为连接配置 SSL/TLS 加密。有关更多信息，请参阅 使用 SSL/TLS 加密与数据库集群的连接。
- 在客户端计算机上导入所需的证书。有关更多信息，请参阅将 SSL 与 Microsoft SQL Server 数据库实例结合使用。

如果要继续使用先前 Babelfish 版本（早于版本 5.1.0）中的加密设置，则可以在数据库集群参数组中将 `rds.force_ssl` 参数设置为 `0`。

主题

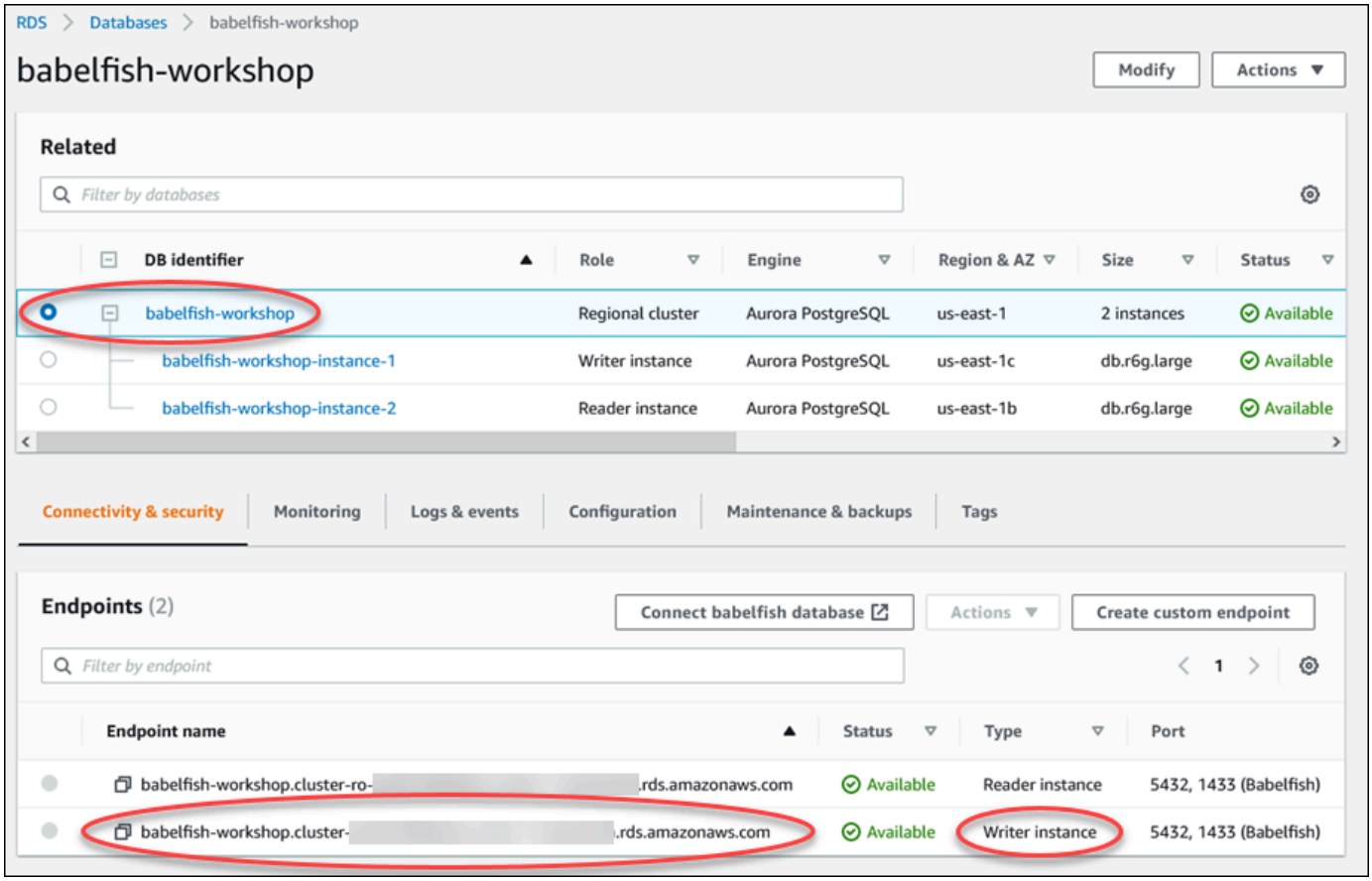
- 查找写入器端点和端口号
- 创建 C# 或 JDBC 客户端到 Babelfish 的连接
- 使用 SQL Server 客户端连接到数据库集群
- 使用 PostgreSQL 客户端连接到数据库集群

查找写入器端点和端口号

要连接到 Babelfish 数据库集群，请使用与数据库集群的写入器（主）实例关联的端点。该实例的状态必须为 Available（可用）。在创建 Babelfish for Aurora PostgreSQL 数据库集群之后，这些实例可能需要多达 20 分钟才能可用。

要查找数据库端点

1. 打开 Babelfish 的控制台。
2. 从导航窗格中选择 Databases (数据库)。
3. 从列出的集群中选择 Babelfish for Aurora PostgreSQL 数据库集群以查看其详细信息。
4. 在 Connectivity & security（连接和安全性）选项卡中，注意可用的集群端点值。在执行数据库写入或读取操作的任何应用程序的连接字符串中，使用写入器实例的集群端点。



有关 Aurora 数据库集群详情的更多信息，请参阅[创建 Amazon Aurora 数据库集群]。

创建 C# 或 JDBC 客户端到 Babelfish 的连接

在下面，您可以找到一些使用 C# 和 JDBC 类连接到 Babelfish for Aurora PostgreSQL 的示例。

Example ：使用 C# 代码连接到数据库集群

```
string dataSource = 'babelfishServer_11_24';

//Create connection
connectionString = @"Data Source=" + dataSource
    +";Initial Catalog=your-DB-name"
    +";User ID=user-id;Password=password";

// [optional] To validate server certificate during TLS/SSL connection
connectionString = connectionString + ";ServerCertificate=/path/to/certificate.pem";


SqlConnection cnn = new SqlConnection(connectionString);
cnn.Open();
```

Example ：使用通用 JDBC API 类和接口连接到数据库集群

```
String dbServer =
   "database-babelfish.cluster-123abc456def.us-east-1-rds.amazonaws.com";
String connectionUrl = "jdbc:sqlserver://" + dbServer + ":1433;" +
    "databaseName=your-DB-name;user=user-id;password=password";

// [optional] To validate server certificate during TLS/SSL connection
connectionUrl = connectionUrl + ";serverCertificate=/path/to/certificate.pem";

// Load the SQL Server JDBC driver and establish the connection.
System.out.print("Connecting Babelfish Server ... ");
Connection cnn = DriverManager.getConnection(connectionUrl);
```

Example ：使用特定于 SQL Server 的 JDBC 类和接口连接到数据库集群

```
// Create datasource.
SQLServerDataSource ds = new SQLServerDataSource();
ds.setUser("user-id");
ds.setPassword("password");
String babelfishServer =
   "database-babelfish.cluster-123abc456def.us-east-1-rds.amazonaws.com";
```

```
ds.setServerName(babelfishServer);
ds.setPortNumber(1433);
ds.setDatabaseName("your-DB-name");

// [optional] To validate server certificate during TLS/SSL connection
ds.setServerCertificate("/path/to/certificate.pem");

Connection con = ds.getConnection();
```

Important

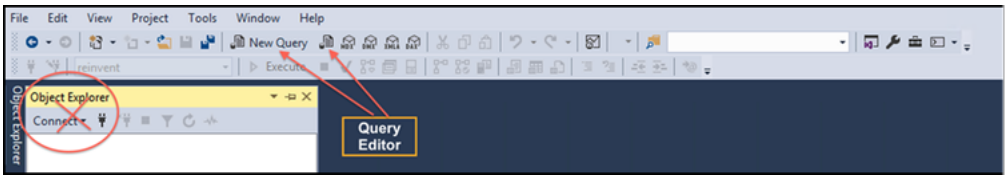
确保证书与 AWS Management Console上数据库集群配置中显示的证书颁发机构相匹配。

使用 SQL Server 客户端连接到数据库集群

您可以使用 SQL Server 客户端与 TDS 端口上的 Babelfish 进行连接。自 Babelfish 2.1.0 及更高版本开始，您可以使用 SSMS Object Explorer 或 SSMS 查询编辑器连接到 Babelfish 集群。

限制

- 在 Babelfish 2.1.0 及更早版本中，使用 PARSE 检查 SQL 语法不能正常工作的问题。PARSE 命令不是在不运行查询的情况下检查语法，而是运行查询但不显示任何结果。使用 SMSS <Ctrl><F5> 组合键检查语法具有相同的异常行为，也就是说，Babelfish 意外地运行查询而不提供任何输出。
- Babelfish 不支持 MARS（多个活动结果集）。请确保用来连接到 Babelfish 的所有客户端应用程序都未设置为使用 MARS。
- 对于 Babelfish 1.3.0 和较早版本，SSMS 只支持查询编辑器。要将 SSMS 与 Babelfish 一起使用，请务必在 SSMS 中打开查询编辑器连接对话框，而不是打开 Object Explorer。如果确实打开了 Object Explorer 对话框，请取消该对话框并重新打开查询编辑器。在下图中，您可以找到连接到 Babelfish 1.3.0 或更早版本时要选择的菜单选项。



有关 SQL Server 和 Babelfish 之间的互操作性和行为差异的更多信息，请参阅 适用于 Aurora PostgreSQL 的 Babelfish 与 SQL Server 之间的区别。

使用 sqlcmd 连接到数据库集群

您只能通过使用版本 19.1 和更早版本的 SQL Server `sqlcmd` 命令行客户端连接到支持 Babelfish 的 Aurora PostgreSQL 数据库集群并与之进行交互。不支持 SSMS 版本 19.2 连接到 Babelfish 集群。使用以下命令进行连接。

```
sqlcmd -S endpoint,port -U login-id -P password -d your-DB-name
```

这些选项如下所示：

- `-S` 是数据库集群的端点和（可选的）TDS 端口。
- `-U` 是用户的登录名。
- `-P` 是与该用户关联的密码。

- -d 是您的 Babelfish 数据库的名称。

连接后，您可以使用许多与 SQL Server 一起使用的相同命令。要获取一些示例，请参阅获取 Babelfish 系统目录中的信息。

使用 SSMS 连接到数据库集群

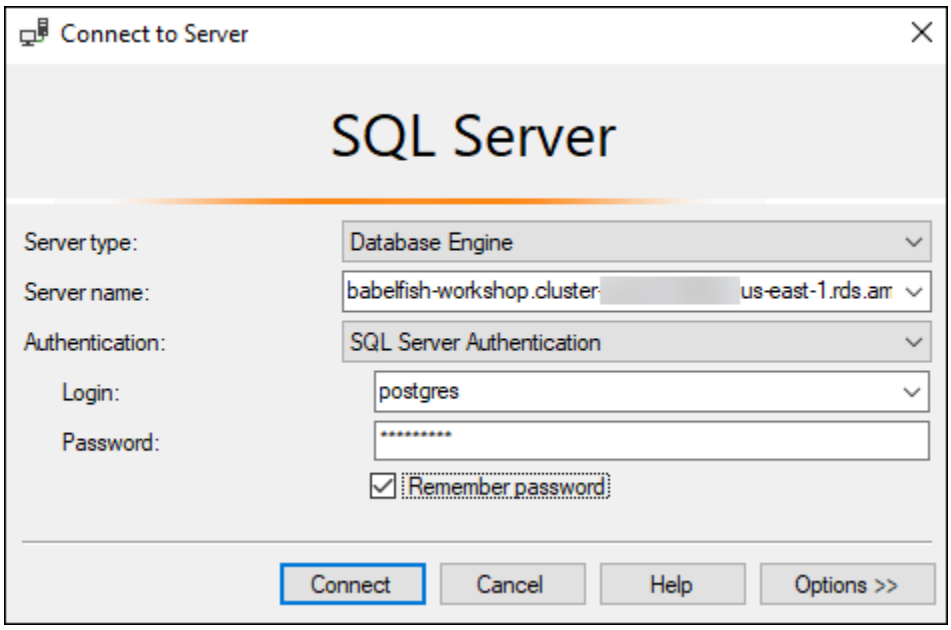
您可以使用 Microsoft SQL Server Management Studio (SSMS) 连接到运行 Babelfish 的 Aurora PostgreSQL 数据库集群。SSMS 包括各种工具，包括在将 SQL Server 数据库迁移到 Babelfish for Aurora PostgreSQL 中讨论的 SQL Server 导入和导出向导。有关 SSMS 的更多信息，请参阅 Microsoft 文档中的下载 SQL Server Management Studio (SSMS)。要配置 SSL/TLS，请参阅将 SSL 与 Microsoft SQL Server 数据库实例结合使用。

要使用 SSMS 连接到 Babelfish 数据库

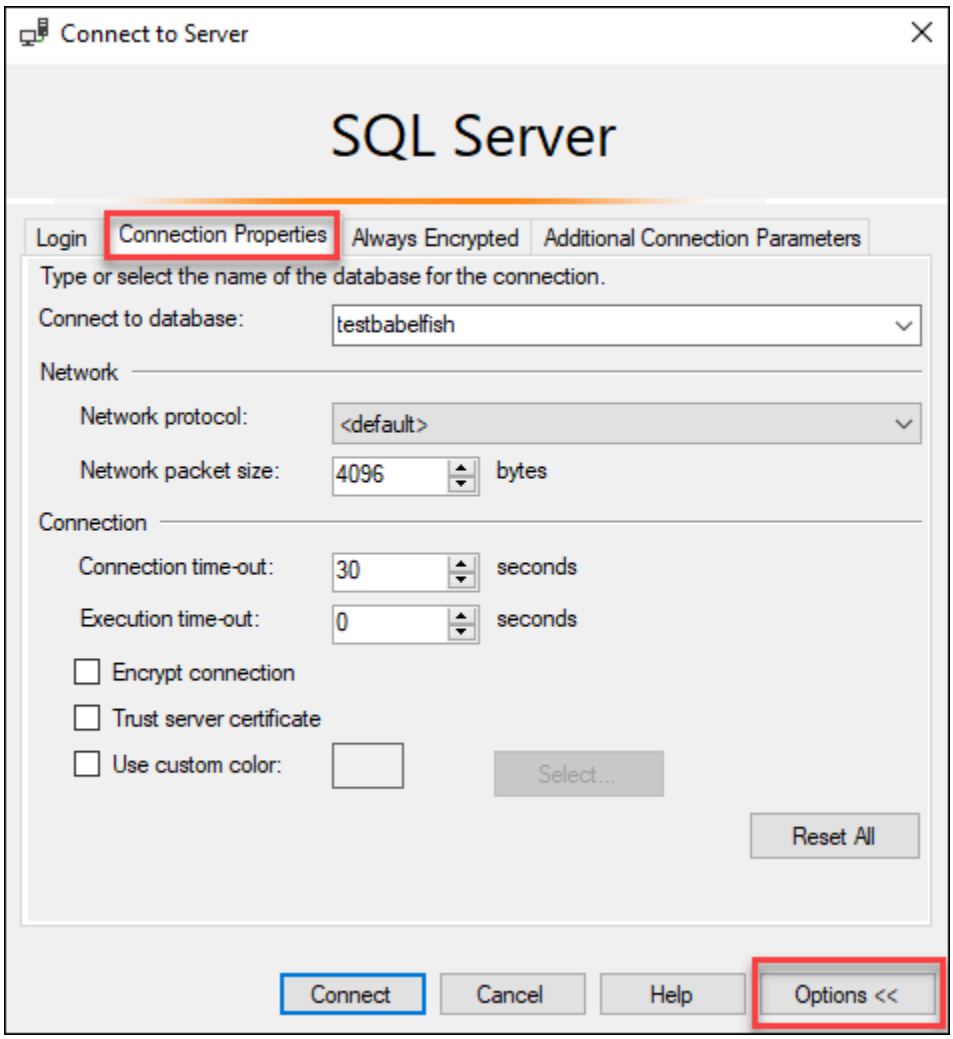
1. 启动 SSMS。
2. 打开 Connect to Server（连接至服务器）对话框。要继续连接，请执行以下操作之一：
   - 选择 New Query（新查询）。
   - 如果查询编辑器已打开，请选择 Query（查询）、Connection（连接）、Connect（连接）。
3. 为您的数据库提供以下信息：
   a. 对于 Server type，选择 Database Engine。
   b. 对于 Server name（服务器名称），请输入 DNS 名称。例如，您的服务器名称应类似于以下示例。

      ```
      cluster-name.cluster-555555555555.aws-region.rds.amazonaws.com,1433
      ```

   c. 对于 Authentication，选择 SQL Server Authentication。
   d. 对于 Login（登录），输入创建数据库时选择的用户名。
   e. 对于 Password（密码），输入创建数据库时选择的密码。



4. （可选）选择 Options（选项），然后选择 Connection Properties（连接属性）选项卡。



5. （可选）对于 Connect to database（连接到数据库）中，指定要连接到的迁移 SQL Server 数据库的名称，然后选择 Connect（连接）。

   如果出现指示 SSMS 无法应用连接字符串的消息，请选择 OK（确定）。

   如果您在连接到 Babelfish 时遇到问题，请参阅连接失败。

   有关 SQL Server 连接问题的更多信息，请参阅《Amazon RDS 用户指南》中的排除与 SQL Server 数据库实例的连接故障。

使用 PostgreSQL 客户端连接到数据库集群

您可以使用 PostgreSQL 客户端连接到 PostgreSQL 端口上的 Babelfish。从版本 5.1.0 开始，Babelfish 服务器默认情况下会强制执行端到端连接加密。更新应用程序以使用 SSL/TLS 证书。有关配置 SSL/TLS 证书的更多信息，请参阅the section called “利用 SSL/TLS 保护 Aurora PostgreSQL 数据”。

使用 psql 连接到数据库集群

您可以从 PostgreSQL 网站下载 PostgreSQL 客户端。请按照特定于您的操作系统版本的说明安装 psql。

您可以通过使用 `psql` 命令行客户端查询支持 Babelfish 的 Aurora PostgreSQL 数据库集群。连接时，请使用 PostgreSQL 端口（默认情况下，端口 5432）。通常，除非您更改了默认端口号，否则不需要指定端口号。使用以下命令从 `psql` 客户端连接到 Babelfish：

```
psql -h bfish-db.cluster-123456789012.aws-region.rds.amazonaws.com
-p 5432 -U postgres -d babelfish_db
```

参数如下所示：

- `-h` – 要访问的数据库实例集群（集群端点）的主机名。
- `-p` – 用于连接到数据库实例的 PostgreSQL 端口号。
- `-d` – 要连接到的数据库。默认值为 `babelfish_db`。
- `-U` – 要访问的数据库用户账户。（该示例显示了默认的主用户名。）

当您在 psql 客户端上运行 SQL 命令时，您可以用分号结束该命令。例如，以下 SQL 命令查询 pg_tables 系统视图以返回有关数据库中每个表的信息。

```
SELECT * FROM pg_tables;
```

psql 客户端还包含一组内置元命令。元命令是一种快捷方式，可调整格式或提供以易于使用的格式返回元数据的快捷方式。例如，以下元命令返回类似以前的 SQL 命令的信息：

```
\d
```

元命令不需要用分号 (;) 终止。

要退出 psql 客户端，请输入 `\q`。

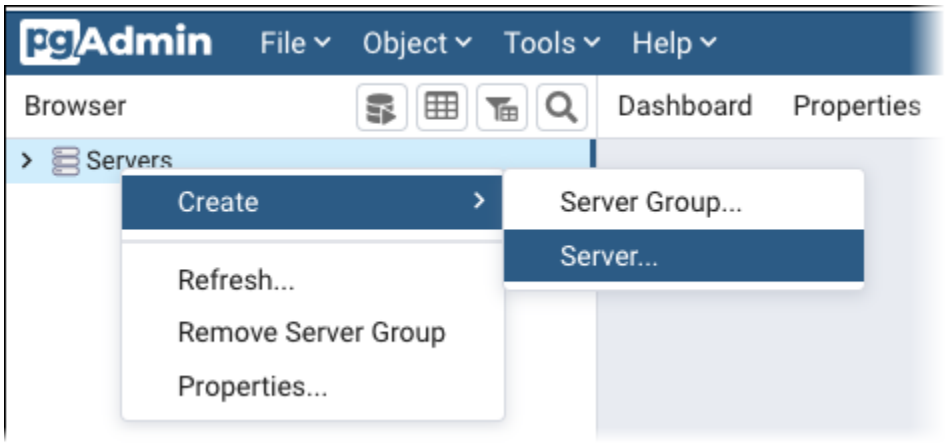
有关使用 psql 客户端查询 Aurora PostgreSQL 集群的更多信息，请参阅 PostgreSQL 文档。

使用 pgAdmin 连接到数据库集群

您可以使用 pgAdmin 客户端以本机 PostgreSQL 方言访问数据。

要使用 pgAdmin 客户端连接到集群

1. 从 pgAdmin 网站中下载并安装 pgAdmin 客户端。
2. 打开客户端并通过 pgAdmin 进行身份验证。
3. 打开 Servers（服务器）的上下文（右键单击）菜单，然后选择 Create（创建）、Server（服务器）。

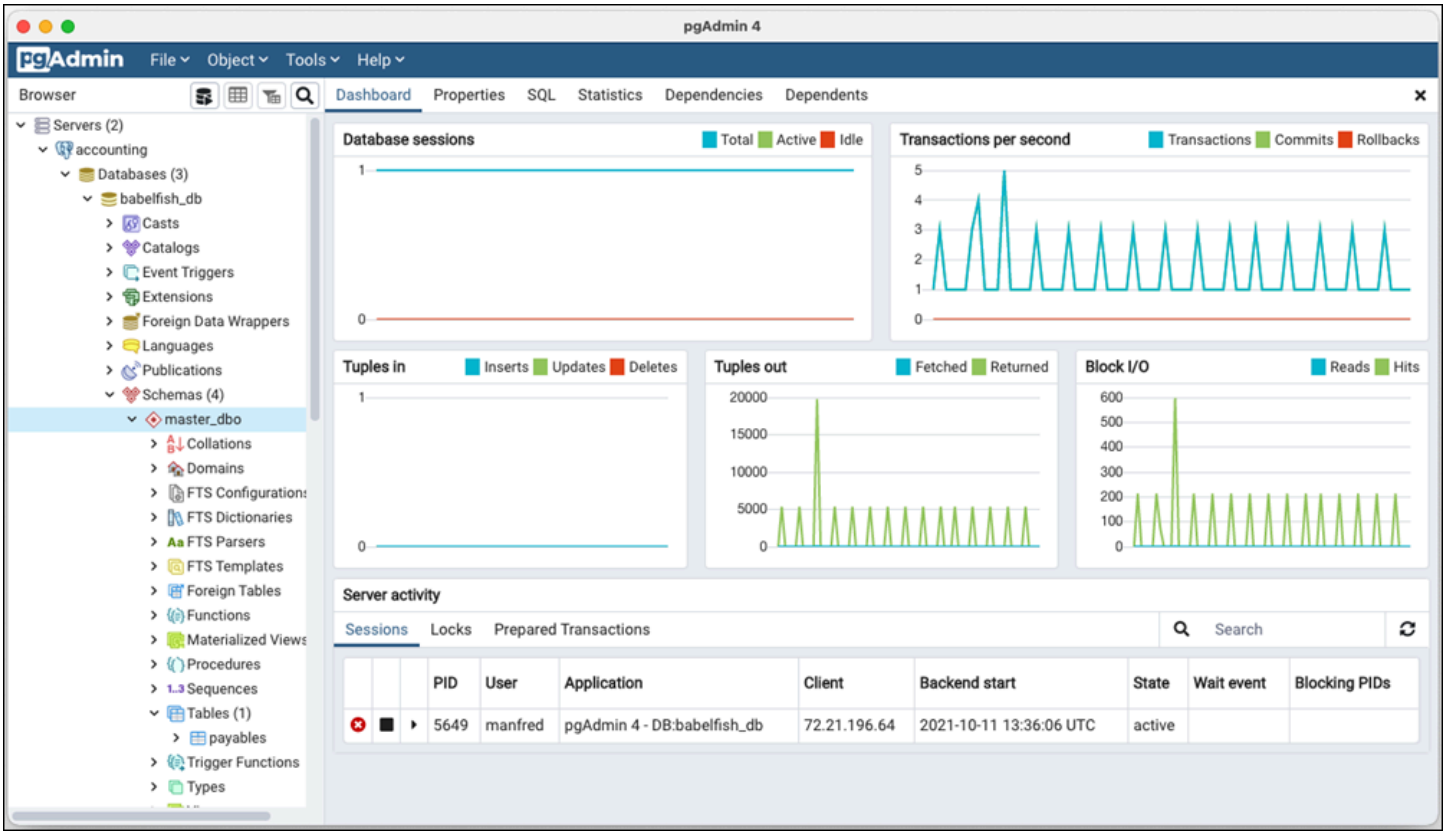


4. 在 Create - Server（创建 - 服务器）对话框中输入信息。

   在 Connection（连接）选项卡中，添加 Host（主机）的 Aurora PostgreSQL 集群地址和 Port（端口）的 PostgreSQL 端口号（默认为 5432）。提供身份验证详细信息，然后选择 Save（保存）。



连接后，您可以使用 pgAdmin 功能在 PostgreSQL 端口上监控和管理 Aurora PostgreSQL 集群。



要了解更多信息，请参阅 pgAdmin Web 页。

使用 Babelfish

接下来，您可以查找 Babelfish 的使用信息，包括使用 Babelfish 和 SQL Server 之间的一些差异以及 Babelfish 和 PostgreSQL 数据库之间的一些差异。

主题

- 获取 Babelfish 系统目录中的信息
- 在适用于 Aurora PostgreSQL 的 Babelfish 中管理权限和访问控制
- 适用于 Aurora PostgreSQL 的 Babelfish 与 SQL Server 之间的区别
- 使用具有有限实施的 Babelfish 功能
- 提高 Babelfish 查询性能
- 将 Aurora PostgreSQL 扩展与 Babelfish 搭配使用
- Babelfish 支持链接服务器
- 在 Babelfish 中使用全文搜索
- Babelfish 支持地理空间数据类型
- 了解 Babelfish 中的分区

获取 Babelfish 系统目录中的信息

通过查询 SQL Server 中使用的许多相同系统视图，可以获取有关存储在 Babelfish 集群中的数据库对象的信息。Babelfish 的每个新版本都增加了对更多系统视图的支持。有关当前可用视图的列表，请参阅 SQL Server system catalog views表。

这些系统视图提供了系统目录 (`sys.schemas`) 中的信息。就 Babelfish 而言，这些视图同时包含 SQL Server 和 PostgreSQL 系统架构。要查询 Babelfish 以获取系统目录信息，可以使用 TDS 端口或 PostgreSQL 端口，如以下示例所示。

- 使用 **`sqlcmd`** 或另一个 SQL Server 客户端查询 T-SQL 端口。

```
1> SELECT * FROM sys.schemas
2> GO
```

此查询返回 SQL Server 和 Aurora PostgreSQL 系统架构，如以下内容中所示。

```
name
------------------------------------------------------------
```

```
demographic_dbo
public
sys
master_dbo
tempdb_dbo
...
```

- 使用 **psql** 或 **pgAdmin** 查询 PostgreSQL 端口 此示例使用 psql 列出架构元命令 (\dn)：

```
babelfish_db=> \dn
```

此查询返回的结果集与 sqlcmd 在 T-SQL 端口上返回的结果集相同。

```
        List of schemas
             Name
------------------------------

 demographic_dbo

 public
 sys
 master_dbo
 tempdb_dbo
...
```

Babelfish 中提供的 SQL Server 系统目录

在下表中，您可以找到 Babelfish 中当前实现的 SQL Server 视图。有关 SQL Server 中的系统目录的更多信息，请参阅 Microsoft 文档中的系统目录视图 (Transact-SQL)。

视图名称	描述或 Babelfish 限制（如果有）
sys.all_columns	所有表和视图中的所有列
sys.all_objects	所有架构中的所有对象
sys.all_sql_modules	sys.sql_modules 和 sys.system_sql_modules 的联合
sys.all_views	所有架构中的所有视图

视图名称	描述或 Babelfish 限制（如果有）
sys.columns	用户定义的表和视图中的所有列
sys.configurations	Babelfish 支持仅限于单个只读配置。
sys.data_spaces	为每个数据空间包含一行。这可以是文件组、分区方案或 FILESTREAM 数据文件组。
sys.database_files	一个每数据库视图，它对于存储在数据库本身中的数据库的每个文件都包含一行。
sys.database_mirroring	有关信息，请参阅 Microsoft Transact-SQL 文档中的 sys.database_mirroring。
sys.database_principals	有关信息，请参阅 Microsoft Transact-SQL 文档中的 sys.database_principals。
sys.database_role_members	有关信息，请参阅 Microsoft Transact-SQL 文档中的 sys.database_role_members。
sys.databases	所有架构中的所有数据库
sys.dm_exec_connections	有关信息，请参阅 Microsoft Transact-SQL 文档中的 sys.dm_exec_connections。
sys.dm_exec_sessions	有关信息，请参阅 Microsoft Transact-SQL 文档中的 sys.dm_exec_sessions。
sys.dm_hadr_database_replica_states	有关信息，请参阅 Microsoft Transact-SQL 文档中的 sys.dm_hadr_database_replica_states。
sys.dm_os_host_info	有关信息，请参阅 Microsoft Transact-SQL 文档中的 sys.dm_os_host_info。
sys.endpoints	有关信息，请参阅 Microsoft Transact-SQL 文档中的 sys.endpoints。
sys.indexes	有关信息，请参阅 Microsoft Transact-SQL 文档中的 sys.indexes。

视图名称	描述或 Babelfish 限制（如果有）
sys.languages	有关信息，请参阅 Microsoft Transact-SQL 文档中的 sys.languages。
sys.schemas	所有架构
sys.server_principals	所有登录名和角色
sys.sql_modules	有关信息，请参阅 Microsoft Transact-SQL 文档中的 sys.sql_modules。
sys.sysconfigures	Babelfish 支持仅限于单个只读配置。
sys.syscurconfigs	Babelfish 支持仅限于单个只读配置。
sys.sysprocesses	有关信息，请参阅 Microsoft Transact-SQL 文档中的 sys.sysprocesses。
sys.system_sql_modules	有关信息，请参阅 Microsoft Transact-SQL 文档中的 sys.system_sql_modules。
sys.table_types	有关信息，请参阅 Microsoft Transact-SQL 文档中的 sys.table_types。
sys.tables	一个架构中的所有表
sys.xml_schema_collections	有关信息，请参阅 Microsoft Transact-SQL 文档中的 sys.xml_schema_collections。

PostgreSQL 实现的系统目录与 SQL Server 对象目录视图类似。有关系统目录的完整列表，请参阅 PostgreSQL 文档的系统目录。

Babelfish 支持 DDL 导出

在 Babelfish 2.4.0 和 3.1.0 版本中，Babelfish 支持使用各种工具导出 DDL。例如，您可以使用 SQL Server Management Studio（SSMS）中的此功能为适用于 Aurora PostgreSQL 的 Babelfish 数据库中的各种对象生成数据定义脚本。然后，可以使用此脚本中生成的 DDL 命令在另一个适用于 Aurora PostgreSQL 的 Babelfish 数据库或 SQL Server 数据库中创建相同的对象。

Babelfish 在指定的版本中支持以下对象的 DDL 导出。

对象列表	2.4.0	3.1.0
用户表	支持	是
主键	支持	是
外键	支持	是
唯一约束	支持	是
索引	支持	是
检查约束	支持	是
视图	支持	是
存储过程	支持	是
用户定义的函数	支持	是
表值函数	支持	是
触发	支持	是
用户定义的数据类型	否	否
用户定义的表类型	否	否
Users	否	否
登录名	否	否
序列	否	否
角色	否	否

导出的 DDL 的局限性

- 在使用导出的 DDL 重新创建对象之前使用备用方案 – Babelfish 并不支持导出的 DDL 脚本中的所有命令。使用备用方案可避免在 Babelfish 中通过 DDL 命令重新创建对象时导致错误。有关备用方案的更多信息，请参阅使用转义孵化管理 Babelfish 错误处理
- 包含带有显式 COLLATE 子句的 CHECK 约束的对象 – 带有从 SQL Server 数据库生成的这些对象的脚本具有与 Babelfish 数据库中不同但等效的排序规则。例如，一些排序规则（如 sql_latin1_general_cp1_cs_as、sql_latin1_general_cp1251_cs_as 和 latin1_general_cs_as）将生成为 latin1_general_cs_as，这是最接近的 Windows 排序规则。

在适用于 Aurora PostgreSQL 的 Babelfish 中管理权限和访问控制

在适用于 Aurora PostgreSQL 的 Babelfish 中，可以管理数据库、架构和对象的权限和访问控制。下表将概述用于在 Babelfish 中授予权限以实现各种访问控制场景的特定 SQL 命令。它将涵盖可以实现的受支持使用案例，以及当前不支持的案例的解决方法。这可让您在使用 Babelfish 数据库时配置适当的权限，来满足安全性和合规性要求。

支持的使用案例

下表说明了 Babelfish 支持的使用案例。对于每个使用案例，该表显示了实现该使用案例所需的操作以及示例 SQL 命令。

应用场景	操作	SQL 命令	评论	Babelfish 版本兼容性
支持登录名在任何数据库中执行 SELECT/DML/DDL	将登录名添加到 sysadmin 服务器角色	ALTER SERVER ROLE sysadmin ADD MEMBER `login`	无	所有版本
支持登录名在数据库中执行 SELECT/DML/DDL	使登录名成为数据库的拥有者	ALTER AUTHORIZATION ON DATABASE::`database` TO `login`	数据库只能有一个拥有者。	版本 3.4 及更高版本

应用场景	操作	SQL 命令	评论	Babelfish 版本兼容性
支持数据库用户对架构执行 SELECT/DML	向数据库用户授予对于架构的权限	GRANT SELECT/EXECUTE/INSERT/UPDATE/DELETE ON SCHEMA::`schema` TO `user`	无	版本 3.6 及更高版本、4.2 及更高版本
支持数据库用户对架构执行 SELECT/DML	在架构创建时将数据库用户设为架构的拥有者	CREATE SCHEMA `schema` AUTHORIZATION `user`	当前不支持在创建架构后更改架构所有权。	版本 1.2 及更高版本
支持数据库用户对于对象执行 SELECT/DML	向数据库用户授予对于对象的权限	GRANT SELECT/EXECUTE/INSERT/UPDATE/DELETE ON OBJECT::`object` TO `user`	无	所有版本
支持数据库用户在数据库中执行 SELECT/DML/DDL，包括删除数据库	将用户添加到 db_owner 固定数据库角色	ALTER ROLE db_owner ADD MEMBER `user`	仅限将用户添加到 db_owner 固定数据库角色。尚不支持将角色添加到 db_owner 角色。	版本 4.5 及更高版本、5.1 及更高版本
支持自定义数据库角色的用户或成员在数据库中仅执行 SELECT	将用户或角色添加到 db_datareader 固定数据库角色	ALTER ROLE db_datareader ADD MEMBER `user` / `role`	无	版本 4.5 及更高版本、5.1 及更高版本

应用场景	操作	SQL 命令	评论	Babelfish 版本兼容性
支持自定义数据库角色的用户或成员在数据库中仅执行 DML	将用户或角色添加到 db_datawriter 固定数据库角色	ALTER ROLE db_datawriter ADD MEMBER `user` / `role`	无	版本 4.5 及更高版本、5.1 及更高版本
支持自定义数据库角色的用户或成员在数据库中仅执行 DDL	将用户或角色添加到 db_accessadmin 固定数据库角色	ALTER ROLE db_accessadmin ADD MEMBER `user` / `role`	无	版本 4.5 及更高版本、5.1 及更高版本
只允许自定义数据库角色的用户或成员对自定义角色执行 CREATE/ALTER/DROP，对数据库中的对象权限执行 GRANT/REVOKE，以及/或者在数据库中执行 CREATE SCHEMA 操作。	将用户或角色添加到 db_securityadmin 固定数据库角色	ALTER ROLE db_securityadmin ADD MEMBER `user` / `role`	无	版本 4.5 及更高版本、5.1 及更高版本

应用场景	操作	SQL 命令	评论	Babelfish 版本兼容性
只允许自定义数据库角色的用户或成员对任意用户执行 CREATE/ALTER/DROP，授予和撤销数据库访问权限以及将用户账户映射到登录名，以及/或者在数据库中执行 CREATE SCHEMA 操作	将用户或角色添加到 db_accessadmin 固定数据库角色	ALTER ROLE db_accessadmin ADD MEMBER `user` / `role`	无	版本 4.5 及更高版本、5.1 及更高版本
支持登录名仅 CREATE/DROP/ALTER 任何数据库	将登录名添加到 dbcreator 固定服务器角色	ALTER SERVER ROLE dbcreator ADD MEMBER `login`	只能更改 dbcreator 登录名有权访问的那些数据库。	版本 4.5 及更高版本、5.1 及更高版本
支持登录名仅 CREATE/ALTER/DROP 任何登录名	将登录名添加到 securityadmin 固定服务器角色	ALTER SERVER ROLE securityadmin ADD MEMBER `login`	无	版本 4.5 及更高版本、5.1 及更高版本

不支持的使用案例以及解决办法

下表说明了 Babelfish 不支持但可以使用解决方法实现的使用案例。

应用场景	操作	SQL 命令	评论	Babelfish 版本兼容性解决方法
支持用户对对象/架构执行 SELECT/DML 以及向其他用户	直接向所有其他用户 GRANT 权限	GRANT SELECT/EXECUTE/INSERT/UPDATE/	GRANT ... WITH GRANT OPTION 当前不受支持。	版本 3.6 及更高版本、4.2 及更高版本

应用场景	操作	SQL 命令	评论	Babelfish 版本兼容性解决方法
GRANT 这些权限的选项		DELETE ON OBJECT/SCHEMA::object/sc TO user		
支持数据库用户在数据库中执行 SELECT/DML/DDL，包括删除数据库	将角色的成员添加到 db_owner 固定数据库角色	ALTER ROLE db_owner ADD MEMBER user	当前不支持将角色添加到 db_owner 角色。	版本 4.5 及更高版本、5.1 及更高版本

不支持的使用案例

下表说明了 Babelfish 不支持的使用案例。

应用场景	评论
拒绝自定义数据库角色的用户或成员在数据库中执行 SELECT	尚不支持 db_denydatareader 固定数据库角色
拒绝自定义数据库角色的用户或成员在数据库中执行 DML	当前不支持 db_denydatawriter 固定数据库角色。
支持登录名仅 KILL 任何数据库连接	当前不支持 processadmin 固定服务器角色。
支持登录名仅添加或移除链接服务器	当前不支持 setupadmin 固定服务器角色。

解决升级后的对象所有权差异

Babelfish 版本 4.6 及更高版本和版本 5.2 及更高版本包括一项更改，即通过 TDS 端点处理对象所有权。当您通过 TDS 端点创建新对象时，这些对象现在归架构所有者拥有，而不归当前用户拥有。当您从 4.6 或 5.2 之前的版本升级时，与现有对象相比，此所有权更改可能会影响新对象的权限行为。

为了解决这些所有权差异，Babelfish 提供了 `sys.generate_alter_ownership_statements()` 函数。此函数生成 SQL 语句，这些语句使对象所有权与架构所有权保持一致。

解决对象所有权时，请注意以下限制：

- 通过 PostgreSQL 端点获得 CREATE 权限的用户无法在这些架构中通过 TDS 端点创建对象。
- 建议不要通过 PostgreSQL 端点修改对 T-SQL 对象的权限，这可能会导致 T-SQL 行为不正确。
- 由于新旧对象的所有权不匹配，它们的访问权限可能会有所不同。例如，考虑一个由 sch_own 拥有的架构，该架构包括由 dbo 拥有的对象。在这种情况下，与在升级后创建的由 sch_own 拥有的对象相比，在升级前创建的由 dbo 拥有的对象可能具有不同的访问权限。这可能会影响诸如 SELECT 和 INSERT 之类的操作。

如果数据库集群包含在 4.6 或 5.2 之前的 Babelfish 版本中创建的对象，请考虑使其所有权保持一致。

解决对象所有权差异

1. 使用 PostgreSQL 端点连接到数据库集群中的 babelfish_db 数据库。
2. 运行以下命令：

   ```
   SELECT * from sys.generate_alter_ownership_statements();
   ```

   这会生成 SQL 语句的列表，这些语句旨在实现对象间所有权的标准化。
3. 先在测试环境中执行生成的语句以验证其效果，然后再应用于生产环境。

我们建议您执行这些语句，以便在整个数据库中实现一致的对象所有权模型。

适用于 Aurora PostgreSQL 的 Babelfish 与 SQL Server 之间的区别

Babelfish 是一项不断发展的 Aurora PostgreSQL 功能，自 Aurora PostgreSQL 13.4 首次推出以来，每个版本都添加了新功能。它旨在使用 TDS 端口通过 T-SQL 方言在 PostgreSQL 之上提供 T-SQL 语义。Babelfish 的每个新版本都添加了可更好地与 T-SQL 功能和行为保持一致的特性和功能，如 Babelfish 的各个版本支持的功能表中所示。为了在使用 Babelfish 时获得最佳效果，我们建议您了解 SQL Server 与最新版本的 Babelfish 支持的 T-SQL 之间目前存在的区别。要了解更多信息，请参阅Babelfish 中的 T-SQL 差异。

除了 Babelfish 和 SQL Server 支持的 T-SQL 之间的区别之外，您可能还需要在 Aurora PostgreSQL 数据库集群的背景下考虑 Babelfish 和 PostgreSQL 之间的互操作性问题。如前所述，Babelfish 使用 TDS 端口通过 T-SQL 方言在 PostgreSQL 之上支持 T-SQL 语义。同时，也可以使用 PostgreSQL SQL 语句通过标准 PostgreSQL 端口访问 Babelfish 数据库。如果您正在考虑在生产部署中同时使用 PostgreSQL 和 Babelfish 功能，则需要意识到架构名称、标识符、权限、事务语义、多个结果集、原

定设置排序规则等之间潜在的互操作性问题。简而言之，当 PostgreSQL 语句或 PostgreSQL 访问发生在 Babelfish 的上下文中时，在新版本的 Babelfish 发布时，PostgreSQL 和 Babelfish 之间可能会发生干扰并可能影响语法、语义和兼容性。有关所有注意事项的完整信息和指导，请参阅 Babelfish for PostgreSQL 文档中的 Babelfish 互操作性指南。

> **Note**
>
> 在同一个应用程序上下文中同时使用 PostgreSQL 原生功能和 Babelfish 功能之前，我们强烈建议您考虑在 Babelfish for PostgreSQL 文档的 Babelfish 互操作性指南中讨论的问题。只有当您计划在与 Babelfish 相同的应用程序上下文中使用 PostgreSQL 数据库实例时，这些互操作性问题（Aurora PostgreSQL 和 Babelfish）才有意义。

主题

- Babelfish 转储和还原
- Babelfish 中的 T-SQL 差异
- Babelfish 中的事务隔离级别

Babelfish 转储和还原

从版本 4.0.0 和 3.4.0 开始，Babelfish 用户现在可以使用转储和还原实用程序来备份和还原其数据库。有关更多信息，请参阅 Babelfish dump and restore。此功能基于 PostgreSQL 转储和还原实用程序而构建。有关更多信息，请参阅 pg_dump 和 pg_restore。为了在 Babelfish 中有效地使用此功能，您需要使用专门适用于 Babelfish 的基于 PostgreSQL 的工具。Babelfish 的备份和还原功能与 SQL Server 的备份和还原功能有很大不同。有关这些差异的更多信息，请参阅 Dump and restore functionality differences: Babelfish and SQL Server。Babelfish for Aurora PostgreSQL 为备份和还原 Amazon Aurora PostgreSQL 数据库集群提供额外功能。有关更多信息，请参阅 备份和还原 Amazon Aurora 数据库集群。

Babelfish 中的 T-SQL 差异

下面，您可以找到当前版本 Babelfish 支持的 T-SQL 功能表，其中包含一些关于行为与 SQL Server 行为差异的注释。

有关各种版本支持的更多信息，请参阅Babelfish 的各个版本支持的功能。有关当前不支持功能的信息，请参阅Babelfish 中不支持的功能。

Babelfish 可用于 Aurora PostgreSQL 兼容版。有关 Babelfish 版本的更多信息，请参阅 Aurora PostgreSQL 版本注释。

功能或语法	行为或差异的描述
\（行延续字符）	当前不支持字符串和十六进制字符串的行延续字符（换行符前的反斜杠）。对于字符串，反斜杠换行符被解释为字符串中的字符。对于十六进制字符串，反斜杠换行符会导致语法错误。
@@version	`@@version` 返回的值的格式与 SQL Server 返回的值略有不同。如果您的代码取决于 `@@version` 的格式化，可能无法正常工作。
聚合函数	部分支持聚合函数（支持 AVG、COUNT、COUNT_BIG、GROUPING、MAX、MIN、STRING_AGG 和 SUM）。有关不支持聚合函数的列表，请参阅不支持的函数。
ALTER TABLE	仅支持添加或删除单个列或约束。
ALTER TABLE..ALTER COLUMN	目前无法指定 NULL 和 NOT NULL。要更改列是否可为 null 值，请使用 PostgreSQL 语句 ALTER TABLE..{SET|DROP} NOT NULL。

AT TIME ZONE	在从夏令时（DST）过渡到标准时间期间，将使用标准时间偏移显示重叠的时段。为了澄清，请考虑以下示例：
	`SELECT CONVERT(DATETIME2(0), '2022-10-30T02:00:00', 126) AT TIME ZONE 'Central European Standard Time';`
	`GO;`
	`Result: 2022-10-30 02:00:00 +01:00`

没有列别名的空白列名	`sqlcmd` 和 `psql` 实用程序以不同方式处理空名的列： • SQL Server `sqlcmd` 返回空白列名。 • PostgreSQL `psql` 返回生成的列名称。

功能或语法	行为或差异的描述
CHECKSUM 函数	Babelfish 和 SQL Server 对 CHECKSUM 函数使用不同的哈希算法。因此，由 Babelfish 中的 CHECKSUM 函数生成的哈希值可能与由 SQL Server 中的 CHECKSUM 函数生成的哈希值不同。
列原定设置	创建列原定设置时，约束名称将被忽略。要删除列原定设置，请使用以下语法：`ALTER TABLE...ALTER COLUMN..DROP DEFAULT...`
Constraint_name	在 SQL Server 中，约束名称在表所属的架构中必须是唯一的。但是，在 Babelfish 中，这仅适用于 PRIMARY KEY 和 UNIQUE 约束。其它类型的约束不受此限制所限。
约束	PostgreSQL 不支持打开和关闭单个约束条件。将忽略语句，并发出警告。
具有 IGNORE_DUP_KEY 的约束	不使用此属性创建约束。
CREATE、ALTER、DROP SERVER ROLE	ALTER SERVER ROLE 仅支持 `sysadmin`。不支持所有其他语法。 对于登录（服务器主体）、数据库和数据库用户（数据库主体）的概念，Babelfish 中的 T-SQL 用户的体验类似于 SQL Server。
CREATE、ALTER LOGIN 子句支持有限的语法	支持 CREATE LOGIN... PASSWORD 子句、...DEFAULT_DATABASE 子句和 ...DEFAULT_LANGUAGE 子句。支持 ALTER LOGIN... PASSWORD 子句，但 ALTER LOGIN... OLD_PASSWORD 子句不受支持。只有系统管理员成员的登录名才能修改密码。
CREATE DATABASE 区分大小写的排序规则	CREATE DATABASE 语句不支持区分大小写的排序规则。
CREATE DATABASE 关键字和子句	不支持 COLLATE 和 CONTAINMENT=NONE 以外的选项。COLLATE 子句被接受并且始终设置为 `babelfishpg_tsql.server_collation_name` 的值。

功能或语法	行为或差异的描述
CREATE SCHEMA... 支持子句	您可以使用 CREATE SCHEMA 命令创建空架构。使用其他命令创建架构对象。
Babelfish 上的数据库 ID 值不同	主数据库和 tempdb 数据库将不是数据库 ID 1 和 2。
支持 FORMAT 日期类型函数，但存在以下限制	不支持单字符子午线。 对于大于 1000 的年份，SQL Server 中的“yyy”格式返回 4 位数，但对于其他年份，只返回 3 位数字。 不支持“g”和“R”格式 “vi-VN”区域设置翻译略有不同。
超过 63 个字符的标识符	PostgreSQL 最多为标识符支持 63 个字符。Babelfish 将长度超过 63 个字符的标识符转换为包含原名哈希的名称。例如，创建为“AB(ABC12345678901234567890123456789012345678901234567890123456789012345678901234567890”的表可能转换为“ABC123456789012345678901234567890123456789012345678901234567890”。
IDENTITY 列支持	`IDENTITY` 列支持数据类型 `tinyint`、`smallint`、`int`、`bigint`。`numeric` 和 `decimal`。 SQL Server 为 IDENTITY 列中的数据类型 `numeric` 和 `decimal` 支持的精度达到 38 位。 PostgreSQL 为 IDENTITY 列中的数据类型 `numeric` 和 `decimal` 支持的精度达到 19 位。
使用 IGNORE_DUP_KEY 的索引	创建包含 IGNORE_DUP_KEY 的索引的语法会创建一个索引，就像省略此属性一样。
包含 32 列以上的索引	索引不能包含超过 32 列。包含的索引列在 PostgreSQL 中计入最大值，但在 SQL Server 中不计。

功能或语法	行为或差异的描述
索引（聚集）	聚集索引的创建就像指定了 NONCLUSTERED 一样。
索引子句	忽略以下子句：FILLFACTOR、ALLOW_PAGE_LOCKS、ALLOW_ROW_LOCKS、PAD_INDEX、STATISTICS_NORECOMPUTE、OPTIMIZE_FOR_SEQUENTIAL_KEY、SORT_IN_TEMPDB、DROP_EXISTING、ONLINE、COMPRESSION_DELAY、MAXDOP 和 DATA_COMPRESSION
JSON 支持	不能保证名称/值对的顺序。但数组类型保持不受影响。
LOGIN 对象	不支持 LOGIN 对象以外的所有选项，但以下各项除外：PASSWORD、DEFAULT_DATABASE、DEFAULT_LANGUAGE、ENABLE、DISABLE。
NEWSEQUENTIALID 函数	作为 NEWID 实施；不保证顺序行为。调用 `NEWSEQUENTIALID` 时，PostgreSQL 会生成一个新的 GUID 值。
支持 OUTPUT 子句，但存在以下限制	同一个 DML 查询中不支持 OUTPUT 和 OUTPUT INTO。不支持在 OUTPUT 子句中对 UPDATE 或 DELETE 操作的非目标表进行引用。OUTPUT... DELETED *、INSERTED * 在同一个查询中不受支持。
过程或函数参数限制	Babelfish 最多支持 100 个过程或函数的参数。
ROWGUIDCOL	此子句当前被忽略。引用 `$GUIDGOL` 的查询导致语法错误。
SEQUENCE 对象支持	数据类型 tinyint、smallint、int、bigint、numeric 和 decimal 支持 SEQUENCE 对象。 对于 SEQUENCE 中的数据类型 numeric 和 decimal，Aurora PostgreSQL 支持的精度达到 19 位。
服务器级角色	支持 `sysadmin` 服务器级角色。不支持 `sysadmin` 以外的其他服务器级角色。

功能或语法	行为或差异的描述
除 db_owner 以外的数据库级别角色	支持 db_owner 数据库级角色和用户定义的数据库级角色。不支持 db_owner 以外的其他服务器级角色。
SQL 关键字 SPARSE	接受并忽略关键字 SPARSE。
SQL 关键字子句 ON filegroup	此子句当前被忽略。
索引和约束的 SQL 关键字 CLUSTERED 和 NONCLUSTERED	Babelfish 接受并忽略 CLUSTERED 和 NONCLUSTERED 关键字。
sysdatabases.cmptlevel	sysdatabases.cmptlevel 始终设置为 120。
tempdb 在重启时没有重新初始化	重新启动数据库时，不会删除在 tempdb 中创建的永久对象（如表和过程）。
TEXTIMAGE_ON 文件组	Babelfish 忽略 TEXTIMAGE_ON *filegroup* 子句。
时间精度	Babelfish 对小数秒支持 6 位数的精度。预计这种行为不会产生负面影响。
事务隔离级别	以与 READCOMMITTED 相同的方式对待 READUNCOMMITTED。
虚拟计算列（非持久性）	虚拟计算列是作为永久列创建的。
无 SCHEMABINDING 子句	函数、过程、触发器或视图不支持此子句。已创建对象，但仿佛指定了 WITH SCHEMABINDING。

Babelfish 中的事务隔离级别

Babelfish 支持事务隔离级别 READ UNCOMMITTED、READ COMMITTED 和 SNAPSHOT。从 Babelfish 3.4 版本开始，支持额外的隔离级别 REPEATABLE READ 和 SERIALIZABLE。PostgreSQL 中相应隔离级别的行为支持 Babelfish 中的所有隔离级别。SQL Server 和 Babelfish 使用不同的底层机制来实现事务隔离级别（阻止并发访问、事务持有的锁、错误处理等）。而且，不同工作负载的并发访问方式可能存在一些细微差异。有关此 PostgreSQL 行为的更多信息，请参阅 Transaction Isolation。

主题

- 事务隔离级别概述
- 设置事务隔离级别
- 启用或禁用事务隔离级别
- 比较 Babelfish 与 SQL Server 隔离级别

事务隔离级别概述

最初的 SQL Server 事务隔离级别基于保守锁，其中只存在一个数据副本，查询在访问行等资源之前必须锁定这些资源。后来，引入了 READ COMMITTED 隔离级别的变体。这使得行版本的使用能够在使用非阻止访问的读取器和写入器之间提供更好的并发性。此外，还提供了一个名为 SNAPSHOT 的新隔离级别。它也使用行版本来提供比 REPEATABLE READ 隔离级别更好的并发性，方法是避免对读取数据使用共享锁，这些锁一直保留直至事务结束。

与 SQL Server 不同，Babelfish 中的所有事务隔离级别都基于乐观锁（MVCC）。无论底层数据的当前状态如何，每个事务都可以在语句（READ COMMITTED）的开头或事务（REPEATABLE READ、SERIALIZABLE）的开头看到数据的快照。因此，Babelfish 中并发事务的执行行为可能与 SQL Server 不同。

例如，设想有一个隔离级别为 SERIALIZABLE 的事务，该事务最初在 SQL Server 中被阻止，但后来成功了。由于与读取或更新相同行的并发事务存在序列化冲突，该事务最终可能会在 Babelfish 中失败。在某些情况下，与 SQL Server 相比，在 Babelfish 中执行多个并发事务可能会产生不同的最终结果。对于使用隔离级别的应用程序，应针对并发场景进行全面测试。

SQL Server 中的隔离级别	Babelfish 隔离级别	PostgreSQL 隔离级别	评论
READ UNCOMMITTED	READ UNCOMMITTED	READ UNCOMMITTED	READ UNCOMMITTED 与 Babelfish 或

SQL Server 中的隔离级别	Babelfish 隔离级别	PostgreSQL 隔离级别	评论
			PostgreSQL 中的 READ COMMITTED 相同
READ COMMITTED	READ COMMITTED	READ COMMITTED	SQL Server READ COMMITTED 基于保守锁，Babelfish READ COMMITTED 基于快照（MVCC）。
READ COMMITTED SNAPSHOT	READ COMMITTED	READ COMMITTED	两者都基于快照（MVCC），但并不完全相同。
SNAPSHOT	SNAPSHOT	REPEATABLE READ	完全相同。
REPEATABLE READ	REPEATABLE READ	REPEATABLE READ	SQL Server REPEATABLE READ 基于保守锁，Babelfish REPEATABLE READ 基于快照（MVCC）。
SERIALIZABLE	SERIALIZABLE	SERIALIZABLE	SQL Server SERIALIZABLE 是保守隔离，Babelfish SERIALIZABLE 基于快照（MVCC）。

> **Note**
>
> 目前不支持表提示，其行为是通过使用 Babelfish 预定义的备用方案 `escape_hatch_table_hints` 来控制的。

设置事务隔离级别

使用以下命令设置事务隔离级别：

Example

```
SET TRANSACTION ISOLATION LEVEL { READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ |
 SNAPSHOT | SERIALIZABLE }
```

启用或禁用事务隔离级别

在 Babelfish 中，事务隔离级别 REPEATABLE READ 和 SERIALIZABLE 默认处于禁用状态，您必须使用 sp_babelfish_configure 将 babelfishpg_tsql.isolation_level_serializable 或 babelfishpg_tsql.isolation_level_repeatable_read 备用方案设置为 pg_isolation 来显式启用它们。有关更多信息，请参阅 使用转义孵化管理 Babelfish 错误处理。

以下是通过设置各自的备用方案来启用或禁用在当前会话中使用 REPEATABLE READ 和 SERIALIZABLE 的示例。（可选）包括 server 参数，用于为当前会话以及所有后续新会话设置备用方案。

仅允许在当前会话中使用 SET TRANSACTION ISOLATION LEVEL REPEATABLE READ。

Example

```
EXECUTE sp_babelfish_configure 'isolation_level_repeatable_read', 'pg_isolation'
```

允许在当前会话和所有后续新会话中使用 SET TRANSACTION ISOLATION LEVEL REPEATABLE READ。

Example

```
EXECUTE sp_babelfish_configure 'isolation_level_repeatable_read', 'pg_isolation',
 'server'
```

禁止在当前会话和后续新会话中使用 SET TRANSACTION ISOLATION LEVEL REPEATABLE READ。

Example

```
EXECUTE sp_babelfish_configure 'isolation_level_repeatable_read', 'off', 'server'
```

仅允许在当前会话中使用 SET TRANSACTION ISOLATION LEVEL SERIALIZABLE。

Example

```
EXECUTE sp_babelfish_configure 'isolation_level_serializable', 'pg_isolation'
```

允许在当前会话和所有后续新会话中使用 SET TRANSACTION ISOLATION LEVEL SERIALIZABLE。

Example

```
EXECUTE sp_babelfish_configure 'isolation_level_serializable', 'pg_isolation', 'server'
```

禁止在当前会话和后续新会话中使用 SET TRANSACTION ISOLATION LEVEL SERIALIZABLE。

Example

```
EXECUTE sp_babelfish_configure 'isolation_level_serializable', 'off', 'server'
```

比较 Babelfish 与 SQL Server 隔离级别

以下是一些关于 SQL Server 和 Babelfish 如何实现 ANSI 隔离级别的细微差别的示例。

> **Note**
>
> - 隔离级别 REPEATABLE READ 和 SNAPSHOT 在 Babelfish 中相同。
> - 隔离级别 READ UNCOMMITTED 和 READ COMMITTED 在 Babelfish 中相同。

以下示例显示了如何为下面提到的所有示例创建基表：

```
CREATE TABLE employee (
```

```
    id sys.INT NOT NULL PRIMARY KEY,
    name sys.VARCHAR(255)NOT NULL,
    age sys.INT NOT NULL
);
INSERT INTO employee (id, name, age) VALUES (1, 'A', 10);
INSERT INTO employee (id, name, age) VALUES (2, 'B', 20);
INSERT INTO employee (id, name, age) VALUES (3, 'C', 30);
```

主题

- Babelfish READ UNCOMMITTED 与 SQL Server READ UNCOMMITTED 隔离级别的比较
- Babelfish READ COMMITTED 与 SQL Server READ COMMITTED 隔离级别的比较
- Babelfish READ COMMITTED 与 SQL Server READ COMMITTED SNAPSHOT 隔离级别的比较
- Babelfish REPEATABLE READ 与 SQL Server REPEATABLE READ 隔离级别的比较
- Babelfish SERIALIZABLE 与 SQL Server SERIALIZABLE 隔离级别的比较

Babelfish **READ UNCOMMITTED** 与 SQL Server **READ UNCOMMITTED** 隔离级别的比较

下表详细介绍了执行并发事务时的脏读。它显示了与 Babelfish 实现相比，在 SQL Server 中使用 READ UNCOMMITTED 隔离级别时观察到的结果。

事务 1	事务 2	SQL Server **READ UNCOMMITTED**	Babelfish **READ UNCOMMITTED**
`BEGIN TRANSACTION`	`BEGIN TRANSACTION`	无	无
`SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;`	`SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;`	无	无
空闲事务	`UPDATE employee SET age=0;`	更新成功。	更新成功。
空闲事务	`INSERT INTO employee VALUES (4, 'D', 40);`	插入成功。	插入成功。

事务 1	事务 2	SQL Server **READ UNCOMMITTED**	Babelfish **READ UNCOMMITTED**
`SELECT * FROM employee;`	空闲事务	事务 1 可以看到事务 2 中未提交的更改。	与 Babelfish 中的 `READ COMMITTED` 相同。事务 2 中未提交的更改对事务 1 不可见。
空闲事务	`COMMIT`	无	无
`SELECT * FROM employee;`	空闲事务	看到事务 2 提交的更改。	看到事务 2 提交的更改。

Babelfish **`READ COMMITTED`** 与 SQL Server **`READ COMMITTED`** 隔离级别的比较

下表详细介绍了执行并发事务时的读取-写入阻止行为。它显示了与 Babelfish 实现相比，在 SQL Server 中使用 `READ COMMITTED` 隔离级别时观察到的结果。

事务 1	事务 2	SQL Server **READ COMMITTED**	Babelfish **READ COMMITTED**
`BEGIN TRANSACTION`	`BEGIN TRANSACTION`	无	无
`SET TRANSACTION ISOLATION LEVEL READ COMMITTED;`	`SET TRANSACTION ISOLATION LEVEL READ COMMITTED;`	无	无
`SELECT * FROM employee;`	空闲事务	无	无
空闲事务	`UPDATE employee SET age=100 WHERE id = 1;`	更新成功。	更新成功。
`UPDATE employee SET age = 0`	空闲事务	在事务 2 提交之前，步骤被阻止。	事务 2 的更改尚不可见。更新 id=3 的行。

事务 1	事务 2	SQL Server **READ COMMITTED**	Babelfish **READ COMMITTED**
`WHERE age IN (SELECT MAX(age) FROM employee);`			
空闲事务	`COMMIT`	事务 2 成功提交。事务 1 现已解除阻止，并且可以看到事务 2 的更新。	事务 2 成功提交。
`SELECT * FROM employee;`	空闲事务	事务 1 更新 id = 1 的行。	事务 1 更新 id = 3 的行。

Babelfish **READ COMMITTED** 与 SQL Server **READ COMMITTED SNAPSHOT** 隔离级别的比较

下表详细介绍了执行并发事务时新插入行的阻止行为。它显示了与 `READ COMMITTED` Babelfish 实现相比，在 SQL Server 中使用 `READ COMMITTED SNAPSHOT` 隔离级别时观察到的结果。

事务 1	事务 2	SQL Server **READ COMMITTED SNAPSHOT**	Babelfish **READ COMMITTED**
`BEGIN TRANSACTION`	`BEGIN TRANSACTION`	无	无
`SET TRANSACTION ISOLATION LEVEL READ COMMITTED;`	`SET TRANSACTION ISOLATION LEVEL READ COMMITTED;`	无	无
`INSERT INTO employee VALUES (4, 'D', 40);`	空闲事务	无	无
空闲事务	`UPDATE employee SET age = 99;`	在事务 1 提交之前，步骤会被阻止。插入的行被事务 1 锁定。	更新了三行。新插入的行尚不可见。

事务 1	事务 2	SQL Server **READ COMMITTED SNAPSHOT**	Babelfish **READ COMMITTED**
COMMIT	空闲事务	提交成功。事务 2 现已解除阻止。	提交成功。
空闲事务	SELECT * FROM employee;	所有 4 行都具有 age=99。	id = 4 的行具有年龄值 40，因为在更新查询期间，事务 2 看不到该行。其他行更新为 age=99。

Babelfish **REPEATABLE READ** 与 SQL Server **REPEATABLE READ** 隔离级别的比较

下表详细介绍了执行并发事务时的读取-写入阻止行为。它显示了与 REPEATABLE READ Babelfish 实现相比，在 SQL Server 中使用 REPEATABLE READ 隔离级别时观察到的结果。

事务 1	事务 2	SQL Server **REPEATABLE READ**	Babelfish **REPEATABLE READ**
BEGIN TRANSACTION	BEGIN TRANSACTION	无	无
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;	SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;	无	无
SELECT * FROM employee;	空闲事务	无	无
UPDATE employee SET name='A_TXN1' WHERE id=1;	空闲事务	无	无

事务 1	事务 2	SQL Server **REPEATABLE READ**	Babelfish **REPEATABLE READ**
空闲事务	`SELECT * FROM employee WHERE id != 1;`	无	无
空闲事务	`SELECT * FROM employee;`	在事务 1 提交之前，事务 2 将被阻止。	事务 2 正常进行。
`COMMIT`	空闲事务	无	无
空闲事务	`SELECT * FROM employee;`	事务 1 中的更新可见。	事务 1 中的更新不可见。
`COMMIT`	空闲事务	无	无
空闲事务	`SELECT * FROM employee;`	看到事务 1 中的更新。	看到事务 1 中的更新。

下表详细介绍了执行并发事务时的写入-写入阻止行为。它显示了与 `REPEATABLE READ` Babelfish 实现相比，在 SQL Server 中使用 `REPEATABLE READ` 隔离级别时观察到的结果。

事务 1	事务 2	SQL Server **REPEATABLE READ**	Babelfish **REPEATABLE READ**
`BEGIN TRANSACTION`	`BEGIN TRANSACTION`	无	无
`SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;`	`SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;`	无	无
`UPDATE employee SET name='A_T`	空闲事务	无	无

事务 1	事务 2	SQL Server **REPEATABLE READ**	Babelfish **REPEATABLE READ**
`XN1' WHERE id=1;`			
空闲事务	`UPDATE employee SET name='A_TXN2' WHERE id=1;`	事务 2 已阻止。	事务 2 已阻止。
`COMMIT`	空闲事务	提交成功且事务 2 已解除阻止。	提交成功，事务 2 失败并出现错误，原因是由于并发更新而导致无法序列化访问。
空闲事务	`COMMIT`	提交成功。	事务 2 已中止。
空闲事务	`SELECT * FROM employee;`	id=1 的行具有 name='A_TX2'。	id=1 的行具有 name='A_TX1'。

下表详细介绍了执行并发事务时的幻读行为。它显示了与 `REPEATABLE READ` Babelfish 实现相比，在 SQL Server 中使用 `REPEATABLE READ` 隔离级别时观察到的结果。

事务 1	事务 2	SQL Server **REPEATABLE READ**	Babelfish **REPEATABLE READ**
`BEGIN TRANSACTION`	`BEGIN TRANSACTION`	无	无
`SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;`	`SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;`	无	无
`SELECT * FROM employee;`	空闲事务	无	无

事务 1	事务 2	SQL Server **REPEATABLE READ**	Babelfish **REPEATABLE READ**
空闲事务	`INSERT INTO employee VALUES (4, 'NewRowName', 20);`	事务 2 在没有任何阻止的情况下继续进行。	事务 2 在没有任何阻止的情况下继续进行。
空闲事务	`SELECT * FROM employee;`	新插入的行可见。	新插入的行可见。
空闲事务	`COMMIT`	无	无
`SELECT * FROM employee;`	空闲事务	事务 2 插入的新行可见。	事务 2 插入的新行不可见。
`COMMIT`	空闲事务	无	无
`SELECT * FROM employee;`	空闲事务	新插入的行可见。	新插入的行可见。

下表详细介绍了执行并发事务时的情景，以及与 `REPEATABLE READ` Babelfish 实现相比，在 SQL Server 中使用 `REPEATABLE READ` 隔离级别时的不同最终结果。

事务 1	事务 2	SQL Server **REPEATABLE READ**	Babelfish **REPEATABLE READ**
`BEGIN TRANSACTION`	`BEGIN TRANSACTION`	无	无
`SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;`	`SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;`	无	无
`UPDATE employee SET age = 100 WHERE age IN`	空闲事务	事务 1 更新 id 为 1 的行。	事务 1 更新 id 为 1 的行。

事务 1	事务 2	SQL Server **REPEATABLE READ**	Babelfish **REPEATABLE READ**
`(SELECT MIN(age) FROM employee);`			
空闲事务	`UPDATE employee SET age = 0 WHERE age IN (SELECT MAX(age) FROM employee);`	由于 SELECT 语句尝试读取事务 1 中由 UPDATE 查询锁定的行，因此事务 2 被阻止。	事务 2 在没有任何阻止的情况下继续进行，因为读取从不会被阻止，SELECT 语句执行，最后更新了 id = 3 的行，因为事务 1 更改尚不可见。
空闲事务	`SELECT * FROM employee;`	此步骤在事务 1 提交后执行。id = 1 的行在上一步中由事务 2 更新，此处可见。	id = 3 的行由事务 2 更新。
`COMMIT`	空闲事务	事务 2 现已解除阻止。	提交成功。
空闲事务	`COMMIT`	无	无
`SELECT * FROM employee;`	空闲事务	两个事务都对 id = 1 的行执行更新。	事务 1 和 2 更新不同的行。

Babelfish **SERIALIZABLE** 与 SQL Server **SERIALIZABLE** 隔离级别的比较

下表详细介绍了执行并发事务时的范围锁。它显示了与 SERIALIZABLE Babelfish 实现相比，在 SQL Server 中使用 SERIALIZABLE 隔离级别时观察到的结果。

事务 1	事务 2	SQL Server **SERIALIZABLE**	Babelfish **SERIALIZABLE**
`BEGIN TRANSACTION`	`BEGIN TRANSACTION`	无	无

事务 1	事务 2	SQL Server **SERIALIZABLE**	Babelfish **SERIALIZABLE**
SET TRANSACTION ISOLATION LEVEL SERILAIZABLE;	SET TRANSACTION ISOLATION LEVEL SERILAIZABLE;	无	无
SELECT * FROM employee;	空闲事务	无	无
空闲事务	INSERT INTO employee VALUES (4, 'D', 35);	在事务 1 提交之前，事务 2 将被阻止。	事务 2 在没有任何阻止的情况下继续进行。
空闲事务	SELECT * FROM employee;	无	无
COMMIT	空闲事务	事务 1 成功提交。事务 2 现已解除阻止。	事务 1 成功提交。
空闲事务	COMMIT	无	无
SELECT * FROM employee;	空闲事务	新插入的行可见。	新插入的行可见。

下表详细介绍了执行并发事务时的情景，以及与 SERIALIZABLE Babelfish 实现相比，在 SQL Server 中使用 SERIALIZABLE 隔离级别时的不同最终结果。

事务 1	事务 2	SQL Server **SERIALIZABLE**	Babelfish **SERIALIZABLE**
BEGIN TRANSACTION	BEGIN TRANSACTION	无	无
SET TRANSACTION ISOLATION LEVEL SERILAIZABLE;	SET TRANSACTION ISOLATION LEVEL SERILAIZABLE;	无	无

事务 1	事务 2	SQL Server **SERIALIZABLE**	Babelfish **SERIALIZABLE**
空闲事务	`INSERT INTO employee VALUES (4, 'D', 40);`	无	无
`UPDATE employee SET age =99 WHERE id = 4;`	空闲事务	在事务 2 提交之前，事务 1 将被阻止。	事务 1 在没有任何阻止的情况下继续进行。
空闲事务	`COMMIT`	事务 2 成功提交。事务 1 现已解除阻止。	事务 2 成功提交。
`COMMIT`	空闲事务	无	无
`SELECT * FROM employee;`	空闲事务	可以看到新插入的年龄值 = 99 的行。	可以看到新插入的年龄值 = 40 的行。

下表详细介绍了当您对具有唯一约束的表执行 INSERT 时的情景。它显示了与 SERIALIZABLE Babelfish 实现相比，在 SQL Server 中使用 SERIALIZABLE 隔离级别时观察到的结果。

事务 1	事务 2	SQL Server **SERIALIZABLE**	Babelfish **SERIALIZABLE**
`BEGIN TRANSACTION`	`BEGIN TRANSACTION`	无	无
`SET TRANSACTION ISOLATION LEVEL SERILAIZABLE;`	`SET TRANSACTION ISOLATION LEVEL SERILAIZABLE;`	无	无
空闲事务	`INSERT INTO employee VALUES (4, 'D', 40);`	无	无
`INSERT INTO employee`	空闲事务	在事务 2 提交之前，事务 1 将被阻止。	在事务 2 提交之前，事务 1 将被阻止。

事务 1	事务 2	SQL Server **SERIALIZABLE**	Babelfish **SERIALIZABLE**
`VALUES ((SELECT MAX(id)+1 FROM employee), 'E', 50);`			
空闲事务	`COMMIT`	事务 2 成功提交。事务 1 现已解除阻止。	事务 2 成功提交。事务 1 中止，出现错误，原因是重复键值违反了唯一约束。
`COMMIT`	空闲事务	事务 1 成功提交。	事务 1 提交失败，原因是由于事务之间的读取或写入依赖关系，无法序列化访问。
`SELECT * FROM employee;`	空闲事务	插入行 (5, 'E', 50)。	仅存在 4 行。

在 Babelfish 中，如果这些事务的执行与这些事务的所有可能的串行（一次一个）执行不一致，则以隔离级别 Serializable 运行的并发事务将失败，并出现序列化异常错误。

下表详细介绍了执行并发事务时的序列化异常。它显示了与 `SERIALIZABLE` Babelfish 实现相比，在 SQL Server 中使用 `SERIALIZABLE` 隔离级别时观察到的结果。

事务 1	事务 2	SQL Server **SERIALIZABLE**	Babelfish **SERIALIZABLE**
`BEGIN TRANSACTION`	`BEGIN TRANSACTION`	无	无
`SET TRANSACTION ISOLATION LEVEL SERILAIZABLE;`	`SET TRANSACTION ISOLATION LEVEL SERILAIZABLE;`	无	无

事务 1	事务 2	SQL Server **SERIALIZABLE**	Babelfish **SERIALIZABLE**
`SELECT * FROM employee;`	空闲事务	无	无
`UPDATE employee SET age=5 WHERE age=10;`	空闲事务	无	无
空闲事务	`SELECT * FROM employee;`	在事务 1 提交之前，事务 2 将被阻止。	事务 2 在没有任何阻止的情况下继续进行。
空闲事务	`UPDATE employee SET age=35 WHERE age=30;`	无	无
`COMMIT`	空闲事务	事务 1 成功提交。	事务 1 首先提交，并且能够成功提交。
空闲事务	`COMMIT`	事务 2 成功提交。	事务 2 提交失败并出现序列化错误，整个事务已回滚。重试事务 2。
`SELECT * FROM employee;`	空闲事务	这两个事务的更改都是可见的。	事务 2 已回滚。只能看到事务 1 更改。

在 Babelfish 中，只有当所有并发事务都以 SERIALIZABLE 隔离级别执行时，才可能出现序列化异常。在下表中，我们以上面的示例为例，但将事务 2 改为隔离级别 REPEATABLE READ。

事务 1	事务 2	SQL Server 隔离级别	Babelfish 隔离级别
`BEGIN TRANSACTION`	`BEGIN TRANSACTION`	无	无

事务 1	事务 2	SQL Server 隔离级别	Babelfish 隔离级别
`SET TRANSACTION ISOLATION LEVEL SERILAIZABLE;`	`SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;`	无	无
`SELECT * FROM employee;`	空闲事务	无	无
`UPDATE employee SET age=5 WHERE age=10;`	空闲事务	无	无
空闲事务	`SELECT * FROM employee;`	在事务 1 提交之前，事务 2 将被阻止。	事务 2 在没有任何阻止的情况下继续进行。
空闲事务	`UPDATE employee SET age=35 WHERE age=30;`	无	无
`COMMIT`	空闲事务	事务 1 成功提交。	事务 1 成功提交。
空闲事务	`COMMIT`	事务 2 成功提交。	事务 2 成功提交。
`SELECT * FROM employee;`	空闲事务	这两个事务的更改都是可见的。	这两个事务的更改都是可见的。

使用具有有限实施的 Babelfish 功能

Babelfish 的每个新版本都增加了对某些功能的支持，这些功能可更好地与 T-SQL 功能和行为保持一致。尽管如此，当前实施中存在一些不受支持的功能和差异。下面，您可以找到有关 Babelfish 和 T-SQL 之间功能差异的信息，以及一些解决方法或使用说明。

从 Babelfish 版本 1.2.0 开始，以下功能当前具有有限实施：

- SQL Server 目录（系统视图）– 目录 `sys.sysconfigures`、`sys.syscurconfigs` 和 `sys.configurations` 仅支持单个只读配置。当前不支持 `sp_configure`。有关 Babelfish 实现的其他 SQL Server 视图的更多信息，请参阅获取 Babelfish 系统目录中的信息。
- 授予权限 – 支持 GRANT... TO PUBLIC，但是当前不支持 GRANT..TO PUBLIC WITH GRANT OPTION。
- SQL Server 所有权链 和权限机制限制 – 在 Babelfish 中，SQL Server 所有权链适用于视图，但不适用于存储过程。这意味着必须授予过程对与调用过程相同的所有者拥有的其他对象的显式访问权限。在 SQL Server 中，授予调用者对该过程的 EXECUTE 权限就足以调用同一所有者拥有的其他对象。在 Babelfish 中，还必须向调用者授予对该过程访问的对象的权限。
- 解析非限定（没有架构名称）对象引用 – 当 SQL 对象（过程、视图、函数或触发器）引用一个对象而没有使用架构名称对其进行限定时，SQL Server 会使用引用发生的 SQL 对象的架构名称来解析对象的架构名称。目前，Babelfish 通过使用执行该过程的数据库用户的默认模式，以不同方式来解析该名称。
- 默认模式更改、会话和连接 – 如果用户使用 `ALTER USER...WITH DEFAULT SCHEMA` 更改默认模式，更改将立即在该会话中生效。但是，对于属于同一用户的其他当前连接的会话，时间有所不同，如下所示：
  - 对于 SQL Server：此更改将立即在所有其他连接中对此用户生效。
  - 对于 Babelfish：此更改将仅在新连接中对此用户生效。
- ROWVERSION 和 TIMESTAMP 数据类型的实现和转义孵化设置 – Babelfish 现在支持 ROWVERSION 和 TIMESTAMP 数据类型。要在 Babelfish 中使用 ROWVERSION 或 TIMESTAMP，必须将转义孵化的设置 `babelfishpg_tsql.escape_hatch_rowversion` 从默认值 (strict) 更改为 `ignore`。ROWVERSION 和 TIMESTAMP 数据类型的 Babelfish 实现在语义上基本上与 SQL Server 相同，但存在以下例外：
  - 内置的 @@DBTS 函数的行为与 SQL Server 类似，但有小的区别。由于底层 PostgreSQL 数据库引擎及其多版本并发控制 (MVCC) 实现，Babelfish 会生成新的时间戳，而不是返回上次使用的 `SELECT @@DBTS` 值。
  - 在 SQL Server 中，每个插入或更新的行都会获得一个唯一的 ROWVERSION/TIMESTAMP 值。在 Babelfish 中，由同一语句更新的每个插入的行都会被分配相同的 ROWVERSION/TIMESTAMP 值。

    例如，当 UPDATE 语句或 INSERT-SELECT 语句影响多行时，在 SQL Server 中，受影响的行在其 ROWVERSION/TIMESTAMP 列中都有不同的值。在 Babelfish (PostgreSQL) 中，行具有相同的值。

- 在 SQL Server 中，当您使用 SELECT-INTO 创建新表时，可以将显式值（例如 NULL）转换为待创建的 ROWVERSION/TIMESTAMP 列。当您在 Babelfish 中做同样的事情时，Babelfish 会为新表中的每一行分配一个实际的 ROWVERSION/TIMESTAMP 值。

  ROWVERSION/TIMESTAMP 数据类型的这些细微差异应该不会对在 Babelfish 上运行的应用程序产生负面影响。

- 模式创建、所有权和权限 – 在非 DBO 用户（使用 `CREATE SCHEMA` *`schema name`* `AUTHORIZATION` *`user name`*）拥有的模式中创建和访问对象的权限与适用于 SQL Server 和 Babelfish 的非 DBO 用户不同，如下表所示：

拥有模式的数据库用户（非 DBO）可以执行以下操作：	SQL Server	Babelfish
在没有 DBO 额外授权的情况下在模式中创建对象？	否	是
访问由 DBO 在模式中创建的对象，而无需额外授权？	是	否

- CREATE OR ALTER VIEW / ALTER VIEW 语法：Babelfish 中对这些语法的支持具有以下限制：
  - 这些语句不能用于附加了 INSTEAD-OF 触发器的视图。
  - 这些语句不能用于那些基于此视图构建了其它视图的视图。

提高 Babelfish 查询性能

可以使用查询提示和 PostgreSQL 优化器在 Babelfish 中实现更快的查询处理。

主题

- 使用解释计划提高 Babelfish 查询性能
- 使用 T-SQL 查询提示提高 Babelfish 查询性能

也可以使用 `sp_babelfish_volatility` 过程提高查询性能。有关更多信息，请参阅 sp_babelfish_volatility。

还可以使用子查询转换和子查询缓存来提高查询性能。有关更多信息，请参阅 优化 Aurora PostgreSQL 中的关联子查询。

使用解释计划提高 Babelfish 查询性能

从版本 2.1.0 开始，Babelfish 包括两个函数，它们透明地使用 PostgreSQL 优化程序为 TDS 端口上的 T-SQL 查询生成估计和实际的查询计划。这些函数类似于对 SQL Server 数据库使用 SET STATISTICS PROFILE 或 SET SHOWPLAN_ALL 来识别和改进运行缓慢的查询。

> **Note**
>
> 目前不支持从函数、控制流程和游标获取查询计划。

在该表中，您可以找到跨 SQL Server、Babelfish 和 PostgreSQL 的查询计划解释函数的比较。

SQL Server	Babelfish	PostgreSQL
SHOWPLAN_ALL	BABELFISH_SHOWPLAN_ALL	EXPLAIN
STATISTICS PROFILE	BABELFISH_STATISTICS PROFILE	EXPLAIN ANALYZE
使用 SQL Server 优化程序	使用 PostgreSQL 优化程序	使用 PostgreSQL 优化程序
SQL Server 输入和输出格式	SQL Server 输入和 PostgreSQL 输出格式	PostgreSQL 输入和输出格式
为会话设置	为会话设置	应用于特定的语句
支持以下各项： • SELECT • INSERT • UPDATE • 删除 • CURSOR	支持以下各项： • SELECT • INSERT • UPDATE • 删除 • CREATE	支持以下各项： • SELECT • INSERT • UPDATE • 删除 • CURSOR

SQL Server	Babelfish	PostgreSQL
• CREATE • EXECUTE • EXEC 和函数，包括控制流程（CASE、WHILE-BREAK-CONTINUE、WAITFOR、BEGIN-END、IF-ELSE 等等）	• EXECUTE • EXEC • RAISEERROR • THROW • PRINT • USE	• CREATE • EXECUTE

按如下方式使用 Babelfish 函数：

- SET BABELFISH_SHOWPLAN_ALL [ON|OFF] – 设置为 ON 可生成估计的查询执行计划。此函数实现 PostgreSQL `EXPLAIN` 命令的行为。使用此命令获取给定查询的解释计划。
- SET BABELFISH_STATISTICS PROFILE [ON|OFF] – 对于实际查询执行计划，设置为 ON。此函数实现 PostgreSQL 的 `EXPLAIN ANALYZE` 命令的行为。

有关 PostgreSQL `EXPLAIN` 和 `EXPLAIN ANALYZE` 的更多信息，请参阅 PostgreSQL 文档中的 EXPLAIN。

> **Note**
>
> 从版本 2.2.0 开始，您可以将 `escape_hatch_showplan_all` 参数设置为 ignore（忽略），以避免在 `SHOWPLAN_ALL` 和 `STATISTICS PROFILE` SET 命令的 SQL Server 语法中使用 BABELFISH_ 前缀。

例如，以下命令序列开启查询规划，然后返回 SELECT 语句的估计查询执行计划而不运行查询。此示例通过 `northwind` 命令行工具查询 TDS 端口，以使用 SQL Server 示例 `sqlcmd` 数据库：

```
1> SET BABELFISH_SHOWPLAN_ALL ON
2> GO
1> SELECT t.territoryid, e.employeeid FROM
2> dbo.employeeterritories e, dbo.territories t
3> WHERE e.territoryid=e.territoryid ORDER BY t.territoryid;
4> GO
```

```
QUERY PLAN


------------------------------------------------------------------------------


Query Text: SELECT t.territoryid, e.employeeid FROM
dbo.employeeterritories e, dbo.territories t
WHERE e.territoryid=e.territoryid ORDER BY t.territoryid
Sort  (cost=6231.74..6399.22 rows=66992 width=10)
  Sort Key: t.territoryid NULLS FIRST
  ->  Nested Loop  (cost=0.00..861.76 rows=66992 width=10)
        ->  Seq Scan on employeeterritories e  (cost=0.00..22.70 rows=1264 width=4)
              Filter: ((territoryid)::"varchar" IS NOT NULL)
        ->  Materialize  (cost=0.00..1.79 rows=53 width=6)
              ->  Seq Scan on territories t  (cost=0.00..1.53 rows=53 width=6)
```

完成查询的检查和调整后，可以停用此函数，如下所示：

```
1> SET BABELFISH_SHOWPLAN_ALL OFF
```

将 BABELFIS_STATIONS PROFISATIONS 设置为 ON 时，每个执行的查询都会返回其常规结果集，然后是显示实际查询执行计划的附加结果集。Babelfish 生成的查询计划会在调用 SELECT 语句时提供最快的结果集。

```
1> SET BABELFISH_STATISTICS PROFILE ON
1>
2> GO
1> SELECT e.employeeid, t.territoryid FROM
2> dbo.employeeterritories e, dbo.territories t
3> WHERE t.territoryid=e.territoryid ORDER BY t.territoryid;
4> GO
```

返回结果集和查询计划（此示例仅显示查询计划）。

```
QUERY PLAN


------------------------------------------------------------------------------
Query Text: SELECT e.employeeid, t.territoryid FROM
dbo.employeeterritories e, dbo.territories t
```

```
WHERE t.territoryid=e.territoryid ORDER BY t.territoryid
Sort  (cost=42.44..43.28 rows=337 width=10)
  Sort Key: t.territoryid NULLS FIRST


  ->  Hash Join  (cost=2.19..28.29 rows=337 width=10)
       Hash Cond: ((e.territoryid)::"varchar" = (t.territoryid)::"varchar")
        ->  Seq Scan on employeeterritories e  (cost=0.00..22.70 rows=1270 width=36)
        ->  Hash  (cost=1.53..1.53 rows=53 width=6)
             ->  Seq Scan on territories t  (cost=0.00..1.53 rows=53 width=6)
```

要了解有关如何分析查询以及 PostgreSQL 优化程序返回的结果的更多信息，请参阅 explain.depesz.com。有关 PostgreSQL EXPLAIN 和 EXPLAIN ANALYZE 的更多信息，请参阅 PostgreSQL 文档中的 EXPLAIN。

控制 Babelfish 解释选项的参数

您可以使用下表中显示的参数来控制查询计划显示的信息类型。

参数	描述
babelfishpg_tsql.explain_buffers	一个布尔值，用于打开（和关闭）优化程序的缓冲区使用信息。（原定设置值：关闭）（允许的值：关闭、打开）
babelfishpg_tsql.explain_costs	一个布尔值，用于打开（和关闭）优化程序的估计启动和总成本信息。（原定设置值：打开）（允许的值：关闭、打开）
babelfishpg_tsql.explain_format	指定 EXPLAIN 计划的输出格式。（原定设置值：text）（允许的值：text、xml、json、yaml）
babelfishpg_tsql.explain_settings	一个布尔值，用于打开（或关闭）在 EXPLAIN 计划输出中包含有关配置参数的信息。（原定设置值：关闭）（允许的值：关闭、打开）
babelfishpg_tsql.explain_summary	一个布尔值，用于打开（或关闭）摘要信息，例如查询计划后的总时间。（原定设置值：打开）（允许的值：关闭、打开）

参数	描述
babelfishpg_tsql.explain_timing	一个布尔值，用于在输出中打开（或关闭）实际启动时间和每个节点花费的时间。（原定设置值：打开）（允许的值：关闭、打开）
babelfishpg_tsql.explain_verbose	一个布尔值，用于打开（或关闭）解释计划的最详细版本。（原定设置值：关闭）（允许的值：关闭、打开）
babelfishpg_tsql.explain_wal	一个布尔值，用于打开（或关闭）WAL 记录信息的生成，以作为解释计划的一部分。（原定设置值：关闭）（允许的值：关闭、打开）

您可以使用 PostgreSQL 客户端或 SQL Server 客户端来检查系统上任何 BabelFish 相关参数的值。运行以下命令来获取当前参数值：

```
1> execute sp_babelfish_configure '%explain%';
2> GO
```

在以下输出中，您可以看到此特定 Babelfish 数据库集群上的所有设置均为原定设置值。此示例中并未显示所有输出。

```


            name                    setting                    short_desc
---------------------------------- --------
 ------------------------------------------------------------
babelfishpg_tsql.explain_buffers   off      Include information on buffer usage
babelfishpg_tsql.explain_costs     on       Include information on estimated startup
 and total cost
babelfishpg_tsql.explain_format    text     Specify the output format, which can be
 TEXT, XML, JSON, or YAML
babelfishpg_tsql.explain_settings  off      Include information on configuration
 parameters
babelfishpg_tsql.explain_summary   on       Include summary information (e.g.,totaled
 timing information) after the query plan
babelfishpg_tsql.explain_timing    on       Include actual startup time and time spent
 in each node in the output
babelfishpg_tsql.explain_verbose   off      Display additional information regarding
 the plan
```

```
babelfishpg_tsql.explain_wal       off      Include information on WAL record
 generation

(8 rows affected)
```

可以使用 sp_babelfish_configure 更改这些参数的设置，如下例所示。

```
1> execute sp_babelfish_configure 'explain_verbose', 'on';
2> GO
```

如果您要在集群范围级别使设置永久化，请包括关键字 server，如以下示例所示。

```
1> execute sp_babelfish_configure 'explain_verbose', 'on', 'server';
2> GO
```

使用 T-SQL 查询提示提高 Babelfish 查询性能

从版本 2.3.0 开始，Babelfish 支持通过 pg_hint_plan 使用查询提示。在 Aurora PostgreSQL 中，原定设置情况下安装了 pg_hint_plan。有关 PostgreSQL 扩展 pg_hint_plan 的更多信息，请参阅 https://github.com/ossc-db/pg_hint_plan。有关 Aurora PostgreSQL 支持的这一扩展的版本的详细信息，请参阅《Aurora PostgreSQL 发布说明》中的 Amazon Aurora PostgreSQL 的扩展版本。

查询优化程序经过精心设计，可以找到 SQL 语句的最佳执行计划。选择计划时，查询优化程序会同时考虑引擎的成本模型以及列和表统计数据。但是，建议的计划可能无法满足数据集的需求。因此，查询提示解决了性能问题以改进执行计划。query hint 是 SQL 标准中添加的语法，用于指示数据库引擎如何执行查询。例如，提示可能会指示引擎进行顺序扫描并覆盖查询优化程序已选择的任何计划。

在 Babelfish 中开启 T-SQL 查询提示

目前，Babelfish 原定设置情况下忽略所有 T-SQL 提示。要应用 T-SQL 提示，请在将 enable_pg_hint 值设为 ON 的情况下运行命令 sp_babelfish_configure。

```
EXECUTE sp_babelfish_configure 'enable_pg_hint', 'on' [, 'server']
```

通过包含 server 关键字，可以在集群级别上使设置永久化。要仅为当前会话配置此设置，请不要使用 server。

开启 enable_pg_hint 后，Babelfish 会应用以下 T-SQL 提示。

- INDEX 提示
- JOIN 提示

- FORCE ORDER 提示
- MAXDOP 提示

例如，以下命令序列开启 pg_hint_plan。

```
1> CREATE TABLE t1 (a1 INT PRIMARY KEY, b1 INT);
2> CREATE TABLE t2 (a2 INT PRIMARY KEY, b2 INT);
3> GO
1> EXECUTE sp_babelfish_configure 'enable_pg_hint', 'on';
2> GO
1> SET BABELFISH_SHOWPLAN_ALL ON;
2> GO
1> SELECT * FROM t1 JOIN t2 ON t1.a1 = t2.a2; --NO HINTS (HASH JOIN)
2> GO
```

不对 SELECT 语句应用任何提示。返回没有提示的查询计划。

```
QUERY PLAN


------------------------------------------------------------------------------
Query Text: SELECT * FROM t1 JOIN t2 ON t1.a1 = t2.a2
Hash Join (cost=60.85..99.39 rows=2260 width=16)
 Hash Cond: (t1.a1 = t2.a2)
 -> Seq Scan on t1 (cost=0.00..32.60 rows=2260 width=8)
 -> Hash (cost=32.60..32.60 rows=2260 width=8)
 -> Seq Scan on t2 (cost=0.00..32.60 rows=2260 width=8)
```

```
1> SELECT * FROM t1 INNER MERGE JOIN t2 ON t1.a1 = t2.a2;
2> GO
```

查询提示应用于 SELECT 语句。以下输出显示返回了带有合并联接的查询计划。

```
QUERY PLAN


------------------------------------------------------------------------------
```

```
Query Text: SELECT/*+ MergeJoin(t1 t2) Leading(t1 t2)*/ * FROM t1 INNER JOIN t2 ON
 t1.a1 = t2.a2
Merge Join (cost=0.31..190.01 rows=2260 width=16)
 Merge Cond: (t1.a1 = t2.a2)
 -> Index Scan using t1_pkey on t1 (cost=0.15..78.06 rows=2260 width=8)
 -> Index Scan using t2_pkey on t2 (cost=0.15..78.06 rows=2260 width=8)
```

```
1> SET BABELFISH_SHOWPLAN_ALL OFF;
2> GO
```

限制

使用查询提示时，请考虑以下限制：

- 如果查询计划在开启 `enable_pg_hint` 之前已缓存，则不会在同一个会话中应用提示。它将在新会话中应用。
- 如果显式给出了模式名称，则无法应用提示。您可以使用表别名作为解决方法。
- 查询提示不能应用于视图和子查询。
- 提示不适用于带有 JOIN 的 UPDATE/DELETE 语句。
- 将忽略不存在的索引或表的索引提示。
- FORCE ORDER 提示不适用于 HASH JOIN 和非 ANSI JOIN。

将 Aurora PostgreSQL 扩展与 Babelfish 搭配使用

Aurora PostgreSQL 提供了与其他 AWS 服务搭配使用的扩展。这些是支持各种使用案例的可选扩展，例如将 Amazon S3 与数据库集群搭配使用来导入或导出数据。

- 要将数据从 Amazon S3 桶导入 Babelfish 数据库集群，可以设置 `aws_s3` Aurora PostgreSQL 扩展。此扩展还允许将数据从 Aurora PostgreSQL 数据库集群导出到 Amazon S3 存储桶。
- AWS Lambda 是一项计算服务，可使您无需调配或管理服务器即可运行代码。您可以使用 Lambda 函数完成许多任务，例如处理来自数据库实例的事件通知。若要了解 Lambda 的更多信息，请参阅《AWS Lambda 开发人员指南》中的什么是 AWS Lambda？ 要从 Babelfish 数据库集群调用 Lambda 函数，您可以设置 `aws_lambda` Aurora PostgreSQL 扩展。

要为 Babelfish 集群设置这些扩展，首先需要向内部 Babelfish 用户授予加载扩展的权限。授予权限后，您就可以加载 Aurora PostgreSQL 扩展程序。

在 Babelfish 数据库集群中启用 Aurora PostgreSQL 扩展

必须向 Babelfish 数据库集群授予所需权限，才能加载 aws_s3 或者 aws_lambda 扩展。

以下过程使用 psql PostgreSQL 命令行工具连接到该数据库集群。有关更多信息，请参阅 使用 psql 连接到数据库集群。您还可以使用 pgAdmin。有关更多信息，请参阅 使用 pgAdmin 连接到数据库集群。

此过程将先后加载 aws_s3 和 aws_lambda。如果只想使用其中一个扩展，则无需同时加载两个扩展。各种情况都需要 aws_commons 扩展，因此会默认加载该扩展，如输出中所示。

使用 Aurora PostgreSQL 扩展的权限设置 Babelfish 数据库集群

1. 连接到 Babelfish 数据库集群。使用创建 Babelfish 数据库集群时指定的“主”用户名 (-U)。默认名称 (postgres) 如示例所示。

   对于 Linux、macOS 或 Unix：

   ```
   psql -h your-Babelfish.cluster.444455556666-us-east-1.rds.amazonaws.com \
   -U postgres \
   -d babelfish_db \
   -p 5432
   ```

   对于 Windows：

   ```
   psql -h your-Babelfish.cluster.444455556666-us-east-1.rds.amazonaws.com ^
   -U postgres ^
   -d babelfish_db ^
   -p 5432
   ```

   该命令将提示输入用户名 (-U) 的密码。

   ```
   Password:
   ```

   输入数据库集群用户名 (-U) 的密码。成功连接后，便可看到类似以下内容的输出。

   ```
   psql (13.4)
   SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256,
    compression: off)
   Type "help" for help.
   ```

```
postgres=>
```

2. 向内部 Babelfish 用户授予创建和加载扩展的权限。

```
babelfish_db=> GRANT rds_superuser TO master_dbo;
GRANT ROLE
```

3. 创建并加载 aws_s3 扩展。aws_commons 扩展是必要扩展，会在 aws_s3 安装时自动安装。

```
babelfish_db=> create extension aws_s3 cascade;
NOTICE:  installing required extension "aws_commons"
CREATE EXTENSION
```

4. 创建并加载 aws_lambda 扩展。

```
babelfish_db=> create extension aws_lambda cascade;
CREATE EXTENSION
babelfish_db=>
```

将 Babelfish 与 Amazon S3 搭配使用

如果尚无可与 Babelfish 数据库集群搭配使用的 Amazon S3 存储桶，可以创建一个。您可以向要使用的任何 Amazon S3 存储桶提供访问权限。

请完成以下一次性步骤，再尝试使用 Amazon S3 存储桶导入或导出数据。

为 Babelfish 数据库实例设置对 Amazon S3 存储桶的访问权限

1. 根据需要为 Babelfish 实例创建 Amazon S3 存储桶。为此，请参照《Amazon Simple Storage Service User Guide》的 Create a bucket 中的说明。
2. 将文件上传到 Amazon S3 存储桶。为此，请参照《Amazon Simple Storage Service User Guide》的 Add an object to a bucket 中的步骤。
3. 根据需要设置权限：

   - 要从 Amazon S3 导入数据，Babelfish 数据库集群需要获得访问桶的权限。建议使用 AWS Identity and Access Management (IAM) 角色，并将 IAM 策略附加到集群的该角色。为此，请按照使用 IAM 角色访问 Amazon S3 存储桶中的步骤进行操作。

- 要从 Babelfish 数据库集群导出数据，必须向集群授予对 Amazon S3 桶的访问权限。与导入一样，建议使用 IAM 角色和策略。为此，请按照设置 Amazon S3 存储桶的访问权限中的步骤进行操作。

现在，您可以将带有 aws_s3 扩展的 Amazon S3 与 Babelfish 数据库集群搭配使用。

将数据从 Amazon S3 导入 Babelfish 以及将 Babelfish 数据导出到 Amazon S3

1. 将 aws_s3 扩展与 Babelfish 数据库集群搭配使用。

   这样做时，请务必参考 PostgreSQL 上下文中存在的表格。也就是说，如果想导入名为 [database].[schema].[tableA] 的 Babelfish 表，则在 aws_s3 函数中将该表称为 database_schema_tableA：

   - 有关使用 aws_s3 函数导入数据的示例，请参阅 将数据从 Amazon S3 导入到 Aurora PostgreSQL 数据库集群。
   - 有关使用 aws_s3 函数导出数据的示例，请参见 使用 aws_s3.query_export_to_s3 函数导出查询数据。
2. 在使用 aws_s3 扩展和 Amazon S3 时，请务必使用 PostgreSQL 命名参考 Babelfish 表，如下表所示。

Babelfish 表	Aurora PostgreSQL 表
database.*schema*.*table*	*database*_*schema*_*table*

若要了解有关将 Amazon S3 与 Aurora PostgreSQL 搭配使用的更多信息，请参阅 将 Amazon S3 中的数据导入到 Aurora PostgreSQL 数据库集群 和 将数据从 Aurora PostgreSQL 数据库集群导出到 Amazon S3。

将 Babelfish 和 AWS Lambda 搭配使用

在 aws_lambda 扩展已加载到 Babelfish 数据库集群中之后，但在 Lambda 函数可以调用之前，您可以按照此过程授予 Lambda 对数据库集群的访问权限。

为 Babelfish 数据库集群设置访问权限以使用 Lambda

此过程使用 AWS CLI 创建 IAM 策略和角色，并将其与 Babelfish 数据库集群关联。

1. 创建允许从 Babelfish 数据库集群访问 Lambda 的 IAM 策略。

```
aws iam create-policy  --policy-name rds-lambda-policy --policy-document '{
    "Version": "2012-10-17",
    "Statement": [
        {
        "Sid": "AllowAccessToExampleFunction",
        "Effect": "Allow",
        "Action": "lambda:InvokeFunction",
        "Resource": "arn:aws:lambda:aws-region:444455556666:function:my-function"
        }
    ]
}'
```

2. 创建该策略可在运行时承担的 IAM 角色。

```
aws iam create-role  --role-name rds-lambda-role --assume-role-policy-document '{
    "Version": "2012-10-17",
    "Statement": [
        {
        "Effect": "Allow",
        "Principal": {
            "Service": "rds.amazonaws.com"
        },
        "Action": "sts:AssumeRole"
        }
    ]
}'
```

3. 将 策略附加到该角色。

```
aws iam attach-role-policy \
    --policy-arn arn:aws:iam::444455556666:policy/rds-lambda-policy \
    --role-name rds-lambda-role --region aws-region
```

4. 向 Babelfish 数据库集群附加角色

```
aws rds add-role-to-db-cluster \
       --db-cluster-identifier my-cluster-name \
       --feature-name Lambda \
       --role-arn  arn:aws:iam::444455556666:role/rds-lambda-role   \
       --region aws-region
```

完成这些任务之后，您可以调用 Lambda 函数。有关使用 AWS Lambda 为 Aurora PostgreSQL 数据库集群设置 AWS Lambda 的更多信息和示例，请参阅 步骤 2：为 Aurora PostgreSQL 数据库集群和 AWS Lambda 配置 IAM。

从 Babelfish 数据库集群调用 Lambda 函数

AWS Lambda 支持使用 Java、Node.js、Python、Ruby 和其他语言编写的函数。如果调用该函数时返回文本，则可以从 Babelfish 数据库集群中调用它。以下示例是返回问候语的占位符 Python 函数。

```
lambda_function.py
import json
def lambda_handler(event, context):
    #TODO implement
    return {
        'statusCode': 200,
        'body': json.dumps('Hello from Lambda!')
```

目前，Babelfish 不支持 JSON。如果函数返回 JSON，则可使用包装器来处理 JSON。例如，假设之前显示的 `lambda_function.py` 作为 `my-function` 存储在 Lambda 中。

1. 使用 `psql` 客户端（或 pgAdmin 客户端）连接到 Babelfish 数据库集群。有关更多信息，请参阅 使用 psql 连接到数据库集群。
2. 创建包装器。此示例使用 PostgreSQL 的 SQL 程序语言：PL/pgSQL。要了解详情，请参阅 PL/pgSQL–SQL 程序语言。

```
create or replace function master_dbo.lambda_wrapper()
returns text
language plpgsql
as
$$
declare
   r_status_code integer;
   r_payload text;
begin
   SELECT payload INTO r_payload
     FROM aws_lambda.invoke(  aws_commons.create_lambda_function_arn('my-function',
 'us-east-1')
                            ,'{"body": "Hello from Postgres!"}'::json );
   return r_payload ;
end;
$$;
```

该函数现在可以从 Babelfish TDS 端口 (1433) 或从 PostgreSQL 端口 (5433) 运行。

a. 若要从 PostgreSQL 端口调用此函数：

```
SELECT * from aws_lambda.invoke(aws_commons.create_lambda_function_arn('my-
function', 'us-east-1'), '{"body": "Hello from Postgres!"}'::json );
```

输出类似于以下内容：

```
status_code |                         payload                          |
 executed_version | log_result
-------------+----------------------------------------------------------
+------------------+------------
         200 | {"statusCode": 200, "body": "\"Hello from Lambda!\""} | $LATEST
         |
(1 row)
```

b. 若要从 TDS 端口调用此函数，请使用 SQL Server `sqlcmd` 命令行客户端连接到此端口。有关更多信息，请参阅 使用 SQL Server 客户端连接到数据库集群。连接后，运行以下命令：

```
1> select lambda_wrapper();
2> go
```

该命令返回的输出类似于下方内容：

```
{"statusCode": 200, "body": "\"Hello from Lambda!\""}
```

若要了解有关将 Lambda 与 Aurora PostgreSQL 搭配使用的更多信息，请参阅 从 Aurora PostgreSQL 数据库集群中调用 AWS Lambda 函数。有关使用 Lambda 函数的更多信息，请参阅《AWS Lambda 开发人员指南》中的 Lambda 入门。

在 Babelfish 中使用 pg_stat_statements

从 3.3.0 开始，适用于 Aurora PostgreSQL 的 Babelfish 支持 `pg_stat_statements` 扩展。要了解更多信息，请参阅 pg_stat_statements。

有关 Aurora PostgreSQL 支持的这一扩展的版本详细信息，请参阅扩展版本。

创建 pg_stat_statements 扩展

要开启 pg_stat_statements，必须开启查询标识符计算。如果在参数组中将 compute_query_id 设置为 on 或 auto，则会自动完成此操作。compute_query_id 参数的默认值为 auto。您还需要创建此扩展，才能开启此功能。使用以下命令从 T-SQL 端点安装扩展：

```
1>EXEC sp_execute_postgresql 'CREATE EXTENSION pg_stat_statements WITH SCHEMA sys';
```

您可以使用以下查询访问查询统计信息：

```
postgres=>select * from pg_stat_statements;
```

> **Note**
>
> 在安装过程中，如果您不提供扩展的模式名称，则默认情况下，将在公共模式中创建名称。要进行访问，必须对模式限定符使用方括号，如下所示：

```
postgres=>select * from [public].pg_stat_statements;
```

您还可以从 PSQL 端点创建扩展。

授权扩展

默认情况下，无需任何授权即可查看在 T-SQL 数据库中执行的查询的统计信息。

要访问其他人创建的查询统计信息，您需要拥有 pg_read_all_stats PostgreSQL 角色。按照下面提到的步骤构建 GRANT pg_read_all_stats 命令。

1. 在 T-SQL 中，使用以下返回内部 PG 角色名称的查询。

   ```
   SELECT rolname FROM pg_roles WHERE oid = USER_ID();
   ```

2. 使用 rds_superuser 权限连接到适用于 Aurora PostgreSQL 的 Babelfish 数据库，然后使用以下命令：

```
GRANT pg_read_all_stats TO <rolname_from_above_query>
```

示例

从 T-SQL 端点：

```
1>SELECT rolname FROM pg_roles WHERE oid = USER_ID();
2>go
```

```
rolname
-------
master_dbo
(1 rows affected)
```

从 PSQL 端点：

```
babelfish_db=# grant pg_read_all_stats to master_dbo;
```

```
GRANT ROLE
```

您可以使用 pg_stat_statements 视图访问查询统计信息：

```
1>create table t1(cola int);
2>go
1>insert into t1 values (1),(2),(3);
2>go
```

```
(3 rows affected)
```

```
1>select userid, dbid, queryid, query from pg_stat_statements;
2>go
```

```
userid dbid queryid             query
------ ---- -------             -----
```

```
37503 34582 6487973085327558478 select * from t1
37503 34582 6284378402749466286 SET QUOTED_IDENTIFIER OFF
37503 34582 2864302298511657420 insert into t1 values ($1),($2),($3)
10    34582 NULL                <insufficient privilege>
37503 34582 5615368793313871642 SET TEXTSIZE 4096
37503 34582 639400815330803392  create table t1(cola int)
(6 rows affected)
```

重置查询统计信息

您可以使用 pg_stat_statements_reset() 重置 pg_stat_statements 到目前为止收集的统计信息。要了解更多信息，请参阅 pg_stat_statements。目前仅通过 PSQL 端点支持它。使用 rds_superuser 权限连接到适用于 Aurora PostgreSQL 的 Babelfish，然后使用以下命令：

```
SELECT pg_stat_statements_reset();
```

限制

- 目前，通过 T-SQL 端点不支持 pg_stat_statements()。pg_stat_statements 视图是推荐的收集统计信息的方法。
- 有些查询可能会由 Aurora PostgreSQL 引擎实现的 T-SQL 解析器重新写入，pg_stat_statements 视图将显示重新写入的查询，而不是原始查询。

  示例

  ```
  select next value for [dbo].[newCounter];
  ```

  在 pg_stat_statements 视图中，上述查询被重新写入为以下内容。

  ```
  select nextval($1);
  ```

- 根据语句的执行流程，pg_stat_statements 可能无法跟踪某些查询，因此在视图中也看不见这些查询。这包括以下语句：use dbname、goto、print、raise error、set、throw、declare cursor。
- 对于 CREATE LOGIN 和 ALTER LOGIN 语句，将不显示查询和查询 ID。它将显示权限不足。
- pg_stat_statements 视图始终包含以下两个条目，因为它们是由 sqlcmd 客户端在内部执行。

- SET QUOTED_IDENTIFIER OFF
- SET TEXTSIZE 4096

在 Babelfish 中使用 pgvector

pgvector 是一个开源扩展，它允许您直接在 Postgres 数据库中搜索类似的数据。从版本 15.6 和 16.2 开始，Babelfish 支持此扩展。有关更多信息，请参阅 pgvector 开源文档。

先决条件

要启用 pgvector 功能，请使用以下方法之一在 sys 架构中安装该扩展：

- 在 sqlcmd 客户端运行以下命令：

```
exec sys.sp_execute_postgresql 'CREATE EXTENSION vector WITH SCHEMA sys';
```

- 连接到 babelfish_db 并在 psql 客户端运行以下命令：

```
CREATE EXTENSION vector WITH SCHEMA sys;
```

> **Note**
>
> 安装 pgvector 扩展后，矢量数据类型将仅在您建立的新数据库连接中可用。现有连接无法识别新的数据类型。

支持的功能

Babelfish 扩展了 T-SQL 功能以支持以下功能：

- 存储

  Babelfish 现在支持与矢量数据类型兼容的语法，从而增强了其 T-SQL 兼容性。要了解有关使用 pgvector 存储数据的更多信息，请参阅存储。

- 查询

  Babelfish 扩展了 T-SQL 表达式支持，以包括矢量相似度运算符。但是，对于所有其他查询，仍然需要标准 T-SQL 语法。

> **Note**
>
> T-SQL 不支持数组类型，而且数据库驱动程序没有任何接口来处理它们。作为一种解决方法，Babelfish 使用文本字符串（varchar/nvarchar）来存储矢量数据。例如，当您请求向量值 [1,2,3] 时，Babelfish 将返回字符串“[1,2,3]”作为响应。您可以根据需要在应用程序级别解析和拆分此字符串。

要了解有关使用 pgvector 查询数据的更多信息，请参阅查询。

- 索引

  T-SQL `Create Index` 现在支持 `USING INDEX_METHOD` 语法。现在，您可以定义在创建索引时用于特定列的相似度搜索运算符。

  该语法还进行了扩展，以支持对所需列进行向量相似度运算（查看 column_name_list_with_order_for_vector 语法）。

```
CREATE [UNIQUE] [clustered] [COLUMNSTORE] INDEX <index_name> ON <table_name> [USING
 vector_index_method] (<column_name_list_with_order_for_vector>)
Where column_name_list_with_order_for_vector is:
    <column_name> [ASC | DESC] [VECTOR_COSINE_OPS | VECTOR_IP_OPS | VECTOR_L2_OPS]
 (COMMA simple_column_name [ASC | DESC] [VECTOR_COSINE_OPS | VECTOR_IP_OPS |
 VECTOR_L2_OPS])
```

  要了解有关使用 pgvector 编制数据索引的更多信息，请参阅索引。

- 性能
  - 使用 `SET BABELFISH_STATISTICS PROFILE ON` 从 T-SQL 端点调试查询计划。
  - 使用 T-SQL 中支持的 `set_config` 函数增加 `max_parallel_workers_get_gather`。
  - 使用 `IVFFlat` 执行近似搜索。有关更多信息，请参阅 IVFFlat。

  要使用 pgvector 提高性能，请参阅性能。

限制

- Babelfish 不支持通过全文搜索进行混合搜索。有关更多信息，请参阅混合搜索。

- Babelfish 目前不支持重新编制索引功能。但是，仍然可以使用 PostgreSQL 端点来重新编制索引。有关更多信息，请参阅清空。

将 Amazon Aurora 机器学习与 Babelfish 结合使用

您可以通过将 Babelfish for Aurora PostgreSQL 数据库集群与 Amazon Aurora 机器学习集成来扩展其功能。这种无缝集成使您可以访问一系列强大的服务，例如 Amazon Comprehend、Amazon SageMaker AI 或 Amazon Bedrock，这些服务都经过了定制，以满足不同的机器学习需求。

作为 Babelfish 用户，在与 Aurora 机器学习结合使用时，您可以利用现有的 T-SQL 语法和语义知识。按照 Aurora PostgreSQL AWS 文档中提供的说明操作。有关更多信息，请参阅 将 Amazon Aurora 机器学习与 Aurora PostgreSQL 结合使用。

先决条件

- 在尝试将 Babelfish for Aurora PostgreSQL 数据库集群设置为使用 Aurora 机器学习之前，请务必了解以下要求和先决条件。有关更多信息，请参阅 将 Aurora 机器学习与 Aurora PostgreSQL 结合使用的要求。
- 确保使用 Postgres 端点或 `sp_execute_postgresql` 存储过程安装 `aws_ml` 扩展。

```
exec sys.sp_execute_postgresql 'Create Extension aws_ml'
```

> **Note**
>
> Babelfish 目前不支持在 Babelfish 中执行 `sp_execute_postgresql` 级联操作。由于 `aws_ml` 依赖 `aws_commons`，因此您需要使用 Postgres 端点单独安装它。

```
create extension aws_common;
```

使用 **`aws_ml`** 函数处理 T-SQL 语法和语义

以下示例说明了如何将 T-SQL 语法和语义应用于 Amazon ML 服务：

Example : aws_bedrock.invoke_model – 使用 Amazon Bedrock 函数的简单查询

```
aws_bedrock.invoke_model(
   model_id      varchar,
```

```
   content_type  text,
   accept_type   text,
   model_input   text)
Returns Varchar(MAX)
```

以下示例说明如何使用 invoke_model 为 Bedrock 调用 Anthropic Claude 2 模型。

```
SELECT aws_bedrock.invoke_model (
    'anthropic.claude-v2', -- model_id
    'application/json', -- content_type
    'application/json', -- accept_type
    '{"prompt": "\n\nHuman:
    You are a helpful assistant that answers questions directly
    and only using the information provided in the context below.
    \nDescribe the answerin detail.\n\nContext: %s \n\nQuestion:
    %s \n\nAssistant:","max_tokens_to_sample":4096,"temperature"
    :0.5,"top_k":250,"top_p":0.5,"stop_sequences":[]}' -- model_input
);
```

Example : aws_comprehend.detect_sentiment – 使用 Amazon Comprehend 函数的简单查询

```
aws_comprehend.detect_sentiment(
   input_text varchar,
   language_code varchar,
   max_rows_per_batch int)
Returns table (sentiment varchar, confidence real)
```

以下示例说明了如何调用 Amazon Comprehend 服务。

```
select sentiment from aws_comprehend.detect_sentiment('This is great', 'en');
```

Example : aws_sagemaker.invoke_endpoint – 使用 Amazon SageMaker 函数的简单查询

```
aws_sagemaker.invoke_endpoint(
  endpoint_name varchar,
  max_rows_per_batch int,
```

```
  VARIADIC model_input "any") -- Babelfish inherits PG's variadic parameter type
Rerurns Varchar(MAX)
```

由于 model_input 标记为 VARIADIC 且类型为“any”，因此用户可以将任意长度和任何数据类型的列表传递给将充当模型输入的函数。以下示例说明了如何调用 Amazon SageMaker 服务。

```
SELECT CAST (aws_sagemaker.invoke_endpoint(
    'sagemaker_model_endpoint_name',
    NULL,
    arg1, arg2 -- model inputs are separate arguments )
AS INT) -- cast the output to INT
```

有关将 Amazon Aurora 机器学习与 Aurora PostgreSQL 结合使用的更多信息，请参阅将 Amazon Aurora 机器学习与 Aurora PostgreSQL 结合使用。

限制

- 虽然 Babelfish 不允许创建数组，但它仍然可以处理表示数组的数据。当您使用类似 `aws_bedrock.invoke_model_get_embeddings` 这样能够返回数组的函数时，结果将以包含数组元素的字符串的形式传递。

Babelfish 支持链接服务器

适用于 Aurora PostgreSQL 的 Babelfish 在版本 3.1.0 中使用 PostgreSQL `tds_fdw` 扩展支持链接服务器。要使用链接服务器，您必须安装 `tds_fdw` 扩展。有关 `tds_fdw` 扩展的更多信息，请参阅使用 Amazon Aurora PostgreSQL 支持的外部数据包装器。

安装 **`tds_fdw`** 扩展

您可以使用以下方法安装 `tds_fdw` 扩展。

从 PostgreSQL 终端节点使用 CREATE EXTENSION

1. 在 PostgreSQL 端口中连接到 Babelfish 数据库上的 PostgreSQL 数据库实例。使用具有 rds_superuser 角色的账户。

   ```
   psql --host=your-DB-instance.aws-region.rds.amazonaws.com --port=5432 --
   username=test --dbname=babelfish_db --password
   ```

2. 安装 tds_fdw 扩展。这是一个一次性安装过程。数据库集群重新启动时，无需重新安装。

```
babelfish_db=> CREATE EXTENSION tds_fdw;
CREATE EXTENSION
```

从 TDS 终端节点调用 **sp_execute_postgresql** 存储过程

从 3.3.0 开始，Babelfish 支持通过调用 sp_execute_postgresql 过程来安装 tds_fdw 扩展。无需退出 T-SQL 端口即可从 T-SQL 终端节点执行 PostgreSQL 语句。有关更多信息，请参阅 使用适用于 Aurora PostgreSQL 的 Babelfish 过程

1. 在 T-SQL 端口中连接到 Babelfish 数据库上的 PostgreSQL 数据库实例。

```
sqlcmd -S your-DB-instance.aws-region.rds.amazonaws.com -U test -P password
```

2. 安装 tds_fdw 扩展。

```
1>EXEC sp_execute_postgresql N'CREATE EXTENSION tds_fdw';
2>go
```

支持的功能

Babelfish 支持将远程 RDS for SQL Server 端点或适用于 Aurora PostgreSQL 的 Babelfish 端点添加为链接服务器。您也可以将其他远程 SQL Server 实例添加为链接服务器。然后，使用 OPENQUERY() 从这些链接服务器检索数据。从 Babelfish 版本 3.2.0 开始，还支持由四部分组成的名称。

为了使用链接服务器，支持以下存储过程和目录视图。

存储过程

- sp_addlinkedserver – Babelfish 不支持 @provstr 参数。
- sp_addlinkedsrvlogin
  - 必须提供明确的远程用户名和密码才能连接到远程数据来源。您无法使用用户自己的凭证进行连接。Babelfish 仅支持 @useself = false。
  - Babelfish 不支持 @locallogin 参数，因为不支持配置特定于本地登录的远程服务器访问。
- sp_linkedservers

- sp_helplinkedsrvlogin
- sp_dropserver
- sp_droplinkedsrvlogin – Babelfish 不支持 `@locallogin` 参数，因为不支持配置特定于本地登录的远程服务器访问。
- sp_serveroption – Babelfish 支持以下服务器选项：
  - 查询超时（自 Babelfish 版本 3.2.0）
  - 连接超时（自 Babelfish 版本 3.3.0）
- sp_testlinkedserver（自 Babelfish 版本 3.3.0）
- sp_enum_oledb_providers（自 Babelfish 版本 3.3.0）

目录视图

- sys.servers
- sys.linked_logins

使用传输中的加密进行连接

从适用于 Aurora PostgreSQL 的 Babelfish 源服务器到目标远程服务器的连接使用传输中加密（TLS/SSL），具体取决于远程服务器数据库配置。如果远程服务器未配置为进行加密，则向远程数据库发出请求的 Babelfish 服务器会回退到未加密状态。

强制连接加密

- 如果目标链接服务器是 RDS for SQL Server 实例，则为目标 SQL Server 实例设置 `rds.force_ssl = on`。有关 RDS for SQL Server 的 SSL/TLS 配置的更多信息，请参阅将 SSL 与 Microsoft SQL Server 数据库实例结合使用
- 如果目标链接服务器是适用于 Aurora PostgreSQL 的 Babelfish 集群，请为目标服务器设置 `babelfishpg_tds.tds_ssl_encrypt = on` 和 `ssl = on`。有关 SSL/TLS 的更多信息，请参阅 Babelfish SSL 设置和客户端连接。

从 SQL Server 将 Babelfish 添加为链接服务器

适用于 Aurora PostgreSQL 的 Babelfish 可以从 SQL Server 添加为链接服务器。在 SQL Server 数据库上，可以使用适用于 ODBC 的 Microsoft OLE DB 提供程序将 Babelfish 添加为链接服务器：MSDASQL。

有两种方法可以使用 MSDASQL 提供程序从 SQL Server 将 Babelfish 配置为链接服务器：

- 提供 ODBC 连接字符串作为提供程序字符串。
- 添加链接服务器时提供 ODBC 数据来源的系统 DSN。

限制

- OPENQUERY() 仅适用于 SELECT，而不适用于 DML。
- 由四部分组成的对象名称仅用于读取，不适用于修改远程表。UPDATE 可以在 FROM 子句中引用远程表，而无需对其进行修改。
- 不支持针对 Babelfish 链接服务器执行存储过程。
- 如果存在依赖于 OPENQUERY( ) 的对象或通过由四部分组成的名称引用的对象，则 Babelfish 主要版本升级可能不起作用。在主要版本升级之前，必须确保删除任何引用 OPENQUERY( ) 或由四部分组成的名称的对象。
- 以下数据类型在远程 Babelfish 服务器上无法按预期运行：`nvarchar(max)`、`varchar(max)`、`varbinary(max)`、`binary(max)` 和 `time`。我们建议使用 CAST 函数将这些数据类型转换为支持的数据类型。

示例

在以下示例中，适用于 Aurora PostgreSQL 的 Babelfish 实例正在连接到云中的 RDS for SQL Server 实例。

```
EXEC master.dbo.sp_addlinkedserver @server=N'rds_sqlserver', @srvproduct=N'',
 @provider=N'SQLNCLI', @datasrc=N'myserver.CB2XKFSFFMY7.US-WEST-2.RDS.AMAZONAWS.COM';
EXEC master.dbo.sp_addlinkedsrvlogin
 @rmtsrvname=N'rds_sqlserver',@useself=N'False',@locallogin=NULL,@rmtuser=N'username',@rmtpassw
```

当链接服务器到位后，您可以使用 T-SQL OPENQUERY() 或标准的四部分命名来引用远程服务器上的表、视图或其他支持的对象：

```
SELECT * FROM OPENQUERY(rds_sqlserver, 'SELECT * FROM TestDB.dbo.t1');
SELECT * FROM rds_sqlserver.TestDB.dbo.t1;
```

要删除链接服务器和所有关联的登录信息，请执行以下操作：

```
EXEC master.dbo.sp_dropserver @server=N'rds_sqlserver', @droplogins=N'droplogins';
```

故障排除

您可以对源服务器和远程服务器使用同一个安全组，以允许它们相互通信。安全组应仅允许 TDS 端口（默认为 1433）上的入站流量，并且可以将安全组中的源 IP 设置为安全组 ID 本身。有关如何设置用于在具有相同安全组的实例之间进行连接的规则的更多信息，请参阅用于在具有相同安全组的实例之间进行连接的规则。

如果访问权限配置不正确，当尝试查询远程服务器时，会出现一条与以下示例类似的错误消息。

```
TDS client library error: DB #: 20009, DB Msg: Unable to connect: server is unavailable
 or does not exist (mssql2019.aws-region.rds.amazonaws.com), OS #: 110, OS Msg:
 Connection timed out, Level: 9
```

在 Babelfish 中使用全文搜索

从版本 4.0.0 开始，Babelfish 为全文搜索（FTS）提供有限支持。FTS 是关系数据库中的一项强大功能，可实现对文本密集型数据的高效搜索和索引。借助该功能，用户能够执行复杂的文本搜索并快速检索相关结果。FTS 对于处理大量文本数据的应用程序特别有价值，例如内容管理系统、电子商务平台和文档归档。

了解 Babelfish 全文搜索支持的功能

Babelfish 支持以下全文搜索功能：

- CONTAINS 子句：
  - 对 CONTAINS 子句的基本支持。

    ```
    CONTAINS (
         {
            column_name
         }
         , '<contains_search_condition>'
    )
    ```

> **Note**
>
> 目前，仅支持英语。

  - 全面处理和翻译 `simple_term` 搜索字符串。
- `FULLTEXT INDEX` 子句：
  - 仅支持 `CREATE FULLTEXT INDEX ON table_name(column_name [...n]) KEY INDEX index_name` 语句。
  - 支持完整 `DROP FULLTEXT INDEX` 语句。

> **Note**
>
> 为了重新对全文索引编制索引，您需要删除全文索引，并在同一列上创建一个新的全文索引。

- 搜索条件中的特殊字符：
  - Babelfish 可确保有效处理搜索字符串中出现的各个特殊字符。

> **Note**
>
> 虽然 Babelfish 现在可以识别搜索字符串中的特殊字符，但必须认识到，与使用 T-SQL 获得的结果相比，使用 Babelfish 获得的结果可能会有所不同。

- 列名中的表别名：
  - 借助表别名支持，用户可以为全文搜索创建更简洁易读的 SQL 查询。

Babelfish 全文搜索限制

- 目前，Babelfish 不支持 `CONTAINS` 子句的以下选项。
  - 不支持特殊字符和英语以外的语言。您将收到有关不支持的字符和语言的通用错误消息

```
Full-text search conditions with special characters or languages other than English
 are not currently supported in Babelfish
```

  - 多列，比如 `column_list`

  - PROPERTY 属性
  - `prefix_term`、`generation_term`、`generic_proximity_term`、`custom_proximity_term` 和 `weighted_term`
  - 不支持布尔运算符，如果使用该运算符，将收到以下错误消息：

    ```
    boolean operators not supported
    ```

  - 不支持带点的标识符名称。
- 目前，Babelfish 不支持 `CREATE FULLTEXT INDEX` 子句的以下选项。
  - [TYPE COLUMN type_column_name]
  - [LANGUAGE language_term]
  - [STATISTICAL_SEMANTICS]
  - 目录文件组选项
  - with 选项
- 不支持创建全文目录。创建全文索引不需要全文目录。
- `CREATE FULLTEXT INDEX` 不支持带点的标识符名称。
- Babelfish 目前不支持在搜索字符串中使用连续的特殊字符。否则，您将收到以下错误消息：

  ```
  Consecutive special characters in the full-text search condition are not currently
   supported in Babelfish
  ```

Babelfish 支持地理空间数据类型

从 3.5.0 和 4.1.0 版本开始，Babelfish 包括对以下两种空间数据类型的支持：

- 几何数据类型 – 这种数据类型用于存储平面或欧几里得（扁平地球）数据。
- 地理数据类型 – 这种数据类型用于存储椭圆或圆地球数据，例如 GPS 纬度和经度坐标。

这些数据类型允许存储和操作空间数据，但有限制。

了解 Babelfish 中的地理空间数据类型

- 各种数据库对象（例如视图、过程和表）都支持地理空间数据类型。

- 支持二维点数据类型，以便将位置数据存储为由纬度、经度和有效的空间参考系统标识符（SRID）定义的点。
- 通过 JDBC、ODBC、DOTNET 和 PYTHON 等驱动程序连接到 Babelfish 的应用程序可以利用此地理空间功能。

Babelfish 中支持的几何数据类型函数

- STGeomFromText (***geometry_tagged_text***, SRID) – 使用熟知文本（WKT）表示法创建几何实例。
- STPointFromText (***point_tagged_text***, SRID) – 使用 WKT 表示法创建点实例。
- Point (X, Y, SRID) – 使用 x 和 y 坐标的浮点值创建点实例。
- <geometry_instance>.STAsText ( ) – 从几何实例中提取 WKT 表示。
- <geometry_instance>.STDistance (other_geometry) – 计算两个几何实例之间的距离。
- <geometry_instance>.STX – 提取几何实例的 X 坐标（经度）。
- <geometry_instance>.STY – 提取几何实例的 Y 坐标（纬度）。

Babelfish 支持的地理数据类型函数

- STGeomFromText (***geography_tagged_text***, SRID) – 使用 WKT 表示法创建地理实例。
- STPointFromText (***point_tagged_text***, SRID) – 使用 WKT 表示法创建点实例。
- Point (Lat, Long, SRID) – 使用纬度和经度的浮点值创建点实例。
- <geography_instance>.STAsText ( ) – 从地理实例中提取 WKT 表示。
- <geography_instance>.STDistance (other_geography) – 计算两个地理实例之间的距离。
- <geography_instance>.Lat – 提取地理实例的纬度值。
- <geography_instance>.Long – 提取地理实例的经度值。

Babelfish 中对地理空间数据类型的限制

- 目前，Babelfish 不支持更高级功能，例如用于地理空间数据类型的点实例的 Z-M 标志。
- 目前不支持点实例以外的几何类型：
  - LineString
  - CircularString

  - CompoundCurve
  - Polygon
  - CurvePolygon
  - MultiPoint
  - MultiLineString
  - MultiPolygon
  - GeometryCollection
- 目前，地理空间数据类型不支持空间索引。
- 目前，这些数据类型仅支持这些列出的函数。有关更多信息，请参阅 Babelfish 中支持的几何数据类型函数和 Babelfish 支持的地理数据类型函数。
- 与 T-SQL 相比，地理数据的 STDistance 函数输出可能存在细微的精度变化。这是由于底层的 PostGIS 实现。有关更多信息，请参阅 ST_Distance。
- 为了获得最佳性能，可使用内置的地理空间数据类型，而无需在 Babelfish 中创建额外的抽象层。

> **Tip**
>
> 虽然您可以创建自定义数据类型，但不建议基于地理空间数据进行创建。这可能会带来复杂性，并且由于支持有限，可能会导致意外行为。

- 在 Babelfish 中，地理空间函数名称用作关键字，只有在按预期方式使用时才会执行空间运算。

> **Tip**
>
> 在 Babelfish 中创建用户定义的函数和过程时，请避免使用与内置地理空间函数相同的名称。如果您有任何同名的现有数据库对象，请使用 `sp_rename` 重命名它们。

了解 Babelfish 中的分区

从版本 4.3.0 开始，Babelfish 引入了表和索引分区，但支持受限。以下各节提供了有关在 Babelfish 中创建分区函数、定义分区方案以及实现分区表和索引的详细信息。

主题

- Babelfish 中的分区简介
- 限制和解决方法

Babelfish 中的分区简介

- 分区函数：
  - `CREATE PARTITION FUNCTION`：通过指定分区列类型和每个分区的值范围，定义如何对表或索引进行分区。
  - `DROP PARTITION FUNCTION`：移除现有的分区函数。
- 分区方案：
  - `CREATE PARTITION SCHEME`：定义分区和文件组之间的映射。

    > **Note**
    >
    > 在 Babelfish 中，文件组被视为虚拟对象，并不表示物理存储位置。

  - `DROP PARTITION SCHEME`：移除现有的分区方案。
- 系统函数：
  - `$PARTITION`：此系统函数返回指定分区表的某个分区列中的指定值所属的分区编号。
- 分区表和索引：
  - `CREATE TABLE ... ON partition_scheme_name (partition_column_name)`：根据指定的分区方案和分区列创建分区表。
  - `CREATE INDEX ... ON partition_scheme_name (partition_column_name)`：根据指定的分区方案和分区列创建分区索引。
- 用于对元数据进行分区的系统视图：

  添加了以下系统视图来提供与分区相关的元数据：

  - `sys.destination_data_spaces`
  - `sys.partitions`
  - `sys.partition_functions`
  - `sys.partition_parameters`
  - `sys.partition_range_values`
  - `sys.partition_schemes`

限制和解决方法

Babelfish 尚不支持以下 SQL Server 分区功能：

- `ALTER PARTITION FUNCTION` 和 `ALTER PARTITION SCHEME`

> **Note**
>
> Babelfish 不支持拆分和合并操作。创建时在分区函数中定义所有分区，因为您以后无法添加或移除分区。

- 计算列作为分区列。
- 分区表的 `INSERT BULK` 和 BCP 实用程序。
- 分区函数的 `LEFT` 边界选项。
- 分区函数的 `SQL_VARIANT` 数据类型。
- `TRUNCATE TABLE ... WITH PARTITION`.
- `ALTER TABLE ... SWITCH PARTITION`.
- 未对齐的分区索引，例如与分区表不同的分区方案和分区列。
- 只有分区表上的完全加载任务才支持从 Babelfish 源迁移 DMS。
- 在分区函数中使用排序规则。
- 使用具有默认数据库排序规则以外的排序规则的分区列。
- Babelfish 不支持这些语法选项，但提供了解决方法：
  - 在 CREATE TABLE 语句中使用带有约束或索引的分区方案。
  - ALTER TABLE ... ADD CONSTRAINT ... ON partition_scheme_name (partition_column_name)。

> **Note**
>
> 可以将 `babelfishpg_tsql.escape_hatch_storage_on_partition` 转义孵化设置为忽略。这将支持解析器忽略与约束或索引一起使用的分区方案选项，并且后端将为每个分区创建单独的约束或索引。

Babelfish 问题排查

下文介绍了某些 Babelfish 数据库集群问题的排查思路和解决方法。

主题

- 连接失败

连接失败

导致新 Aurora 数据库集群与 Babelfish 连接失败的常见原因包括：

- 安全组不允许访问 – 如果您在连接到 Babelfish 时遇到问题，请确保将 IP 地址添加到原定设置的 Amazon EC2 安全组中。您可以使用 https://checkip.amazonaws.com/ 以确定您的 IP 地址，然后将其添加到 TDS 端口和 PostgreSQL 端口的入站规则中。有关更多信息，请参阅《Amazon EC2 用户指南》中的向安全组添加规则。
- SSL 配置不匹配 – 如果在 Aurora PostgreSQL 上启用 `rds.force_ssl` 参数（设置为 1），则客户端必须通过 SSL 连接到 Babelfish。如果您的客户端设置不正确，则您会看到错误消息，如下所示：

  ```
  Cannot connect to your-Babelfish-DB-cluster, 1433
  ----------------------
  ADDITIONAL INFORMATION:
  no pg_hba_conf entry for host "256.256.256.256", user "your-user-name",
  "database babelfish_db", SSL off (Microsoft SQL Server, Error: 33557097)
  ...
  ```

  此错误表明您的本地客户端和 Babelfish 数据库集群之间可能存在 SSL 配置问题，并且集群要求客户端使用 SSL（其 `rds.force_ssl` 参数设置为 1）。有关配置 SSL 的更多信息，请参阅《Amazon RDS 用户指南》中的将 SSL 与 PostgreSQL 数据库实例结合使用。

  如果您使用 SQL Server Management Studio (SSMS) 连接到 Babelfish 并且看到此错误，则可以选择 Connection Properties（连接属性）窗格中的 Encrypt connection（加密连接）和 Trust server certificate（信任服务器证书）连接选项，然后重试。这些设置处理 SSMS 的 SSL 连接要求。

有关 Aurora 连接问题排查的更多信息，请参阅 无法连接到 Amazon RDS 数据库实例。

关闭 Babelfish

当您不再需要 Babelfish 时，可以关闭 Babelfish 功能。

请注意一些注意事项：

- 在某些情况下，您可能会在迁移到 Aurora PostgreSQL 之前关闭 Babelfish。如果您这样做且您的 DDL 依赖于 SQL Server 数据类型，或者您在代码中使用任何 T-SQL 功能，您的代码将失败。
- 如果您在预置 Babelfish 实例后关闭了 Babelfish 扩展，则无法在同一集群上再次预置相同的数据库。

要关闭 Babelfish，请将您的参数组设置 `rds.babelfish_status` 修改为 `OFF`。您可以在关闭 Babelfish 的情况下继续使用 SQL Server 数据类型，方法将 `rds.babelfish_status` 设置为 `datatypesonly`。

如果您在参数组中关闭 Babelfish，则使用该参数组的所有集群都将失去 Babelfish 功能。

有关修改参数组的更多信息，请参阅 Amazon Aurora 的参数组。有关特定于 Babelfish 的参数的信息，请参阅 Babelfish 的数据库集群参数组设置。

管理适用于 Aurora PostgreSQL 的 Babelfish 版本更新

Babelfish 可用于 Aurora PostgreSQL 13.4 及更高版本。Babelfish 更新随 Aurora PostgreSQL 数据库引擎的某些新版本提供。有关更多信息，请参阅 Aurora PostgreSQL 版本注释。

> **Note**
>
> 在 Aurora PostgreSQL 13 的任何版本上运行的 Babelfish 数据库集群都无法升级到 Aurora PostgreSQL 14.3、14.4 和 14.5。此外，Babelfish 不支持从 13.x 直接升级到 15.x。您必须先将 13.x 数据库集群升级到 14.6 及更高版本，然后升级到 15.x 版本。

有关不同 Babelfish 版本中支持的功能的列表，请参阅 Babelfish 的各个版本支持的功能。

有关当前不受支持的功能的列表，请参阅 Babelfish 中不支持的功能。

可以使用 describe-db-engine-versions AWS CLI 命令，通过查询 AWS 区域来获取支持 Babelfish 的 Aurora PostgreSQL 版本列表，如下所示：

对于 Linux、macOS 或 Unix：

```
$ aws rds describe-db-engine-versions --region us-east-1 \
    --engine aurora-postgresql \
    --query '*[]|[?SupportsBabelfish==`true`].[EngineVersion]' \
    --output text
```

有关更多信息，请参阅 AWS CLI 命令参考 中的 describe-db-engine-versions。

在以下主题中，您可以了解如何识别 Aurora PostgreSQL 数据库集群上运行的 Babelfish 版本，以及如何升级到新版本。

目录

- 确定您的 Babelfish 版本
- 将 Babelfish 集群升级到新版本
  - 将 Babelfish 升级到新的次要版本
  - 将 Babelfish 升级到新的主要版本
    - 在将 Babelfish 升级到新的主要版本之前
    - 执行主要版本升级

    - 在升级到新的主要版本之后
    - 示例：将 Babelfish 数据库集群升级到主要版本
- 使用 Babelfish 产品版本参数
    - 配置 Babelfish 产品版本参数
    - 受影响的查询和参数
    - 带有 babelfishpg_tsql.version 参数的接口

确定您的 Babelfish 版本

您可以查询 Babelfish 以查找有关 Babelfish 版本、Aurora PostgreSQL 版本和兼容的 Microsoft SQL Server 版本的详细信息。您可以使用 TDS 端口或 PostgreSQL 端口。

- To use the TDS port to query for version information
- To use the PostgreSQL port to query for version information

使用 TDS 端口查询版本信息

1. 使用 `sqlcmd` 或 `ssms` 连接到 Babelfish 数据库集群的端点。

```
sqlcmd -S bfish_db.cluster-123456789012.aws-region.rds.amazonaws.com,1433 -U
    login-id -P password -d db_name
```

2. 要确定 Babelfish 版本，请运行以下查询：

```
1> SELECT CAST(serverproperty('babelfishversion') AS VARCHAR)
2> GO
```

   该查询返回的结果类似于以下内容：

```
serverproperty
------------------------------
3.4.0

(1 rows affected)
```

3. 要确定 Aurora PostgreSQL 数据库集群的版本，请运行以下查询：

```
1> SELECT aurora_version() AS aurora_version
```

```
2> GO
```

该查询返回的结果类似于以下内容：

```
aurora_version


------------------------------------------------
15.5.0

(1 rows affected)
```

4. 要确定兼容的 Microsoft SQL Server 版本，请运行以下查询：

```
1> SELECT @@VERSION AS version
2> GO
```

该查询返回的结果类似于以下内容：

```
Babelfish for Aurora PostgreSQL with SQL Server Compatibility - 12.0.2000.8
Dec 7 2023 09:43:06
Copyright (c) Amazon Web Services
PostgreSQL 15.5 on x86_64-pc-linux-gnu (Babelfish 3.4.0)

(1 rows affected)
```

作为显示 Babelfish 和 Microsoft SQL Server 之间的一个细微差异的示例，您可以运行以下查询。在 Babelfish 上，查询返回 1，而在 Microsoft SQL Server 上时，查询返回 NULL。

```
SELECT CAST(serverproperty('babelfish') AS VARCHAR) AS runs_on_babelfish
```

您也可以使用 PostgreSQL 端口来获取版本信息，如以下过程所示。

使用 PostgreSQL 端口查询版本信息

1. 使用 `psql` 或 pgAdmin 连接到 Babelfish 数据库集群的端点。

```
psql host=bfish_db.cluster-123456789012.aws-region.rds.amazonaws.com
     port=5432 dbname=babelfish_db user=sa
```

2. 开启 psql 的扩展功能 (\x) 以获得更具可读性的输出。

```
babelfish_db=> \x
babelfish_db=> SELECT
babelfish_db=> aurora_version() AS aurora_version,
babelfish_db=> version() AS postgresql_version,
babelfish_db=> sys.version() AS Babelfish_compatibility,
babelfish_db=> sys.SERVERPROPERTY('BabelfishVersion') AS Babelfish_Version;
```

查询返回类似于以下内容的输出：

```
-[ RECORD 1 ]-----------
+-------------------------------------------------------------------------------------------
aurora_version          | 15.5.0
postgresql_version      | PostgreSQL 15.5 on x86_64-pc-linux-gnu, compiled by
 x86_64-pc-linux-gnu-gcc (GCC) 9.5.0, 64-bit
babelfish_compatibility | Babelfish for Aurora Postgres with SQL Server
 Compatibility - 12.0.2000.8                    +
                         | Dec 7 2023 09:43:06
                                    +
                         | Copyright (c) Amazon Web Services
                                     +
                         | PostgreSQL 15.5 on x86_64-pc-linux-gnu (Babelfish 3.4.0)
babelfish_version       | 3.4.0
```

将 Babelfish 集群升级到新版本

Babelfish 的新版本将在 Aurora PostgreSQL 数据库引擎 13.4 版之后的一些新版本中提供。每个新版本的 Babelfish 都有自己的版本号。与 Aurora PostgreSQL 一样，Babelfish 对版本使用 *major.minor.patch* 命名方案。例如，第一个 Babelfish 版本（即 Babelfish 1.0.0）作为 Aurora PostgreSQL 13.4.0 的一部分提供。

Babelfish 不需要单独的安装过程。如 创建 Babelfish for Aurora PostgreSQL 数据库集群 中所述，Turn on Babelfish（打开 Babelfish）是您在创建 Aurora PostgreSQL 数据库集群时选择的选项。

同样，您不能独立于支持的 Aurora 数据库集群升级 Babelfish。要将现有适用于 Aurora PostgreSQL 的 Babelfish 数据库集群升级到新版本的 Babelfish，请将 Aurora PostgreSQL 数据库集群升级到支持要使用的 Babelfish 版本的新版本。您遵循的升级过程取决于支持 Babelfish 部署的 Aurora PostgreSQL 版本，如下所示。

主要版本升级

您必须先将以下 Aurora PostgreSQL 版本升级到 Aurora PostgreSQL 14.6 及更高版本，然后才能升级到 Aurora PostgreSQL 15.2 版本。

- Aurora PostgreSQL 13.8 及所有更高版本
- Aurora PostgreSQL 13.7.1 及所有更高次要版本
- Aurora PostgreSQL 13.6.4 及所有更高次要版本

您可以将 Aurora PostgreSQL 14.6 及更高版本升级到 Aurora PostgreSQL 15.2 及更高版本。

将 Aurora PostgreSQL 数据库集群升级到新的主要版本涉及到几项初步任务。有关更多信息，请参阅 执行主要版本升级。要成功升级适用于 Aurora PostgreSQL 的 Babelfish 数据库集群，您需要为新的 Aurora PostgreSQL 版本创建自定义数据库集群参数组。这个新参数组必须包含与您正在升级的集群相同的 Babelfish 参数设置。有关更多信息以及主要版本升级源和目标的表，请参阅 将 Babelfish 升级到新的主要版本。

次要版本升级和补丁

次要版本和补丁不要求为升级创建新的数据库集群参数组。次要版本和补丁可以使用次要版本升级过程，无论是自动应用还是手动应用。有关更多信息以及版本源和目标的表，请参阅 将 Babelfish 升级到新的次要版本。

> **Note**
>
> 在执行主要或次要升级之前，将所有待处理的维护任务应用于适用于 Aurora PostgreSQL 的 Babelfish 集群。

主题

- 将 Babelfish 升级到新的次要版本
- 将 Babelfish 升级到新的主要版本

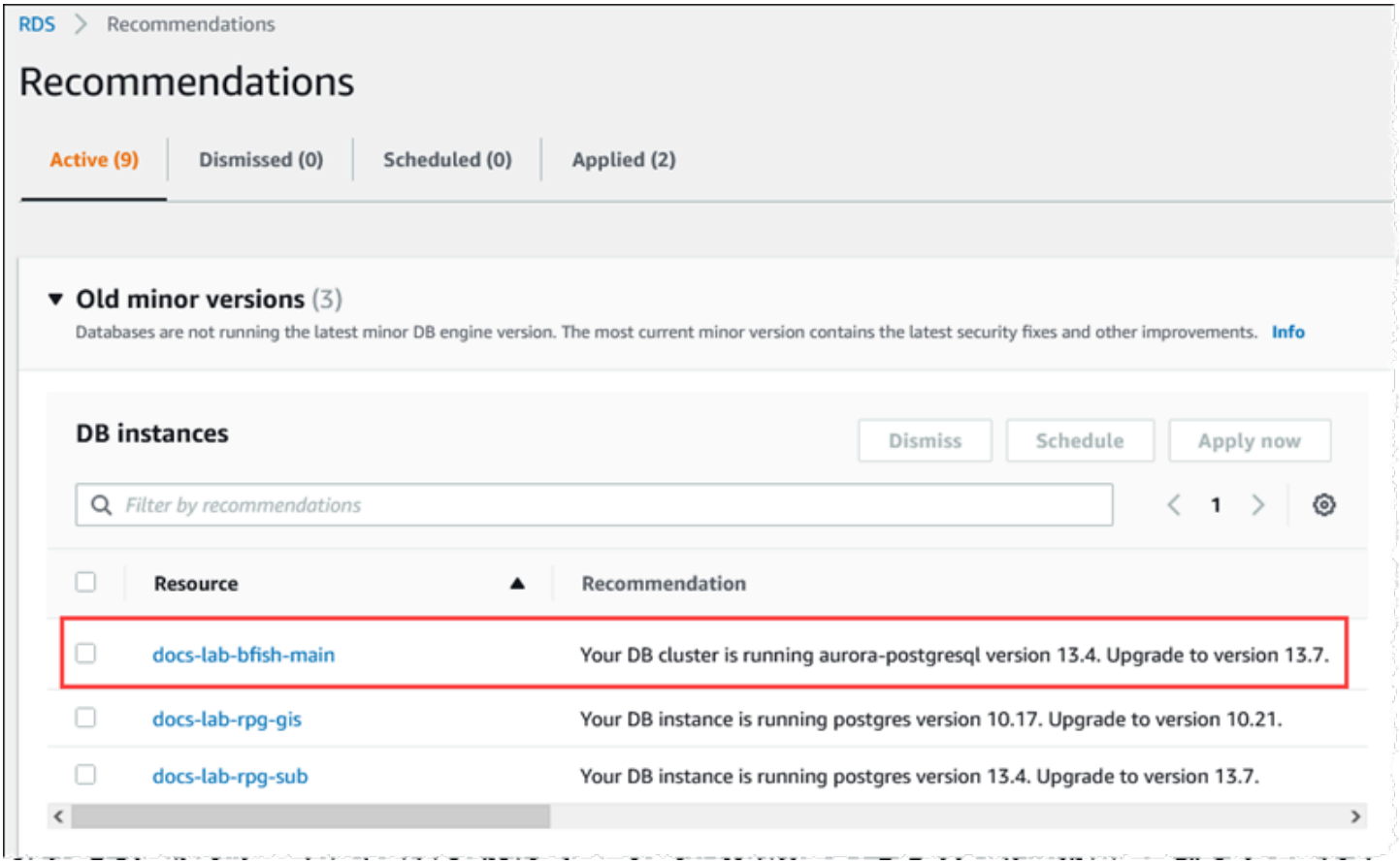
将 Babelfish 升级到新的次要版本

次要版本更新旨在保持向后兼容性。但是，在某些情况下，关键安全修复或重要错误修复可能无法完全向后兼容。补丁版本包括对发布后的次要版本的重要修复。例如，Aurora PostgreSQL 13.4 的第一个版本的版本标签是 Aurora PostgreSQL 13.4.0。迄今为止，该次要版本的多个补丁已经发布，包

括 Aurora PostgreSQL 13.4.1、13.4.2 和 13.4.4。您可以在每个 Aurora PostgreSQL 版本的 Aurora PostgreSQL 发布说明顶部的 Patch releases（补丁版本）列表中找到该版本的可用补丁。有关示例，请参阅《Aurora PostgreSQL 发布说明》中的 PostgreSQL 14.3。

如果为 Aurora PostgreSQL 数据库集群配置了 Auto minor version upgrade（自动次要版本升级）选项，则适用于 Aurora PostgreSQL 的 Babelfish 数据库集群将在集群的维护时段内自动升级。要了解有关自动次要版本升级（AmVU）及其使用方法的更多信息，请参阅 Aurora 数据库集群的自动次要版本升级。如果您的集群没有使用 AmVU，您可以通过响应维护任务或修改集群以使用新版本，手动将适用于 Aurora PostgreSQL 的 Babelfish 数据库集群升级到新的次要版本。

当您选择要安装的 Aurora PostgreSQL 版本以及在 AWS Management Console 中查看现有的 Aurora PostgreSQL 数据库集群时，版本仅显示 *major.minor* 数字。例如，控制台上的以下图片显示了现有的适用于 Aurora PostgreSQL 的 Babelfish 数据库集群（具有 Aurora PostgreSQL 13.4），该图建议将该集群升级到 Aurora PostgreSQL 的新次要版本，即版本 13.7。



要获取完整的版本详细信息，包括 *patch* 级别，您可以使用 `aurora_version` Aurora PostgreSQL 函数查询 Aurora PostgreSQL 数据库集群。有关更多信息，请参阅Aurora PostgreSQL 函数参考 中的 aurora_version。您可以在 确定您的 Babelfish 版本 的 To use the PostgreSQL port to query for version information 过程中找到使用该函数的示例。

下表显示 Aurora PostgreSQL 和 Babelfish 版本以及可以支持次要版本升级过程的可用目标版本。

当前源版本	最新升级目标
Aurora PostgreSQL（Babelfish）	Aurora PostgreSQL（Babelfish）
17.4（5.1）	17.5（5.2）
16.8（4.5）	16.9（4.6）
16.6（4.4.0）	16.9（4.6）、16.8（4.5.0）
16.4（4.3.0）	16.9（4.6）、16.8（4.5.0）、16.6（4.4.0）
16.3（4.2.0）	16.9（4.6）、16.8（4.5.0）、16.6（4.4.0）、16.4（4.3.0）
16.2（4.1.0）	16.9（4.6）、16.8（4.5.0）、16.6（4.4.0）、16.4（4.3.0）、16.3（4.2.0）
16.1（4.0.0）	16.9（4.6）、16.8（4.5.0）、16.6（4.4.0）、16.4（4.3.0）、16.3（4.2.0）、16.2（4.1.0）
15.12（3.9.0）	15.13（3.10）
15.10（3.8.0）	15.13（3.10）、15.12（3.9.0）
15.8（3.7.0）	15.13（3.10）、15.12（3.9.0）、15.10（3.8.0）
15.7（3.6.0）	15.13（3.10）、15.12（3.9.0）、15.10（3.8.0）、15.8（3.7.0）
15.6（3.5.0）	15.13（3.10）、15.12（3.9.0）、15.10（3.8.0）、15.8（3.7.0）、15.7（3.6.0）
15.5（3.4.0）	15.13（3.10）、15.12（3.9.0）、15.10（3.8.0）、15.8（3.7.0）、15.7（3.6.0）、15.6（3.5.0）

当前源版本	最新升级目标
15.4（3.3.0）	15.13（3.10）、15.12（3.9.0）、15.10（3.8.0）、15.8（3.7.0）、15.7（3.6.0）、15.6（3.5.0）、15.5（3.4.0）
15.12（3.9.0）、15.3（3.2.0）	15.13（3.10）、15.12（3.9.0）、15.10（3.8.0）、15.8（3.7.0）、15.7（3.6.0）、15.6（3.5.0）、15.5（3.4.0）、15.4（3.3.0）
15.2（3.1.0）	15.13（3.10）、15.12（3.9.0）、15.10（3.8.0）、15.8（3.7.0）、15.7（3.6.0）、15.6（3.5.0）、15.5（3.4.0）、15.4（3.3.0）、15.3（3.2.0）
14.17（2.12.0）	14.18（2.13.0）
14.15（2.11.0）	14.18（2.13.0）、14.17（2.12.0）
14.13（2.10.0）	14.18（2.13.0）、14.17（2.12.0）、14.15（2.11.0）
14.12（2.9.0）	14.18（2.13.0）、14.17（2.12.0）、14.15（2.11.0）、14.13（2.10.0）
14.11（2.8.0）	14.18（2.13.0）、14.17（2.12.0）、14.15（2.11.0）、14.13（2.10.0）、14.12（2.9.0）
14.10（2.7.0）	14.18（2.13.0）、14.17（2.12.0）、14.15（2.11.0）、14.13（2.10.0）、14.12（2.9.0）、14.11（2.8.0）
14.9（2.6.0）	14.18（2.13.0）、14.17（2.12.0）、14.15（2.11.0）、14.13（2.10.0）、14.12（2.9.0）、14.11（2.8.0）、14.10（2.7.0）

当前源版本	最新升级目标
14.8（2.5.0）	14.18（2.13.0）、14.17（2.12.0）、14.15（2.11.0）、14.13（2.10.0）、14.12（2.9.0）、14.11（2.8.0）、14.10（2.7.0）、14.9（2.6.0）
14.7（2.4.0）	14.18（2.13.0）、14.17（2.12.0）、14.15（2.11.0）、14.13（2.10.0）、14.12（2.9.0）、14.11（2.8.0）、14.10（2.7.0）、14.9（2.6.0）、14.8（2.5.0）
14.6（2.3.0）	14.18（2.13.0）、14.17（2.12.0）、14.15（2.11.0）、14.13（2.10.0）、14.12（2.9.0）、14.11（2.8.0）、14.10（2.7.0）、14.9（2.6.0）、14.8（2.5.0）、14.7（2.4.0）
14.5（2.2.0）	14.18（2.13.0）、14.17（2.12.0）、14.15（2.11.0）、14.13（2.10.0）、14.12（2.9.0）、14.11（2.8.0）、14.10（2.7.0）、14.9（2.6.0）、14.8（2.5.0）、14.7（2.4.0）、14.6（2.3.0）
14.3（2.1.0）	14.18（2.13.0）、14.6（2.3.0）
13.8（1.4.0）	13.9（1.5.0）
13.7（1.3.0）	13.9（1.5.0）、13.8（1.4.0）

将 Babelfish 升级到新的主要版本

要进行主要版本升级，您需要先将适用于 Aurora PostgreSQL 的 Babelfish 数据库集群升级到支持主要版本升级的版本。要实现此目的，请对数据库集群应用补丁更新或次要版本升级。有关更多信息，请参阅 将 Babelfish 升级到新的次要版本。

下表显示了可以支持主要版本升级的 Aurora PostgreSQL 版本和 Babelfish 版本。

当前源版本	最新升级目标
Aurora PostgreSQL（Babelfish）	Aurora PostgreSQL（Babelfish）
16.9（4.6.0）	17.5（5.2.0）
16.8（4.5.0）	17.5（5.2.0）、17.4（5.1.0）
16.6（4.4.0）	17.5（5.2.0）、17.4（5.1.0）
16.4（4.3.0）	17.5（5.2.0）、17.4（5.1.0）
16.3（4.2.0）	17.5（5.2.0）、17.4（5.1.0）
16.2（4.1.0）	17.5（5.2.0）、17.4（5.1.0）
16.1（4.0.0）	17.5（5.2.0）、17.4（5.1.0）
15.13（3.10）	17.5（5.2.0） 16.9（4.6.0）
15.12（3.9.0）	17.5（5.2.0）、17.4（5.1.0） 16.9（4.6.0）、16.8（4.5.0）
15.10（3.8.0）	17.5（5.2.0）、17.4（5.1.0） 16.9（4.6.0）、16.8（4.5.0）、16.6（4.4.0）
15.8（3.7.0）	17.5（5.2.0）、17.4（5.1.0） 16.9（4.6.0）、16.8（4.5.0）、16.6（4.4.0）、16.4（4.3.0）
15.7（3.6.0）	17.5（5.2.0）、17.4（5.1.0） 16.9（4.6.0）、16.8（4.5.0）、16.6（4.4.0）、16.4（4.3.0）、16.3（4.2.0）
15.6（3.5.0）	17.5（5.2.0）、17.4（5.1.0）

当前源版本	最新升级目标
	16.9（4.6.0）、16.8（4.5.0）、16.6（4.4.0）、16.4（4.3.0）、16.3（4.2.0）、16.2（4.1.0）
15.5（3.4.0）	17.5（5.2.0）、17.4（5.1.0） 16.9（4.6.0）、16.8（4.5.0）、16.6（4.4.0）、16.4（4.3.0）、16.3（4.2.0）、16.2（4.1.0）、16.1（4.0.0）
15.4（3.3.0）	17.5（5.2.0）、17.4（5.1.0） 16.9（4.6.0）、16.8（4.5.0）、16.6（4.4.0）、16.4（4.3.0）、16.3（4.2.0）、16.2（4.1.0）、16.1（4.0.0）
15.3（3.2.0）	17.5（5.2.0）、17.4（5.1.0） 16.9（4.6.0）、16.8（4.5.0）、16.6（4.4.0）、16.4（4.3.0）、16.3（4.2.0）、16.2（4.1.0）、16.1（4.0.0）
15.2（3.1.0）	17.5（5.2.0）、17.4（5.1.0） 16.9（4.6.0）、16.8（4.5.0）、16.6（4.4.0）、16.4（4.3.0）、16.3（4.2.0）、16.2（4.1.0）、16.1（4.0.0）
14.18（2.13.0）	17.5（5.2.0） 16.9（4.6.0） 15.13（3.10.0）
14.17（2.12.0）	17.5（5.2.0）、17.4（5.1.0） 16.9（4.6.0）、16.8（4.5.0） 15.13（3.10.0）、15.12（3.9.0）

当前源版本	最新升级目标
14.15（2.11.0）	17.5（5.2.0）、17.4（5.1.0） 16.9（4.6.0）、16.8（4.5.0）、16.6（4.4.0） 15.13（3.10.0）、15.12（3.9.0）、15.10（3.8.0）
14.13（2.10.0）	17.5（5.2.0）、17.4（5.1.0） 16.9（4.6.0）、16.8（4.5.0）、16.6（4.4.0）、16.4（4.3.0） 15.13（3.10.0）、15.12（3.9.0）、15.10（3.8.0）、15.8（3.7.0）
14.12（2.9.0）	17.5（5.2.0）、17.4（5.1.0） 16.9（4.6.0）、16.8（4.5.0）、16.6（4.4.0）、16.4（4.3.0）、16.3（4.2.0） 15.13（3.10.0）、15.12（3.9.0）、15.10（3.8.0）、15.8（3.7.0）、15.7（3.6.0）
14.11（2.8.0）	17.5（5.2.0）、17.4（5.1.0） 16.9（4.6.0）、16.8（4.5.0）、16.6（4.4.0）、16.4（4.3.0）、16.3（4.2.0）、16.2（4.1.0） 15.13（3.10.0）、15.12（3.9.0）、15.10（3.8.0）、15.8（3.7.0）、15.7（3.6.0）、15.6（3.5.0）

当前源版本	最新升级目标
14.10（2.7.0）	17.5（5.2.0）、17.4（5.1.0） 16.9（4.6.0）、16.8（4.5.0）、16.6（4.4.0）、16.4（4.3.0）、16.3（4.2.0）、16.2（4.1.0）、16.1（4.0.0） 15.13（3.10.0）、15.12（3.9.0）、15.10（3.8.0）、15.8（3.7.0）、15.7（3.6.0）、15.6（3.5.0）、15.5（3.4.0）
14.9（2.6.0）	17.5（5.2.0）、17.4（5.1.0） 16.9（4.6.0）、16.8（4.5.0）、16.6（4.4.0）、16.4（4.3.0）、16.3（4.2.0）、16.2（4.1.0）、16.1（4.0.0） 15.13（3.10.0）、15.12（3.9.0）、15.10（3.8.0）、15.8（3.7.0）、15.7（3.6.0）、15.6（3.5.0）、15.5（3.4.0）、15.4（3.3.0）
14.8（2.5.0）	17.5（5.2.0）、17.4（5.1.0） 16.9（4.6.0）、16.8（4.5.0）、16.6（4.4.0）、16.4（4.3.0）、16.3（4.2.0）、16.2（4.1.0）、16.1（4.0.0） 15.13（3.10.0）、15.12（3.9.0）、15.10（3.8.0）、15.8（3.7.0）、15.7（3.6.0）、15.6（3.5.0）、15.5（3.4.0）、15.4（3.3.0）、15.3（3.2.0）

当前源版本	最新升级目标
14.7 (2.4.0)	17.5 (5.2.0) 、17.4 (5.1.0) 16.9 (4.6.0) 、16.8 (4.5.0) 、16.6 (4.4.0) 、16.4 (4.3.0) 、16.3 (4.2.0) 、16.2 (4.1.0) 、16.1 (4.0.0) 15.13 (3.10.0) 、15.12 (3.9.0) 、15.10 (3.8.0) 、15.8 (3.7.0) 、15.7 (3.6.0) 、15.6 (3.5.0) 、15.5 (3.4.0) 、15.4 (3.3.0) 、15.3 (3.2.0) 、15.2 (3.1.0)
14.6 (2.3.0)	17.5 (5.2.0) 、17.4 (5.1.0) 16.9 (4.6.0) 、16.8 (4.5.0) 、16.6 (4.4.0) 、16.4 (4.3.0) 、16.3 (4.2.0) 、16.2 (4.1.0) 、16.1 (4.0.0) 15.13 (3.10.0) 、15.12 (3.9.0) 、15.10 (3.8.0) 、15.8 (3.7.0) 、15.7 (3.6.0) 、15.6 (3.5.0) 、15.5 (3.4.0) 、15.4 (3.3.0) 、15.3 (3.2.0) 、15.2 (3.1.0)
13.9 (1.5.0)	14.6 (2.3.0)
13.8 (1.4.0)	14.6 (2.3.0)
13.7 (1.3.0)	14.6 (2.3.0)

在将 Babelfish 升级到新的主要版本之前

升级可能涉及短暂的中断。因此，建议您在维护时段或利用率低的其他时段执行或安排升级。

在执行主要版本升级之前

1. 使用 确定您的 Babelfish 版本 中概述的命令识别现有 Aurora PostgreSQL 数据库集群的 Babelfish 版本。Aurora PostgreSQL 版本和 Babelfish 版本信息由 PostgreSQL 处理，因此请按照 To use the PostgreSQL port to query for version information 过程中详细介绍的步骤获取详细信息。

2. 验证您的版本是否支持主要版本升级。有关支持主要版本升级功能的版本列表，请参阅 将 Babelfish 升级到新的次要版本 并执行必要的升级前任务。

   例如，如果 Babelfish 版本在 Aurora PostgreSQL 13.5 数据库集群上运行并且您要升级到 Aurora PostgreSQL 15.2，则首先应用所有次要版本和补丁，以将集群升级到 Aurora PostgreSQL 14.6 或更高版本。当您的集群版本为 14.6 或更高版本时，继续执行主要版本升级过程。

3. 创建当前 Babelfish 数据库集群的手动快照作为备份。备份允许您将集群还原到其 Aurora PostgreSQL 版本、Babelfish 版本，并将所有数据还原到升级前的状态。有关更多信息，请参阅创建数据库集群快照。如果您决定将此集群还原到其升级前的状态，请务必保留现有的自定义数据库集群参数组以供再次使用。有关更多信息，请参阅从数据库集群快照还原和参数组注意事项。

4. 为目标 Aurora PostgreSQL 数据库版本准备自定义数据库集群参数组。从您当前的适用于 Aurora PostgreSQL 的 Babelfish 数据库集群中复制 Babelfish 参数的设置。要找到所有 Babelfish 参数的列表，请参阅 Babelfish 的数据库集群参数组设置。对于主要版本升级，以下参数需要与源数据库集群相同的设置。要使升级获得成功，所有设置都必须相同。

   - rds.babelfish_status
   - babelfishpg_tds.tds_default_numeric_precision
   - babelfishpg_tds.tds_default_numeric_scale
   - babelfishpg_tsql.database_name
   - babelfishpg_tsql.default_locale
   - babelfishpg_tsql.migration_mode
   - babelfishpg_tsql.server_collation_name

   > **Warning**
   >
   > 如果新 Aurora PostgreSQL 版本的自定义数据库集群参数组中 Babelfish 参数的设置与您正在升级的集群的参数值不匹配，则 `ModifyDBCluster` 操作将失败。`InvalidParameterCombination` 错误消息出现在 AWS Management Console 或 `modify-db-cluster` AWS CLI 命令的输出中。

5. 使用 AWS Management Console 或 AWS CLI 创建自定义数据库集群参数组。为您要升级的 Aurora PostgreSQL 版本选择适用的 Aurora PostgreSQL 系列。

> **Tip**
>
> 参数组在 AWS 区域 级别进行管理。使用 AWS CLI 时，您可以使用默认区域进行配置，而不是在命令中指定 `--region`。要了解有关使用 AWS CLI 的更多信息，请参阅《AWS Command Line Interface 用户指南》中的快速设置。

执行主要版本升级

1. 将 Aurora PostgreSQL 数据库集群升级到新的主要版本。有关更多信息，请参阅 将 Aurora PostgreSQL 引擎升级到新的主要版本。
2. 重启集群的写入器实例，以便参数设置生效。

在升级到新的主要版本之后

在主要版本升级到新的 Aurora PostgreSQL 版本后，带有 `IDENTITY` 列的表中的 `IDENTITY` 值可能比升级前的值更大（+32）。结果是，当将下一行插入到此类表中时，生成的身份列值会跳至 +32 数字并从那里开始序列。这种情况不会对 Babelfish 数据库集群的功能产生负面影响。但如果需要，可以根据列的最大值重置序列对象。为此，请使用 `sqlcmd` 或另一个 SQL Server 客户端连接到 Babelfish 写入器实例上的 T-SQL 端口。有关更多信息，请参阅 使用 SQL Server 客户端连接到数据库集群。

```
sqlcmd -S bfish-db.cluster-123456789012.aws-region.rds.amazonaws.com,1433 -U
     sa -P ******** -d dbname
```

连接后，使用以下 SQL 命令生成可用于为关联序列对象做种子的语句。此 SQL 命令同时适用于单一数据库和多数据库 Babelfish 配置。有关这两种部署模型的更多信息，请参阅 将 Babelfish 与单个数据库或多个数据库结合使用。

```
DECLARE @schema_prefix NVARCHAR(200) = ''
IF current_setting('babelfishpg_tsql.migration_mode') = 'multi-db'
    SET @schema_prefix = db_name() + '_'
SELECT 'SELECT setval(pg_get_serial_sequence(''' + @schema_prefix +
 schema_name(tables.schema_id)
    + '.' + tables.name + ''', ''' + columns.name + '''),(select max(' + columns.name +
 ')
    FROM ' + schema_name(tables.schema_id) + '.' + tables.name + '));
    'FROM sys.tables tables JOIN sys.columns
    columns ON tables.object_id = columns.object_id
```

```
    WHERE columns.is_identity = 1
GO
```

该查询会生成一系列 SELECT 语句，然后您可以运行这些语句来重置最大 IDENTITY 值并缩小任何差距。下面显示了使用在 Babelfish 集群上运行的示例 SQL Server 数据库 Northwind 时的输出。

```
-----------------------------------------------------
SELECT setval(pg_get_serial_sequence('northwind_dbo.categories', 'categoryid'),(select
 max(categoryid)
    FROM dbo.categories));

SELECT setval(pg_get_serial_sequence('northwind_dbo.orders', 'orderid'),(select
 max(orderid)
    FROM dbo.orders));


SELECT setval(pg_get_serial_sequence('northwind_dbo.products', 'productid'),(select
 max(productid)
    FROM dbo.products));


SELECT setval(pg_get_serial_sequence('northwind_dbo.shippers', 'shipperid'),(select
 max(shipperid)
    FROM dbo.shippers));


SELECT setval(pg_get_serial_sequence('northwind_dbo.suppliers', 'supplierid'),(select
 max(supplierid)
    FROM dbo.suppliers));


(5 rows affected)
```

逐条运行语句以重置序列值。

示例：将 Babelfish 数据库集群升级到主要版本

在此示例中，您可以找到一系列 AWS CLI 命令，这些命令解释了如何将运行 Babelfish 版本 1.2.2 的 Aurora PostgreSQL 13.6.4 数据库集群升级到 Aurora PostgreSQL 14.6。首先，为 Aurora PostgreSQL 14 创建一个自定义数据库集群参数组。接下来，修改参数值，使其与 Aurora PostgreSQL 版本 13 源中的参数值相匹配。最后，您可以通过修改源集群来执行升级。有关更

多信息，请参阅 Babelfish 的数据库集群参数组设置。在该主题中，您还可以找到有关使用 AWS Management Console 执行升级的信息。

使用 create-db-cluster-parameter-group CLI 命令为新版本创建数据库集群参数组。

对于 Linux、macOS 或 Unix：

```
aws rds create-db-cluster-parameter-group \
    --db-cluster-parameter-group-name docs-lab-babelfish-apg-14 \
    --db-parameter-group-family aurora-postgresql14 \
    --description 'New custom parameter group for upgrade to new major version' \
    --region us-west-1
```

当您发出此命令时，将在 AWS 区域 中创建自定义数据库集群参数组。您将看到类似以下内容的输出。

```
{
    "DBClusterParameterGroup": {
        "DBClusterParameterGroupName": "docs-lab-babelfish-apg-14",
        "DBParameterGroupFamily": "aurora-postgresql14",
        "Description": "New custom parameter group for upgrade to new major version",
        "DBClusterParameterGroupArn": "arn:aws:rds:us-west-1:111122223333:cluster-
pg:docs-lab-babelfish-apg-14"
    }
}
```

有关更多信息，请参阅在 Amazon Aurora 中创建数据库集群参数组。

使用 modify-db-cluster-parameter-group CLI 命令修改设置，使其与源集群相匹配。

对于 Windows：

```
aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-name docs-lab-
babelfish-apg-14 ^
  --parameters
 "ParameterName=rds.babelfish_status,ParameterValue=on,ApplyMethod=pending-reboot" ^
 "ParameterName=babelfishpg_tds.tds_default_numeric_precision,ParameterValue=38,ApplyMethod=pen
reboot" ^
 "ParameterName=babelfishpg_tds.tds_default_numeric_scale,ParameterValue=8,ApplyMethod=pending-
reboot" ^
 "ParameterName=babelfishpg_tsql.database_name,ParameterValue=babelfish_db,ApplyMethod=pending-
reboot" ^
```

```
 "ParameterName=babelfishpg_tsql.default_locale,ParameterValue=en-
US,ApplyMethod=pending-reboot" ^
 "ParameterName=babelfishpg_tsql.migration_mode,ParameterValue=single-
db,ApplyMethod=pending-reboot" ^
 "ParameterName=babelfishpg_tsql.server_collation_name,ParameterValue=sql_latin1_general_cp1_ci
reboot"
```

响应类似于以下内容。

```
{
    "DBClusterParameterGroupName": "docs-lab-babelfish-apg-14"
}
```

使用 modify-db-cluster CLI 命令修改集群，以使用新版本和新的自定义数据库集群参数组。您还可指定 `--allow-major-version-upgrade` 参数，如以下示例中所示。

```
aws rds modify-db-cluster \
--db-cluster-identifier docs-lab-bfish-apg-14 \
--engine-version 14.6 \
--db-cluster-parameter-group-name docs-lab-babelfish-apg-14 \
--allow-major-version-upgrade \
--region us-west-1 \
--apply-immediately
```

使用 reboot-db-instance CLI 命令重启集群的写入器实例，以便参数设置生效。

```
aws rds reboot-db-instance \
--db-instance-identifier docs-lab-bfish-apg-14-instance-1\
--region us-west-1
```

使用 Babelfish 产品版本参数

从 Babelfish 2.4.0 和 3.1.0 版本中引入了一个名为 `babelfishpg_tds.product_version` 的全新 Grand Unified Configuration（GUC）参数。此参数允许您将 SQL Server 产品版本号设置为 Babelfish 的输出。

该参数是一个由四部分组成的版本 ID 字符串，每个部分应以“.”分隔。

语法

```
Major.Minor.Build.Revision
```

- 主要版本：介于 11 到 16 之间的数字。
- 次要版本：介于 0 到 255 之间的数字。
- 构建版本：介于 0 到 65535 之间的数字。
- 修订版：0 和任意正数。

配置 Babelfish 产品版本参数

您必须使用集群参数组在控制台中设置 babelfishpg_tds.product_version 参数。有关如何修改数据库集群参数的更多信息，请参阅在 Amazon Aurora 中修改数据库集群参数组中的参数。

当您将产品版本参数设置为无效值时，更改将不会生效。尽管控制台可能会向您显示新值，但该参数会保留先前的值。有关错误消息的详细信息，请检查引擎日志文件。

对于 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster-parameter-group \
--db-cluster-parameter-group-name mydbparametergroup \
--parameters
 "ParameterName=babelfishpg_tds.product_version,ParameterValue=15.2.4000.1,ApplyMethod=immediat
```

对于 Windows：

```
aws rds modify-db-cluster-parameter-group ^
--db-cluster-parameter-group-name mydbparametergroup ^
--parameters
 "ParameterName=babelfishpg_tds.product_version,ParameterValue=15.2.4000.1,ApplyMethod=immediat
```

受影响的查询和参数

查询/参数	结果	生效时间
SELECT @@VERSION	返回用户定义的 SQL Server 版本（babelfishpg_tsql.version 值 = 原定设置）	立即
SELECT SERVERPROPERTY('ProductVersion')	返回用户定义的 SQL Server 版本	立即
SELECT SERVERPROPERTY('ProductMajorVersion')	返回用户定义的 SQL Server 版本的主要版本	立即
PRELOGIN 响应消息中的 VERSION 标记	服务器返回带有用户定义的 SQL Server 版本的 PRELOGIN 消息	在用户创建新会话时生效
使用 JDBC 时 LoginAck 中的 SQLServerVersion	DatabaseMetaData.getDatabaseProductVersion() 返回用户定义的 SQL Server 版本	在用户创建新会话时生效

带有 babelfishpg_tsql.version 参数的接口

您可以使用参数 babelfishpg_tsql.version 和 babelfishpg_tds.product_version 来设置 @@VERSION 的输出。以下示例显示这两个参数的接口方式。

- 当 babelfishpg_tsql.version 参数为“default”且 babelfishpg_tds.product_version 为 15.0.2000.8 时。
  - @@version 的输出 – 15.0.2000.8。
- 当 babelfishpg_tsql.version 参数设置为 13.0.2000.8 且 babelfishpg_tds.product_version 参数设置为 15.0.2000.8 时。
  - @@version 的输出 – 13.0.2000.8。

Babelfish for Aurora PostgreSQL 参考

主题

- Babelfish 的各个版本支持的功能
- Babelfish 中不支持的功能
- 使用适用于 Aurora PostgreSQL 的 Babelfish 过程

Babelfish 的各个版本支持的功能

在下表中，可以找到不同 Babelfish 版本支持的所有功能的列表。有关不受支持的功能的列表，请参阅Babelfish 中不支持的功能。有关各种 Babelfish 版本的信息，请参阅 Aurora PostgreSQL 版本注释。

Aurora 和 PostgreSQL 功能

在下表中，可以找到不同 Babelfish 版本支持的 Aurora 和 PostgreSQL 功能。

Aurora 和 PostgreSQL 功能	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
Aurora 机器学习服务	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否
使用 AWS Directory Service 通过 Kerberos 进行数据库身份验证	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否
转储和还原	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否
pg_stat_statement 扩展		是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否

Aurora 和 PostgreSQL 功能	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
	支持																															
pgvector	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否
零停机时间修补（ZDP）	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否

T-SQL 功能或语法

在下表中，可以找到不同 Babelfish 版本支持的 T-SQL 功能或语法。

T-SQL 功能或语法	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
SELECT 语句由 4 部分组成的对象名称引用	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否
用于更改数据库所有者的 ALTER AUTHORIZATION 语法	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否
ALTER DATABASE *<db_name>* MODIFY NAME = *<new_db_name>*	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否
ALTER FUNCTION		是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否

T-SQL 功能或语法	5.7	5.6	4.3	4.2	4.1	4.0	3.11	3.10	3.9	3.8	3.7	3.6	3.5	3.4	3.3	3.2	3.1	3.0	2.10	2.9	2.8	2.7	2.6	2.5	2.4	2.3	2.2	2.1	2.0	1.0
	支持																													
ALTER PROCEDURE	支持	是	是	是	是	是	是	否	否	是	是	是	是	是	是	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否
ALTER ROLE	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否	否	否	否	否	否	否	否	否	否	否
ALTER USER...WITH LOGIN	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否	否	否	否	否	否	否	否	否	否	否
ALTER VIEW	是	是	是	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否
CREATE FUNCTION 中的 AS 关键字	支持	是	是	是	是	是	是	是	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否
AT TIME ZONE 子句	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否	否	否	否	否	否	否	否	否	否	否
Babelfish 实例作为链接服务器	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否	否	否	否	否	否	否	否	否	否

| T-SQL 功能或语法 | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2

T-SQL 功能或语法	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
CREATE TRIGGER	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
CREATE OR ALTER VIEW	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否
创建唯一索引	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
跨数据库过程执行	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否
跨数据库引用 SELECT、INSERT、UPDATE、DELETE	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
光标类型的参数仅用于输入参数（不用于输出）	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
使用 bcp 客户端实用程序进行数据迁移	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是

| T-SQL 功能或语法 | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2

| T-SQL 功能或语法 | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2

T-SQL 功能或语法	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	2.1.0
GRANT permission [,...n] ON SCHEMA	支持	是	是	是	是	是	是	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否
GUC babelfish pg_tds.product_version	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否	否
带前导点字符的标识符	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
针对表的 INSTEAD OF 触发器	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
针对视图的 INSTEAD OF 触发器	支持	是	是	是	是	是	是	是	是	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否
KILL	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否
$PARTITION.partition_function_name(partition_column_value)	支持	是	是	是	是	是	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否

T-SQL 功能或语法	5.5.4.4.4.4.4.4.4.3.3.3.3.3.3.3.3.3.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.1.0
PIVOT（使用场合：4.4.0 和 3.8.0 中的视图定义，3.4.0 中的公用表表达式，3.4.0 中的 join）	支持否
REVOKE	支持是
REVOKE permission [,...n] ON SCHEMA	支持支持支持支持支持支持支持不支持否
SELECT... OFFSET...FETCH 子句	支持支持支持支持支持支持支持支持不支持否
SELECT FOR JSON AUTO	支持支持支持支持支持支持支持支持不支持否
SET BABELFISH_SHOWPLAN_ALL ON（和 OFF）	支持是
SET BABELFISH_STATISTICS PROFILE ON（OFF）	支持是

T-SQL 功能或语法	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
SET CONTEXT_INFO	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	是	是	是	是	是	是	是	否	否	否	否	否	否	否
SET LOCK_TIMEOUT	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
SET NO_BROWSETABLE	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否
SET rowcount	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	是	是	是	是	是	是	是	否	否	否	否	否	否	否
SET SHOWPLAN_ALL	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否
SET STATISTICS IO	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否

| T-SQL 功能或语法 | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2

T-SQL 功能或语法	5.5	5.4	4.4	4.4	4.4	4.4	4.4	4.4	4.3	3.3	3.3	3.3	3.3	3.3	3.3	3.3	3.3	2.2	2.2	2.2	2.2	2.2	2.2	2.2	2.2	2.2	2.2	2.2	2.2	2.0	1.0
带 LIKE 谓词的 T-SQL 方括号语法	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否
UNPIVOT	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否
存储过程调用中未加引号的字符串值和默认值	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否
VAR	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否
VARP	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否

T-SQL 内置函数

在下表中，可以找到不同 Babelfish 版本支持的 T-SQL 内置函数。

T-SQL 内置函数	5.5	5.4	4.4	4.4	4.4	4.4	4.4	4.3	3.3	3.3	3.3	3.3	3.3	3.3	3.3	3.3	2.2	2.2	2.2	2.2	2.2	2.2	2.2	2.2	2.2	2.2	2.2	2.2	2.0	1.0
APP_NAME	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否
ATN2		是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否

| T-SQL 内置函数 | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2

| T-SQL 内置函数 | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2

| T-SQL 内置函数 | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2

T-SQL 内置函数	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
fn_listextendedproperty	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	否
FOR JSON	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	否
FULLTEXTSERVICEPROPERTY	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	是
HAS_DBACCESS	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	是
HAS_PERMS_BY_NAME	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	是
HOST_NAME	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	否
HOST_ID	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	否
IDENTITY	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	否

| T-SQL 内置函数 | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2

| T-SQL 内置函数 | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2

| T-SQL 内置函数 | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2

| T-SQL 内置函数 | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2

| T-SQL 内置函数 | 5.5.4.4.4.4.4.4.4.3.3.3.3.3.3.3.3.3.2.2.2

T-SQL INFORMATION_SCHEMA 目录	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
COLUMN_DOMAIN_USAGE	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	否
COLUMNS	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	是
CONSTRAINT_COLUMN_USAGE	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	是
DOMAINS	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	是
KEY_COLUMN_USAGE	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	否
ROUTINES	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	否
TABLES	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	是

T-SQL INFORMATION_SCHEMA 目录	5.5	5.4	4.4	4.3	3.3	3.2	2.2	2.1.0
TABLE_CONSTRAINTS	支持	支持	支持	支持	支持	支持	支持	是
VIEWS	支持	支持	支持	支持	支持	支持	支持	是

T-SQL 系统定义的 @@ 变量

在下表中，可以找到不同 Babelfish 版本支持的 T-SQL 系统定义的 @@ 变量。

T-SQL 系统定义的 @@ 变量	5.5	5.4	4.4	4.3	3.3	3.2	2.2	2.1.0
@@CURSOR_ROWS	支持	支持	支持	支持	支持	支持	支持	是
@@DATEFIRST	支持	支持	支持	支持	支持	支持	支持	是
@@DBTS	支持	支持	支持	支持	支持	支持	支持	是
@@ERROR	支持	支持	支持	支持	支持	支持	支持	是

T-SQL 系统定义的 @@ 变量	5.5	5.4	4.4	4.3	3.3	3.2	2.2	2.1.0
@@ERROR=213	支持	支持	支持	支持	支持	支持	支持	是
@@FETCH_STATUS	支持	支持	支持	支持	支持	支持	支持	是
@@IDENTITY	支持	支持	支持	支持	支持	支持	支持	是
@@LANGUAGE	支持	支持	支持	支持	支持	支持	支持	是
@@LOCK_TIMEOUT	支持	支持	支持	支持	支持	支持	支持	是
@@MAX_CONNECTIONS	支持	支持	支持	支持	支持	支持	支持	是
@@MAX_PRECISION	支持	支持	支持	支持	支持	支持	支持	是

| T-SQL 系统定义的 @@ 变量 | 5 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2

T-SQL 系统定义的 @@ 变量	5.5	5.4	4.4	4.3	3.3	2.2	1.0
@@TRANCOUNT	支持	支持	支持	支持	支持	支持	是
@ @VERSION（请注意 Babelfish 中的 T-SQL 差异中描述的格式差异）。	支持	支持	支持	支持	支持	支持	是

T-SQL 系统存储过程

在下表中，可以找到不同 Babelfish 版本支持的 T-SQL 系统存储过程。

T-SQL 系统存储过程	5.5	5.4	4.4	4.3	3.3	2.2	1.0
sp_addextendedproperty	支持	支持	支持	支持	支持	不支持	否
sp_addlinkedserver	支持	支持	支持	支持	支持	不支持	否
sp_addlinkedsrvlogin	支持	支持	支持	支持	支持	不支持	否
sp_addrole	支持	支持	支持	支持	支持	不支持	否
sp_addrolemember							

T-SQL 系统存储过程	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	1.0
	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否
sp_babelfish_volatility	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否
sp_column_privileges	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
sp_columns	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
sp_columns_100	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
sp_columns_managed	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
sp_cursor	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
sp_cursor_list	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是

T-SQL 系统存储过程	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	1.0
sp_cursorclose	支持	是
sp_cursorexecute	支持	是
sp_cursorfetch	支持	是
sp_cursoropen	支持	是
sp_cursoroption	支持	是
sp_cursorprepare	支持	是
sp_cursorprepexec	支持	是
sp_cursorunprepare	支持	是

T-SQL 系统存储过程	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
sp_databases	支持	是
sp_datatype_info	支持	是
sp_datatype_info_100	支持	是
sp_describe_cursor	支持	是
sp_describe_first_result_set	支持	是
sp_describe_undeclared_parameters	支持	是
sp_dropextendedproperty	支持	否
sp_droplinkedsrvlogin	支持	否

T-SQL 系统存储过程	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
sp_droprole	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否
sp_droprolemember	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否
sp_dropserver	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否
sp_enum_oledb_providers	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否	是	是	是	是	是	是	是	否	否	否	否	否	否
sp_execute	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
sp_execute_postgresql(CREATE, ALTER, DROP)	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否	是	是	是	是	是	是	是	否	否	否	否	否	否
sp_executesql	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
sp_fkeys	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是

T-SQL 系统存储过程	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
sp_getapplock	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
sp_helpdb	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
sp_helpdbfixedrole	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否
sp_helplinkedsrvlogin	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否
sp_helprole	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
sp_helprolemember	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
sp_helpsrvrolemember	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否
sp_helpuser	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是

T-SQL 系统存储过程	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
sp_linkedservers	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否
sp_oledb_ro_usrname	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
sp_pkeys	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
sp_prefix	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否
sp_prepare	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
sp_procedure_params_100_managed	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否
sp_releaseapplock	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
sp_rename	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否

T-SQL 系统存储过程	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	1.0
sp_renamedb	支持	是	是	是	是	是	是	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否
sp_rename（适用于 object_type 'DATABASE'）	支持	是	是	是	是	是	是	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否	否
sp_reset_connection	支持	是	是	是	是	否	否	否	否	是	是	是	否	否	否	否	否	否	否	是	是	是	否	否	否	否	否	否	否	否	否
sp_serverOption（connect_timeout 选项）	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否	否	否	否	否	是	是	是	是	是	是	否	否	否	否	否
sp_set_session_context	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否	否
sp_special_columns	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
sp_sproc_columns	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
sp_sproc_columns_100	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是

T-SQL 系统存储过程	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
sp_statistics	支持	是
sp_statistics_100	支持	是
sp_stored_procedures	支持	是
sp_table_privileges	支持	是
sp_tablecollations_100	支持	是
sp_tables	支持	是
sp_testlinkedserver	支持	否
sp_unprepare	支持	是

| T-SQL 系统存储过程 | 5.5.4.4.4.4.4.4.4.3.3.3.3.3.3.3.3.3.2.2.2

| CONNECTIONPROPERTY 系统函数支持的 T-SQL 属性 | 5.5.4.4.4.4.4.4.4.3.3.3.3.3.3.3.3.3.2.2.2

OBJECTPROPERTY 系统函数支持的 T-SQL 属性	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
	支持																																
IsScalarFunction	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
IsTableFunction	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是

SERVERPROPERTY 函数支持的 T-SQL 属性

在下表中，可以找到 SERVERPROPERTY 函数支持的 T-SQL 属性（由不同的 Babelfish 版本支持）。

SERVERPROPERTY 函数支持的 T-SQL 属性	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
Babelfish	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
排序规则	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
CollationID		是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是

SERVERPROPERTY 函数支持的 T-SQL 属性	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
	支持																																
版本	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
EditionID	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
EngineEdition	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
InstanceName	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否
IsAdvancedAnalyticsInstalled	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
IsBigDataCluster	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
IsFullTextInstalled	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是

SERVERPROPERTY 函数支持的 T-SQL 属性	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
IsIntegratedSecurityOnly	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	是
IsLocalDB	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	是
IsPolyBaseInstalled	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	是
IsSingleUser	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	是
IsXTPSupported	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	是
Japanese_CI_AI	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	是
Japanese_CI_AS	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持	是

SERVERPROPERTY 函数支持的 T-SQL 属性	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	2012.0
Japanese_CS_AS	支持	是																															
LicenseType	支持	是																															
MachineName	支持	否																															
ProductLevel	支持	否																															
ProductMajorVersion	支持	是																															
ProductMinorVersion	支持	是																															
ProductUpdateLevel	支持	否																															

SERVERPROPERTY 函数支持的 T-SQL 属性	5.5.4	4.4.4	4.4.3	3.3.3	3.3.2	2.2.2	2.2.1	2.0.1	1.0
ProductVersion	支持	支持	支持	支持	支持	支持	支持	支持	是
ServerName	支持	支持	支持	支持	支持	支持	支持	支持	是

Babelfish 支持的 SQL Server 视图

在下表中，可以找到不同 Babelfish 版本支持的 SQL Server 视图。

Babelfish 支持的 SQL Server 视图	5.5.4	4.4.4	4.4.3	3.3.3	3.3.2	2.2.2	2.2.1	2.0.1	1.0
information_schema.key_column_usage	支持	支持	支持	支持	支持	支持	支持	支持	否
information_schema.routines	支持	支持	支持	支持	支持	支持	支持	支持	否
information_schema.schemata	支持	支持	支持	支持	支持	支持	支持	支持	否
information_schema.sequences		支持	支持	支持	支持	支持	支持	支持	否

Babelfish 支持的 SQL Server 视图	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
	支持																																
sys.all_columns	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
sys.all_objects	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
sys.all_parameters	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	否	否
sys.all_sql_modules	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
sys.all_views	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
sys.columns	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是
sys.configurations	支持	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是	是

| Babelfish 支持的 SQL Server 视图 | 5.5 | 5.4 | 4.4 | 4.4 | 4.4 | 4.4 | 4.3 | 3.3 | 3.3 | 3.3 | 3.3 | 3.3 | 3.3 | 3.3 | 3.3 | 2.2 | 2.2 | 2.2

Babelfish 支持的 SQL Server 视图	5.5	5.4	4.4	4.3	3.3	3.2	2.2	2.1	1.0
sys.dm_exec_connections	支持	支持	支持	支持	支持	支持	支持	支持	是
sys.dm_exec_sessions	支持	支持	支持	支持	支持	支持	支持	支持	是
sys.dm_hadr_database_replica_states	支持	支持	支持	支持	支持	支持	支持	支持	是
sys.dm_os_host_info	支持	支持	支持	支持	支持	支持	支持	支持	是
sys.dm_os_sys_info	支持	支持	支持	支持	支持	支持	支持	支持	否
sys.endpoints	支持	支持	支持	支持	支持	支持	支持	支持	是
sys.extended_properties	支持	支持	支持	支持	支持	支持	支持	支持	否

Babelfish 支持的 SQL Server 视图	5.5	5.4	4.4	4.4	4.4	4.4	4.4	4.4	3.3	3.3	3.3	3.3	3.3	3.3	3.3	3.3	3.3	2.2	2.2	2.2	2.2	2.2	2.2	2.2	2.2	2.2	2.2	2.2	2.2	1.0
sys.indexes	支持	是
sys.partitions	支持	否
sys.partition_functions	支持	否
sys.partition_parameters	支持	否
sys.partition_range_values	支持	否
sys.partition_schemes	支持	否
sys.schemas	支持	是

Babelfish 支持的 SQL Server 视图	5.5	5.4	4.4	4.3	3.3	3.2	2.2	2.1.0
sys.server_principals	支持	支持	支持	支持	支持	支持	支持	是
sys.server_role_members	支持	支持	支持	支持	支持	支持	支持	否
sys.sql_modules	支持	支持	支持	支持	支持	支持	支持	是
sys.sysconfigures	支持	支持	支持	支持	支持	支持	支持	是
sys.syscurconfigs	支持	支持	支持	支持	支持	支持	支持	是
sys.syslogins	支持	支持	支持	支持	支持	支持	支持	否
sys.sysprocesses	支持	支持	支持	支持	支持	支持	支持	是

Babelfish 支持的 SQL Server 视图	5.5	5.4	4.4	4.3	3.3	3.2	2.2	2.1.0
sys.sysusers	支持	支持	支持	支持	支持	支持	支持	否
sys.table_types	支持	支持	支持	支持	支持	支持	支持	是
sys.tables	支持	支持	支持	支持	支持	支持	支持	是
sys.types	支持	支持	支持	支持	支持	支持	支持	否
sys.xml_schema_collections	支持	支持	支持	支持	支持	支持	支持	是
syslanguages	支持	支持	支持	支持	支持	支持	支持	是
sysobjects.crdate	支持	支持	支持	支持	支持	支持	支持	否

Babelfish 中不支持的功能

在以下表和列表中，可以找到 Babelfish 当前不支持的功能。Aurora PostgreSQL 版本中包含对 Babelfish 的更新。有关更多信息，请参阅 Aurora PostgreSQL 版本注释。

主题

- 当前不支持的功能
- 不支持的设置
- 不支持的命令
- 不支持的列名称或属性
- 不支持的数据类型
- 不支持的对象类型
- 不支持的函数
- 不支持的语法

当前不支持的功能

在表格中，您可以找到有关当前不支持的某些功能的信息。

功能或语法	描述
程序集模块和 SQL 公共语言运行时 (CLR) 例程	不支持与组装模块和 CLR 例程相关的功能。
列属性	不支持 ROWGUIDCOL、SPARSE、FILESTREAM 和 MASKED。
包含的数据库	不支持在数据库级别而不是在服务器级别验证了登录名的包含数据库。
跨数据库 DDL	尚不支持执行跨多个数据库引用或操作对象的 DDL 语句。
光标（可更新）	不支持可更新的光标。
光标（全局）	不支持 GLOBAL 光标。

功能或语法	描述
光标（获取行为）	不支持以下光标获取行为：FETCH PRIOR、FIRST、LAST、ABSOLUTE 和 RELATIVE
光标类型的输出参数	输出参数不支持光标类型的变量和参数（引发错误）。
光标选项	SCROLL、KEYSET、DYNAMIC、FAST_FORWARD、SCROLL_LOCKS、OPTIMISTIC、TYPE_WARNING 和 FOR UPDATE
数据加密	不支持数据加密。
数据层应用程序（DAC）	不支持使用 DAC 程序包（.dacpac）或 DAC 备份（.bacpac）文件进行数据层应用程序（DAC）导入或导出操作。
DBCC 命令	不支持 Microsoft SQL Server 数据库控制台命令 (DBCC)。在 Babelfish 3.4.0 及更高版本中支持 DBCC CHECKIDENT。
DROP IF EXISTS	USER 和 SCHEMA 对象不支持此语法。对象 TABLE、VIEW、PROCEDURE、FUNCTION 和 DATABASE 都支持此语法。
加密	内置函数和语句不支持加密。
ENCRYPT_CLIENT_CERT 连接	不支持客户端证书连接。
EXECUTE AS 语句	不支持此语句。
EXECUTE AS SELF 子句	函数、过程或触发器不支持此子句。
EXECUTE AS USER 子句	函数、过程或触发器不支持此子句。
引用数据库名称的外键约束	不支持引用数据库名称的外键约束。
FORMAT	不支持用户定义的类型。
具有大于 100 个参数的函数声明	包含 100 个以上参数的函数声明不受支持。

功能或语法	描述
包含 DEFAULT 作为参数值的函数调用	DEFAULT 不是函数调用支持的参数值。Babelfish 3.4.0 及更高版本支持将 DEFAULT 作为函数调用的参数值。
外部定义的函数	不支持 SQL CLR 函数等外部函数。
全局临时表（名称以 ## 开头的表）	不支持全局临时表。
图表功能	不支持所有 SQL 图表功能。
一般扩展存储过程	某些系统存储过程可为各种维护活动提供从 SQL Server 实例到外部程序的接口，但此处不支持此类系统存储过程。这包括 `xp_cmdshell` 和其他系统存储过程。有关更多信息，请参阅 General Extended stored procedures。
具有多个前导 @ 字符的标识符（变量或参数）	不支持以多个前导 @ 开头的标识符。
包含 @ 或]] 字符的标识符、表或列名	不支持包含 @ 符号或方括号的表或列名。
内联索引	不支持内联索引。
调用名称在变量中的过程	不支持使用变量作为过程名称。
具体化视图	不支持具体化视图。
NOT FOR REPLICATION 子句	接受并忽略此语法。
ODBC 转义函数	不支持 ODBC 转义函数。
包含 DEFAULT 作为参数值的过程调用	DEFAULT 不是支持的参数值。Babelfish 3.4.0 及更高版本支持将 DEFAULT 作为函数调用的参数值。
包含 100 个以上参数的过程声明	不支持包含 100 个以上参数的声明。

功能或语法	描述
外部定义的过程	不支持 SQL CLR 过程等外部定义的过程。
过程版本控制	不支持过程版本控制。
过程 WITH RECOMPILE	不支持 WITH RECOMPILE（与 DECLARE 和 EXECUTE 语句结合使用时）。
远程对象引用	不支持针对 Babelfish 链接服务器执行存储过程。由四部分组成的对象名称仅用于读取，不适用于修改远程表。UPDATE 可以在 FROM 子句中引用远程表，而无需对其进行修改。有关更多信息，请参阅 Babelfish 支持链接服务器。
行级别安全性	不支持具有 CREATE SECURITY POLICY 和内联表值函数的行级安全性。
服务代理功能	不支持服务代理功能。
SESSIONPROPERTY	不支持的属性：ANSI_NULLS、ANSI_PADDING、ANSI_WARNINGS、ARITHABORT、CONCAT_NULL_YIELDS_NULL 和 NUMERIC_ROUNDABORT
SET LANGUAGE	此语句不支持 `english` 或 `us_english` 之外的任何值。
SP_CONFIGURE	不支持此系统存储过程。
SQL 关键字 SPARSE	接受并忽略关键字 SPARSE。
表值构造函数语句（FROM 子句）	不支持的语法适用于使用 FROM 子句构造的派生表。
临时表	不支持临时表。
临时过程不会自动删除	不支持此函数。
外部定义的触发器	这些触发器不受支持，包括 SQL 公共语言运行时 (CLR)。

功能或语法	描述
无 SCHEMABINDING 子句	不支持在不使用 SCHEMABINDING 的情况下创建视图，但创建视图时似乎指定了 WITH SCHEMABINTING。在创建函数、过程和触发器时使用 SCHEMABINDING 会被无提示忽略。

不支持的设置

不支持以下设置：

- SET ANSI_NULL_DFLT_OFF ON
- SET ANSI_NULL_DFLT_ON OFF
- SET ANSI_PADDING OFF
- SET ANSI_WARNINGS OFF
- SET ARITHABORT OFF
- SET ARITHIGNORE ON
- SET CURSOR_CLOSE_ON_COMMIT ON
- SET NUMERIC_ROUNDABORT ON
- SET PARSEONLY ON（命令未按预期工作）
- SET FMTONLY ON（命令未按预期工作。它仅抑制 SELECT 语句的执行，而不抑制其他语句的执行。）

不支持的命令

不支持以下命令的某些功能：

- ADD SIGNATURE
- ALTER DATABASE、ALTER DATABASE SET
- BACKUP/RESTORE DATABASE/LOG
- BACPAC 和 DACPAC FILES RESTORE
- CREATE、ALTER、DROP AUTHORIZATION。数据库对象支持 ALTER AUTHORIZATION。
- CREATE、ALTER、DROP AVAILABILITY GROUP
- CREATE、ALTER、DROP BROKER PRIORITY
- CREATE、ALTER、DROP COLUMN ENCRYPTION KEY

- CREATE、ALTER、DROP DATABASE ENCRYPTION KEY
- CREATE、ALTER、DROP、BACKUP CERTIFICATE
- CREATE AGGREGATE
- CREATE CONTRACT
- CHECKPOINT

不支持的列名称或属性

不支持以下列名称：

- $IDENTITY
- $ROWGUID
- IDENTITYCOL

不支持的数据类型

不支持以下数据类型：

- HIERARCHYID

不支持的对象类型

不支持以下对象类型：

- COLUMN MASTER KEY
- CREATE、ALTER EXTERNAL DATA SOURCE
- CREATE、ALTER、DROP DATABASE AUDIT SPECIFICATION
- CREATE、ALTER、DROP EXTERNAL LIBRARY
- CREATE、ALTER、DROP SERVER AUDIT
- CREATE、ALTER、DROP SERVER AUDIT SPECIFICATION
- CREATE、ALTER、DROP、OPEN/CLOSE SYMMETRIC KEY
- CREATE、DROP DEFAULT
- CREDENTIAL
- CRYPTOGRAPHIC PROVIDER
- DIAGNOSTIC SESSION

- 索引视图
- SERVICE MASTER KEY
- SYNONYM

不支持的函数

不支持以下内置函数：

聚合函数

- APPROX_COUNT_DISTINCT
- CHECKSUM_AGG
- GROUPING_ID
- 使用 WITHIN GROUP 子句的 STRING_AGG

加密函数

- CERTENCODED 函数
- CERTID 函数
- CERTPROPERTY 函数

元数据函数

- COLUMNPROPERTY
- TYPEPROPERTY
- SERVERPROPERTY 函数 — 不支持以下属性：
  - BuildClrVersion
  - ComparisonStyle
  - ComputerNamePhysicalNetBIOS
  - HadrManagerStatus
  - InstanceDefaultDataPath
  - InstanceDefaultLogPath
  - IsClustered
  - IsHadrEnabled

- LCID
- NumLicenses
- ProcessID
- ProductBuild
- ProductBuildType
- ProductUpdateReference
- ResourceLastUpdateDateTime
- ResourceVersion
- ServerName
- SqlCharSet
- SqlCharSetName
- SqlSortOrder
- SqlSortOrderName
- FilestreamShareName
- FilestreamConfiguredLevel
- FilestreamEffectiveLevel

安全函数

- CERTPRIVATEKEY
- LOGINPROPERTY

语句、运算符、其他函数

- EVENTDATA 函数
- GET_TRANSMISSION_STATUS
- OPENXML

不支持的语法

不支持以下语法：

- ALTER DATABASE

- ALTER DATABASE SCOPED CONFIGURATION
- ALTER DATABASE SCOPED CREDENTIAL
- ALTER DATABASE SET HADR
- ALTER INDEX
- ALTER PARTITION FUNCTION
- ALTER PARTITION SCHEME
- ALTER SCHEMA
- ALTER SERVER CONFIGURATION
- ALTER SERVICE、BACKUP/RESTORE SERVICE MASTER KEY 子句
- BEGIN CONVERSATION TIMER
- BEGIN DISTRIBUTED TRANSACTION
- BEGIN DIALOG CONVERSATION
- BULK INSERT
- CREATE COLUMNSTORE INDEX
- CREATE EXTERNAL FILE FORMAT
- CREATE EXTERNAL TABLE
- CREATE、ALTER、DROP APPLICATION ROLE
- CREATE、ALTER、DROP ASSEMBLY
- CREATE、ALTER、DROP ASYMMETRIC KEY
- CREATE, ALTER, DROP CREDENTIAL
- CREATE, ALTER, DROP CRYPTOGRAPHIC PROVIDER
- CREATE, ALTER, DROP ENDPOINT
- CREATE、ALTER、DROP EVENT SESSION
- CREATE, ALTER, DROP EXTERNAL LANGUAGE
- CREATE、ALTER、DROP EXTERNAL RESOURCE POOL
- CREATE、ALTER、DROP FULLTEXT CATALOG
- CREATE、ALTER、DROP FULLTEXT INDEX
- CREATE、ALTER、DROP FULLTEXT STOPLIST
- CREATE, ALTER, DROP MESSAGE TYPE
- CREATE、ALTER、DROP、OPEN/CLOSE、BACKUP/RESTORE MASTER KEY

- CREATE、ALTER、DROP QUEUE
- CREATE、ALTER、DROP RESOURCE GOVERNOR
- CREATE, ALTER, DROP RESOURCE POOL
- CREATE、ALTER、DROP ROUTE
- CREATE, ALTER, DROP SEARCH PROPERTY LIST
- CREATE, ALTER, DROP SECURITY POLICY
- CREATE, ALTER, DROP SELECTIVE XML INDEX clause
- CREATE、ALTER、DROP SERVICE
- CREATE, ALTER, DROP SPATIAL INDEX
- CREATE, ALTER, DROP TYPE
- CREATE, ALTER, DROP XML INDEX
- CREATE, ALTER, DROP XML SCHEMA COLLECTION
- CREATE/DROP RULE
- CREATE, DROP WORKLOAD CLASSIFIER
- CREATE、ALTER、DROP WORKLOAD GROUP
- ALTER TRIGGER
- CREATE TABLE... GRANT 子句
- CREATE TABLE... IDENTITY 子句
- CREATE USER — 不支持此语法。PostgreSQL 语句 CREATE USER 不会创建与 SQL Server CREATE USER 语法相当的用户。有关更多信息，请参阅 Babelfish 中的 T-SQL 差异。
- DENY
- END, MOVE CONVERSATION
- EXECUTE with AS LOGIN or AT option
- GET CONVERSATION GROUP
- GROUP BY ALL clause
- GROUP BY CUBE clause
- GROUP BY ROLLUP clause
- INSERT... DEFAULT VALUES
- MERGE
- READTEXT
- REVERT

- SELECT TOP x PERCENT WHERE x <> 100
- SELECT TOP... WITH TIES
- SELECT... FOR BROWSE
- SELECT... FOR XML AUTO
- SELECT... FOR XML EXPLICIT
- SELECT... FOR XML PATH
- SEND
- SET DATEFORMAT
- SET DEADLOCK_PRIORITY
- SET FMTONLY
- SET FORCEPLAN
- SET NUMERIC_ROUNDABORT ON
- SET OFFSETS
- SET REMOTE_PROC_TRANSACTIONS
- SET SHOWPLAN_TEXT
- SET SHOWPLAN_XML
- SET STATISTICS
- SET STATISTICS PROFILE
- SET STATISTICS TIME
- SET STATISTICS XML
- SHUTDOWN statement
- UPDATE STATISTICS
- UPDATETEXT
- Using EXECUTE to call a SQL function
- VIEW... CHECK OPTION clause
- VIEW... VIEW_METADATA clause
- WAITFOR DELAY
- WAITFOR TIME
- WAITFOR, RECEIVE
- WITH XMLNAMESPACES construct

- WRITETEXT
- XPATH expressions

使用适用于 Aurora PostgreSQL 的 Babelfish 过程

概述

您可以对运行适用于 Aurora PostgreSQL 的 Babelfish 的 Amazon RDS 数据库实例使用以下过程，以提高查询性能：

- sp_babelfish_volatility
- sp_execute_postgresql

sp_babelfish_volatility

PostgreSQL 函数不稳定性有助于优化器更好地执行查询，当在某些子句的某些部分中使用时，会对查询性能产生重大影响。

语法

```
sp_babelfish_volatility ‘function_name’, ‘volatility’
```

参数

function_name（可选）

您可以将此参数的值指定为由两部分组成的名称 `schema_name.function_name`，也可以仅指定 `function_name`。如果您仅指定 `function_name`，则架构名称是当前用户的原定设置架构。

volatility（可选）

有效的 PostgreSQL 不稳定性值为 `stable`、`volatile` 或 `immutable`。有关更多信息，请参阅 https://www.postgresql.org/docs/current/xfunc-volatility.html

> **Note**
>
> 当使用具有多个定义的 `function_name` 调用 `sp_babelfish_volatility` 时，将引发错误。

结果集

如果未提及参数，则结果集将显示在以下各列之下：schemaname、functionname、volatility。

使用说明

PostgreSQL 函数不稳定性有助于优化器更好地执行查询，当在某些子句的某些部分中使用时，会对查询性能产生重大影响。

示例

以下示例显示了如何创建简单的函数，稍后将说明如何使用不同的方法在这些函数上使用 sp_babelfish_volatility。

```
1> create function f1() returns int as begin return 0 end
2> go
```

```
1> create schema test_schema
2> go
```

```
1> create function test_schema.f1() returns int as begin return 0 end
2> go
```

以下示例显示了函数的不稳定性：

```
1> exec sp_babelfish_volatility
2> go

schemaname  functionname volatility
----------- ------------ ----------
dbo         f1           volatile
test_schema f1           volatile
```

以下示例显示了如何更改函数的不稳定性：

```
1> exec sp_babelfish_volatility 'f1','stable'
2> go
1> exec sp_babelfish_volatility 'test_schema.f1','immutable'
```

```
2> go
```

当您仅指定 function_name 时，它会显示该函数的架构名称、函数名称和不稳定性。以下示例显示了更改值后函数的不稳定性：

```
1> exec sp_babelfish_volatility 'test_schema.f1'
2> go

schemaname  functionname volatility
----------- ------------ ----------
test_schema f1           immutable
```

```
1> exec sp_babelfish_volatility 'f1'
2> go

schemaname  functionname volatility
----------- ------------ ----------
dbo         f1           stable
```

当您不指定任何参数时，它会显示当前数据库中存在的函数列表（架构名称、函数名称、函数的不稳定性）：

```
1> exec sp_babelfish_volatility
2> go

schemaname  functionname volatility
----------- ------------ ----------
dbo         f1           stable
test_schema f1           immutable
```

sp_execute_postgresql

您可以从 T-SQL 终端节点执行 PostgreSQL 语句。这简化了您的应用程序，因为您无需退出 T-SQL 端口即可执行这些语句。

语法

```
sp_execute_postgresql [ @stmt = ] statement
```

参数

[@stmt] statement

参数的数据类型为 varchar。此参数接受 PG 方言语句。

> **Note**
>
> 您只能传递一个 PG 方言语句作为参数，否则会引发以下错误。

```
1>exec sp_execute_postgresql 'create extension pg_stat_statements; drop extension
 pg_stat_statements'
2>go
```

```
Msg 33557097, Level 16, State 1, Server BABELFISH, Line 1
expected 1 statement but got 2 statements after parsing
```

使用说明

CREATE EXTENSION

创建新的扩展并将其加载到当前数据库中。

```
1>EXEC sp_execute_postgresql 'create extension [ IF NOT EXISTS ] <extension name>
 [ WITH ] [SCHEMA schema_name] [VERSION version]';
2>go
```

以下示例说明了如何创建扩展：

```
1>EXEC sp_execute_postgresql 'create extension pg_stat_statements with schema sys
 version "1.10"';
2>go
```

使用以下命令访问扩展对象：

```
1>select * from pg_stat_statements;
2>go
```

> **Note**
>
> 如果在创建扩展期间未明确提供模式名称，则默认情况下，扩展将安装在公共模式中。您必须提供模式限定符才能访问扩展对象，如下所述：

```
1>select * from [public].pg_stat_statements;
2>go
```

支持的扩展

Aurora PostgreSQL 提供的以下扩展适用于 Babelfish。

- `pg_stat_statements`
- `tds_fdw`
- `fuzzystrmatch`

限制

- 用户需要在 T-SQL 上具有 sysadmin 角色，在 postgres 上具有 rds_superuser 角色才能安装扩展。
- 扩展不能安装在用户创建的模式中，也不能安装在主数据库、tempdb 和 msdb 数据库的 dbo 和访客模式中。
- 不支持 CASCADE 选项。

ALTER EXTENSION

您可以使用 ALTER 扩展升级到新的扩展版本。

```
1>EXEC sp_execute_postgresql 'alter extension <extension name> UPDATE TO
 <new_version>';
```

```
2>go
```

限制

- 您只能使用 ALTER Extension 语句升级扩展的版本。不支持其它操作。

DROP EXTENSION

删除指定的扩展。您也可以使用 `if exists` 或 `restrict` 选项删除扩展。

```
1>EXEC sp_execute_postgresql 'drop extension <extension name>';
2>go
```

限制

- 不支持 CASCADE 选项。

Amazon Aurora PostgreSQL 的性能和扩展

下文将介绍如何管理 Amazon Aurora PostgreSQL 数据库集群的性能和扩缩。还包括有关其他维护任务的信息。

主题

- 扩展 Aurora PostgreSQL 数据库实例
- 至 Aurora PostgreSQL 数据库实例的最大连接数
- Aurora PostgreSQL 的临时存储限制
- Aurora PostgreSQL 的大页
- 使用错误注入查询测试 Amazon Aurora PostgreSQL
- 显示 Aurora PostgreSQL 数据库集群的卷状态
- 指定 stats_temp_directory 的 RAM 磁盘
- 使用 PostgreSQL 管理临时文件

扩展 Aurora PostgreSQL 数据库实例

您可通过两种方式扩展 Aurora PostgreSQL 数据库实例，即实例扩展和读取扩展。有关读取扩展的更多信息，请参阅读取扩展。

您可以修改数据库集群中的每个数据库实例的数据库实例类，以扩展 Aurora PostgreSQL 数据库集群。Aurora PostgreSQL 支持多种针对 Aurora 优化的数据库实例类。不要将 db.t2 或 db.t3 实例类用于大小超过 40 TB 的较大 Aurora 集群。

> **Note**
>
> 建议仅将 T 数据库实例类用于开发和测试服务器，或其他非生产服务器。有关 T 实例类的更多详细信息，请参阅数据库实例类类型。

扩缩不会即时完成。可能需要 15 分钟或更长时间才能完成对其他数据库实例类的更改。如果使用此方法修改数据库实例类，则应在下一个计划的维护时段内（而不是立即）应用更改，从而避免影响用户。

作为直接修改数据库实例类的替代方法，您可以使用 Amazon Aurora 的高可用性功能显著减少停机时间。首先，将 Aurora 副本添加到集群中。创建副本时，选择要用于集群的数据库实例类大小。Aurora 副本与集群同步后，您可以失效转移至新添加的副本。要了解更多信息，请参阅 Aurora 副本 和 Amazon Aurora PostgreSQL 的快速故障转移。

有关 Aurora PostgreSQL 支持的数据库实例类的详细规格，请参阅数据库实例类支持的数据库引擎。

至 Aurora PostgreSQL 数据库实例的最大连接数

Aurora PostgreSQL 数据库集群根据数据库实例类及其可用内存分配资源。与数据库集群的每个连接都会占用这些资源（如内存和 CPU）的增量。每个连接占用的内存取决于查询类型、计数以及是否使用临时表。即使是空闲连接也会占用内存和 CPU。这是因为当查询在连接上运行时，系统将为每个查询分配更多内存，而且即使处理停止，内存也不会完全释放。因此，建议您确保应用程序不会占用空闲连接：每个空闲连接都会浪费资源并对性能产生负面影响。有关更多信息，请参阅空闲 PostgreSQL 连接占用的资源。

Aurora PostgreSQL 数据库实例允许的最大连接数量由该数据库实例的参数组中指定的 `max_connections` 参数值确定。`max_connections` 参数的理想设置是既能支持应用程序所需的所有客户端连接，又不会有太多未使用的连接，外加至少 3 个支持 AWS 自动化的连接。修改 `max_connections` 参数设置之前，建议您考虑以下内容：

- 如果 `max_connections` 值太低，当客户端尝试连接时，Aurora PostgreSQL 数据库实例可能没有足够的可用连接。如果发生这种情况，尝试使用 `psql` 进行连接会引发如下错误消息：

```
psql: FATAL: remaining connection slots are reserved for non-replication superuser
 connections
```

- 如果 max_connections 值超过了实际需要的连接数，未使用的连接可能会导致性能降低。

max_connections 的默认值源自以下 Aurora PostgreSQL LEAST 函数：

LEAST({DBInstanceClassMemory/9531392},5000).

如果要更改 max_connections 的值，您需要创建自定义数据库集群参数组并在其中更改其值。将自定义数据库参数组应用到集群之后，请务必重启主实例，这样新值才能生效。有关更多信息，请参阅Amazon Aurora PostgreSQL 参数和在 Amazon Aurora 中创建数据库集群参数组。

> **Tip**
>
> 如果您的应用程序经常打开和关闭连接，或使大量长期连接保持打开，建议您使用 Amazon RDS 代理。RDS 代理是一种完全托管的高可用性数据库代理，它使用连接池安全有效地共享数据库连接。要了解有关 RDS 代理的更多信息，请参阅 适用于 Aurora 的 Amazon RDS 代理。

有关 Aurora Serverless v2 实例如何处理此参数的详细信息，请参阅 Aurora Serverless v2 的最大连接数。

Aurora PostgreSQL 的临时存储限制

Aurora PostgreSQL 将表和索引储存在 Aurora 存储子系统中。Aurora PostgreSQL 对于非持久性临时文件使用单独的临时存储。这包括用于在查询处理过程对大型数据集进行排序或者用于索引构建操作等用途的文件。有关更多信息，请参阅文章如何解决 Aurora PostgreSQL 兼容实例中的本地存储问题？

这些本地存储卷由 Amazon Elastic Block Store 提供支持，并可以通过使用更大的数据库实例类来进行扩展。有关存储的更多信息，请参阅 Amazon Aurora 存储。您还可以使用启用 NVMe 的实例类型和启用 Aurora 优化型读取功能的临时对象，来增加临时对象的本地存储。有关更多信息，请参阅 使用 Aurora 优化型读取功能提高 Aurora PostgreSQL 的查询性能。

> **Note**
>
> 当扩展数据库实例（例如，从 db.r5.2xlarge 扩展到 db.r5.4xlarge）时，您可能会看到 storage-optimization 事件。

下表显示了每个 Aurora PostgreSQL 数据库实例类可用的最大临时存储空间。有关 Aurora 的数据库实例类支持的更多信息，请参阅 Amazon Aurora 数据库实例类。

数据库实例类	最大可用临时存储空间 (GiB)
db.x2g.16xlarge	1829
db.x2g.12xlarge	1606
db.x2g.8xlarge	1071
db.x2g.4xlarge	535
db.x2g.2xlarge	268
db.x2g.xlarge	134
db.x2g.large	67
db.r8g.48xlarge	3072
db.r8g.24xlarge	1536
db.r8g.16xlarge	998
db.r8g.12xlarge	749
db.r8g.8xlarge	499
db.r8g.4xlarge	250
db.r8g.2xlarge	125
db.r8g.xlarge	63
db.r8g.large	31
db.r7g.16xlarge	1008
db.r7g.12xlarge	756
db.r7g.8xlarge	504

数据库实例类	最大可用临时存储空间 (GiB)
db.r7g.4xlarge	252
db.r7g.2xlarge	126
db.r7g.xlarge	63
db.r7g.large	32
db.r7i.48xlarge	3072
db.r7i.24xlarge	1500
db.r7i.16xlarge	1008
db.r7i.12xlarge	748
db.r7i.8xlarge	504
db.r7i.4xlarge	249
db.r7i.2xlarge	124
db.r7i.xlarge	62
db.r7i.large	31
db.r6g.16xlarge	1008
db.r6g.12xlarge	756
db.r6g.8xlarge	504
db.r6g.4xlarge	252
db.r6g.2xlarge	126
db.r6g.xlarge	63
db.r6g.large	32

数据库实例类	最大可用临时存储空间 (GiB)
db.r6i.32xlarge	1829
db.r6i.24xlarge	1500
db.r6i.16xlarge	1008
db.r6i.12xlarge	748
db.r6i.8xlarge	504
db.r6i.4xlarge	249
db.r6i.2xlarge	124
db.r6i.xlarge	62
db.r6i.large	31
db.r5.24xlarge	1500
db.r5.16xlarge	1008
db.r5.12xlarge	748
db.r5.8xlarge	504
db.r5.4xlarge	249
db.r5.2xlarge	124
db.r5.xlarge	62
db.r5.large	31
db.r4.16xlarge	960
db.r4.8xlarge	480
db.r4.4xlarge	240

数据库实例类	最大可用临时存储空间 (GiB)
db.r4.2xlarge	120
db.r4.xlarge	60
db.r4.large	30
db.t4g.large	16.5
db.t4g.medium	8.13
db.t3.large	16
db.t3.medium	7.5

> **Note**
>
> 启用 NVMe 的实例类型最多可以将可用临时空间增加到 NVMe 的总大小。有关更多信息，请参阅 使用 Aurora 优化型读取功能提高 Aurora PostgreSQL 的查询性能。

您可以使用 `FreeLocalStorage` CloudWatch 指标监控数据库实例可用的临时存储，如Amazon Aurora 的 Amazon CloudWatch 指标中所述。（这不适用于 Aurora Serverless v2。）

对于某些工作负载，您可以通过为执行操作的进程分配更多内存来减少临时存储量。要增加操作可用的内存，请增加 work_mem 或 maintenance_work_mem PostgreSQL 参数的值。

Aurora PostgreSQL 的大页

大页是一项内存管理功能，可以减少数据库实例处理大量连续内存数据块（例如共享缓冲区使用的内存数据块）时的开销。Aurora PostgreSQL 的当前所有可用版本都支持这项 PostgreSQL 功能。

对于除 t3.medium、db.t3.large、db.t4g.medium、db.t4g.large 实例类之外的所有数据库实例类，默认情况下都会开启 `Huge_pages` 参数。您无法在 Aurora PostgreSQL 支持的实例类中更改 `huge_pages` 参数值或关闭此功能。

在不支持大页内存功能的 Aurora PostgreSQL 数据库实例上，特定进程内存使用量可能会增加，而不会相应地改变工作负载。

在服务器启动期间，系统会分配共享内存段，例如缓冲区缓存。当大内存页不可用时，系统不会向 postmaster 进程收取这些分配的费用。相反，它包括首先访问共享内存段中每个 4 KB 页面的进程中的内存。

> **Note**
>
> 活动连接会根据需要共享分配的内存，而无论如何跨进程跟踪共享内存使用量。

使用错误注入查询测试 Amazon Aurora PostgreSQL

可使用错误注入查询来测试 Aurora PostgreSQL 数据库集群的容错能力。错误注入查询作为 SQL 命令发布到 Amazon Aurora 实例。错误注入查询允许您使实例崩溃，以便您可以测试失效转移和恢复。您还可以模拟 Aurora 副本故障、磁盘故障和磁盘拥塞。所有可用的 Aurora PostgreSQL 版本都支持错误注入查询，如下所示。

- Aurora PostgreSQL 版本 12、13、14 及更高版本
- Aurora PostgreSQL 版本 11.7 及更高版本
- Aurora PostgreSQL 版本 10.11 及更高版本

主题

- 测试实例崩溃
- 测试 Aurora 副本故障
- 测试磁盘故障
- 测试磁盘拥塞

当故障注入查询指定崩溃时，它会强制 Aurora PostgreSQL 数据库实例崩溃。其他错误注入查询将导致模拟故障事件，但不会导致事件发生。提交错误注入查询时，还可指定故障事件模拟发生的时间长度。

可通过连接到 Aurora 副本的端点来将错误注入查询提交到 Aurora 副本实例之一。有关更多信息，请参阅“Amazon Aurora 端点连接”。

测试实例崩溃

您可使用错误注入查询函数 `aurora_inject_crash()` 强制使 Aurora PostgreSQL 实例发生崩溃。

对于该错误注入查询，不会进行故障转移。如果要测试故障转移，您可以在 RDS 控制台中为数据库集群选择 Failover（故障转移）实例操作，或者使用 failover-db-cluster AWS CLI 命令或 FailoverDBCluster RDS API 操作。

语法

```
SELECT aurora_inject_crash ('instance' | 'dispatcher' | 'node');
```

选项

该错误注入查询采用下列崩溃类型之一。崩溃类型不区分大小写：

'instance'

模拟 Amazon Aurora 实例的 PostgreSQL 兼容数据库崩溃。

'dispatcher'

模拟 Aurora 数据库集群的主实例上的调度程序崩溃。调度程序 将更新写入到 Amazon Aurora 数据库集群的集群卷中。

'node'

模拟 Amazon Aurora 实例的 PostgreSQL 兼容数据库和调度程序崩溃。

测试 Aurora 副本故障

可使用错误注入查询函数 `aurora_inject_replica_failure()` 来模拟 Aurora 副本的故障。

Aurora 副本故障将在指定的时间间隔内按指定的百分比阻止复制到 Aurora 副本或数据库集群中的所有 Aurora 副本。在该时间间隔过后，受影响的 Aurora 副本将自动与主实例同步。

语法

```
SELECT aurora_inject_replica_failure(
   percentage_of_failure,
   time_interval,
   'replica_name'
);
```

选项

该错误注入查询采用以下参数：

percentage_of_failure

在故障事件期间阻止的复制百分比。该值可为 0 到 100 之间的双数。如果指定 0，则不会阻止任何复制。如果指定 100，则将阻止所有复制。

time_interval

模拟 Aurora 副本故障的时间长度。间隔以秒为单位。例如，如果值为 20，则模拟会运行 20 秒。

> **Note**
>
> 在指定 Aurora 副本故障事件的时间间隔时，请小心谨慎。如果指定的时间间隔太长，并且您的写入器实例在故障事件期间写入大量数据，则您的 Aurora 数据库集群可能假定您的 Aurora 副本已发生崩溃并将替换它。

replica_name

要在其中注入故障模拟的 Aurora 副本。指定单个 Aurora 副本的名称可模拟单个 Aurora 副本故障的情况。指定空字符串可模拟数据库群集中所有 Aurora 副本故障的情况。

要确定副本名称，请参阅 `server_id` 函数中的 `aurora_replica_status()` 列。例如：

```
postgres=> SELECT server_id FROM aurora_replica_status();
```

测试磁盘故障

可使用错误注入查询函数 `aurora_inject_disk_failure()` 模拟 Aurora PostgreSQL 数据库集群的磁盘故障。

磁盘故障模拟期间，Aurora PostgreSQL 数据库集群会随机将磁盘区段标记为故障。在模拟期内，将阻止对这些区段的请求。

语法

```
SELECT aurora_inject_disk_failure(
   percentage_of_failure,
   index,
   is_disk,
   time_interval
);
```

选项

该错误注入查询采用以下参数：

percentage_of_failure

在故障事件期间标记为故障的磁盘百分比。该值可为 0 到 100 之间的双数。如果指定 0，则不会将任何磁盘标记为故障。如果指定 100，则整个磁盘将标记为故障。

index

模拟故障事件的特定逻辑数据块。如果超出了可用逻辑数据块或存储节点数据的范围，您会收到一条错误，告知您可指定的最大索引值。要避免此错误，请参阅 显示 Aurora PostgreSQL 数据库集群的卷状态。

is_disk

指示注入失败是针对逻辑数据块还是存储节点。指定 true 意味着注入失败针对逻辑数据块。指定 false 意味着注入失败针对存储节点。

time_interval

模拟磁盘故障的时间长度。间隔以秒为单位。例如，如果值为 20，则模拟会运行 20 秒。

测试磁盘拥塞

可使用错误注入查询函数 `aurora_inject_disk_congestion()` 模拟 Aurora PostgreSQL 数据库集群的磁盘拥塞情况。

磁盘拥塞模拟期间，Aurora PostgreSQL 数据库集群会将磁盘区段标记为拥塞。在模拟持续时间内，对这些区段的请求将在指定的最小和最大延迟时间之间延迟。

语法

```
SELECT aurora_inject_disk_congestion(
   percentage_of_failure,
   index,
   is_disk,
   time_interval,
   minimum,
   maximum
);
```

选项

该错误注入查询采用以下参数：

percentage_of_failure

在故障事件期间标记为拥塞的磁盘百分比。这是介于 0 到 100 之间的双精度值。如果指定 0，则不会将任何磁盘标记为拥塞。如果指定 100，则整个磁盘将标记为拥塞。

index

用于模拟故障事件的特定逻辑数据块或存储节点。

如果超出了数据的可用逻辑数据块或数据存储节点的范围，您将收到一条错误，告知您可指定的最大索引值。要避免此错误，请参阅 显示 Aurora PostgreSQL 数据库集群的卷状态。

is_disk

指示注入失败是针对逻辑数据块还是存储节点。指定 true 意味着注入失败针对逻辑数据块。指定 false 意味着注入失败针对存储节点。

time_interval

模拟磁盘拥塞情况的时间长度。间隔以秒为单位。例如，如果值为 20，则模拟会运行 20 秒。

最小值，最大值

拥塞延迟的最小和最大时间长度（以毫秒为单位）。有效值范围为 0.0 至 100.0 毫秒。标记为拥塞的磁盘区段，在模拟持续时间内将随机延迟一段时间，该时间在最小和最大时间之间。最大值必须大于最小值。

显示 Aurora PostgreSQL 数据库集群的卷状态

在 Amazon Aurora 中，数据库集群卷中包含一组逻辑块。每个逻辑块代表 10 GB 分配的存储空间。这些块称为保护组。

每个保护组中的数据在六个物理存储设备中进行复制，这些存储设备称为存储节点。这些存储节点分配给数据库集群所在区域中的三个可用区 (AZ)。而每个存储节点又包含数据库集群卷的一个或多个逻辑数据块。有关保护组和存储节点的更多信息，请参阅 AWS 数据库博客上的 Aurora 存储引擎简介。要概括地了解有关 Aurora 集群卷的更多信息，请参阅Amazon Aurora 存储。

使用 `aurora_show_volume_status()` 函数返回以下服务器状态变量：

- `Disks`：数据库集群卷的逻辑数据块总数。

- Nodes：数据库集群卷的存储节点总数。

使用 aurora_show_volume_status() 故障注入函数时，您可以使用 aurora_inject_disk_failure() 函数来帮助避免错误。aurora_inject_disk_failure() 故障注入函数模拟整个存储节点的故障，或存储节点中单个逻辑数据块的故障。在函数中，您应指定特定逻辑数据块或存储节点的索引值。不过，如果您指定的索引值大于数据库集群卷使用的逻辑数据块或存储节点数，该语句将返回错误。有关错误注入查询的更多信息，请参阅 使用错误注入查询测试 Amazon Aurora PostgreSQL。

> **Note**
>
> aurora_show_volume_status() 函数可用于 Aurora PostgreSQL 版本 10.11。有关 Aurora PostgreSQL 版本的更多信息，请参阅Amazon Aurora PostgreSQL 版本和引擎版本。

语法

```
SELECT * FROM aurora_show_volume_status();
```

示例

```
customer_database=> SELECT * FROM aurora_show_volume_status();
 disks | nodes
-------+-------
    96 |    45
```

指定 stats_temp_directory 的 RAM 磁盘

您可使用 Aurora PostgreSQL 参数 rds.pg_stat_ramdisk_size 指定分配给 RAM 磁盘的用于存储 PostgreSQL stats_temp_directory 的系统内存。RAM 磁盘参数仅在 Aurora PostgreSQL 14 及更低版本中可用。

在某些工作负载下，设置该参数可提高性能并降低 IO 要求。有关 stats_temp_directory 的更多信息，请参阅 PostgreSQL 文档中的运行时统计数据。从 PostgreSQL 版本 15 开始，PostgreSQL 社区已改用动态共享内存。因此，无需设置 stats_temp_directory。

要为您的 stats_temp_directory 启用 RAM 磁盘，可在数据库集群使用的数据库集群参数组中将 rds.pg_stat_ramdisk_size 参数设置为非零值。此参数表示 MB，因此必须使用整数值。表达

式、公式和函数对 rds.pg_stat_ramdisk_size 参数无效。请务必重启数据库集群，以使更改生效。有关设置参数的信息，请参阅Amazon Aurora 的参数组。有关重新启动数据库集群的更多信息，请参阅重启 Amazon Aurora 数据库集群或 Amazon Aurora 数据库实例。

例如，以下 AWS CLI 命令将 RAM 磁盘参数设置为 256MB。

```
aws rds modify-db-cluster-parameter-group \
    --db-cluster-parameter-group-name db-cl-pg-ramdisk-testing \
    --parameters "ParameterName=rds.pg_stat_ramdisk_size, ParameterValue=256,
 ApplyMethod=pending-reboot"
```

在重新启动数据库集群后，请运行以下命令查看 stats_temp_directory 的状态：

```
postgres=> SHOW stats_temp_directory;
```

该命令应返回以下内容：

```
stats_temp_directory
----------------------------
/rdsdbramdisk/pg_stat_tmp
(1 row)
```

使用 PostgreSQL 管理临时文件

在 PostgreSQL 中，一个复杂的查询可能会同时执行几个排序或哈希操作，每个操作都使用实例内存来存储结果，直至达到在 work_mem 参数中指定的值。当实例内存不足时，会创建临时文件来存储结果。这些文件写入磁盘以完成查询执行。稍后，将在查询完成后自动删除这些文件。在 Aurora PostgreSQL，这些文件与其他日志文件共享本地存储。您可以通过观看 Amazon CloudWatch 的 FreeLocalStorage 指标来监控 Aurora PostgreSQL 数据库集群的本地存储空间。有关更多信息，请参阅排查本地存储问题。

对于涉及多个并发查询的工作负载（这会增加临时文件的使用量），我们建议使用 Aurora 优化型读取集群。这些集群使用基于本地非易失性存储规范（NVMe）的固态硬盘（SSD）块级存储来存放临时文件。有关更多信息，请参阅 使用 Aurora 优化型读取功能提高 Aurora PostgreSQL 的查询性能。

您可以使用下面的参数和函数来管理实例中的临时文件。

- **temp_file_limit** – 此参数取消任何超过 temp_files 大小（以 KB 为单位）的查询。此限制可防止任何查询无休止地运行并使用临时文件消耗磁盘空间。您可以使用来自 log_temp_files 参数的结

果来估计该值。作为最佳实践，请检查工作负载行为并根据估计值设置限制。以下示例显示了当查询超过限制时如何取消查询。

```
postgres=>select * from pgbench_accounts, pg_class, big_table;
```

```
ERROR: temporary file size exceeds temp_file_limit (64kB)
```

- **log_temp_files** – 当删除会话的临时文件时，此参数会向 postgresql.log 发送消息。此参数在查询成功完成后生成日志。因此，它可能无助于对长时间运行的活跃查询进行故障排除。

  以下示例显示，当查询成功完成后，条目将记录在 postgresql.log 文件中，同时清理临时文件。

```
2023-02-06 23:48:35 UTC:205.251.233.182(12456):adminuser@postgres:[31236]:LOG:
 temporary file: path "base/pgsql_tmp/pgsql_tmp31236.5", size 140353536
2023-02-06 23:48:35 UTC:205.251.233.182(12456):adminuser@postgres:[31236]:STATEMENT:
  select a.aid from pgbench_accounts a, pgbench_accounts b where a.bid=b.bid order by
 a.bid limit 10;
2023-02-06 23:48:35 UTC:205.251.233.182(12456):adminuser@postgres:[31236]:LOG:
 temporary file: path "base/pgsql_tmp/pgsql_tmp31236.4", size 180428800
2023-02-06 23:48:35 UTC:205.251.233.182(12456):adminuser@postgres:[31236]:STATEMENT:
  select a.aid from pgbench_accounts a, pgbench_accounts b where a.bid=b.bid order by
 a.bid limit 10;
```

- **pg_ls_tmpdir** – 此函数在 RDS for PostgreSQL 13 及更高版本中提供，可让用户了解当前临时文件使用情况。完成的查询不会出现在该函数的结果中。在以下示例中，您可以查看此函数的结果。

```
postgres=>select * from pg_ls_tmpdir();
```

```
      name       |    size    |      modification
-----------------+------------+------------------------
 pgsql_tmp8355.1 | 1072250880 | 2023-02-06 22:54:56+00
 pgsql_tmp8351.0 | 1072250880 | 2023-02-06 22:54:43+00
 pgsql_tmp8327.0 | 1072250880 | 2023-02-06 22:54:56+00
 pgsql_tmp8351.1 |  703168512 | 2023-02-06 22:54:56+00
```

```
 pgsql_tmp8355.0 | 1072250880 | 2023-02-06 22:54:00+00
 pgsql_tmp8328.1 |  835031040 | 2023-02-06 22:54:56+00
 pgsql_tmp8328.0 | 1072250880 | 2023-02-06 22:54:40+00
(7 rows)
```

```
postgres=>select query from pg_stat_activity where pid = 8355;

query
------------------------------------------------------------------------------------------
select a.aid from pgbench_accounts a, pgbench_accounts b where a.bid=b.bid order by
 a.bid
(1 row)
```

文件名包括生成了临时文件的会话的处理 ID（PID）。更高级的查询（如以下示例所示）对每个 PID 的临时文件执行总和。

```
postgres=>select replace(left(name, strpos(name, '.')-1),'pgsql_tmp','') as pid,
 count(*), sum(size) from pg_ls_tmpdir() group by pid;
```

```
 pid  | count |   sum
------+-------+------------------
 8355 |     2 | 2144501760
 8351 |     2 | 2090770432
 8327 |     1 | 1072250880
 8328 |     2 | 2144501760
(4 rows)
```

- **pg_stat_statements** – 如果您激活 pg_stat_sat_statements 参数，则可以查看每个调用的平均临时文件使用量。您可以识别查询的 query_id 并使用它来检查临时文件使用情况，如以下示例所示。

```
postgres=>select queryid from pg_stat_statements where query like 'select a.aid from
 pgbench%';
```

```
       queryid
----------------------
 -7170349228837045701
```

```
(1 row)
```

```
postgres=>select queryid, substr(query,1,25), calls, temp_blks_read/calls
 temp_blks_read_per_call, temp_blks_written/calls temp_blks_written_per_call from
 pg_stat_statements where queryid = -7170349228837045701;
```

```
       queryid        |          substr           | calls | temp_blks_read_per_call |
 temp_blks_written_per_call
----------------------+---------------------------+-------+-------------------------
+---------------------------
 -7170349228837045701 | select a.aid from pgbench |    50 |                  239226 |
                      388678
(1 row)
```

- **Performance Insights** – 在性能详情控制面板中，可以通过开启指标 temp_bytes 和 temp_files 来查看临时文件使用情况。然后，您可以看到这两个指标的平均值，并查看它们与查询工作负载的对应关系。性能详情中的视图并未专门显示正在生成临时文件的查询。但是，当您将性能详情与针对 pg_ls_tmpdir 显示的查询相结合时，您可以排查、分析并确定查询工作负载的变化。

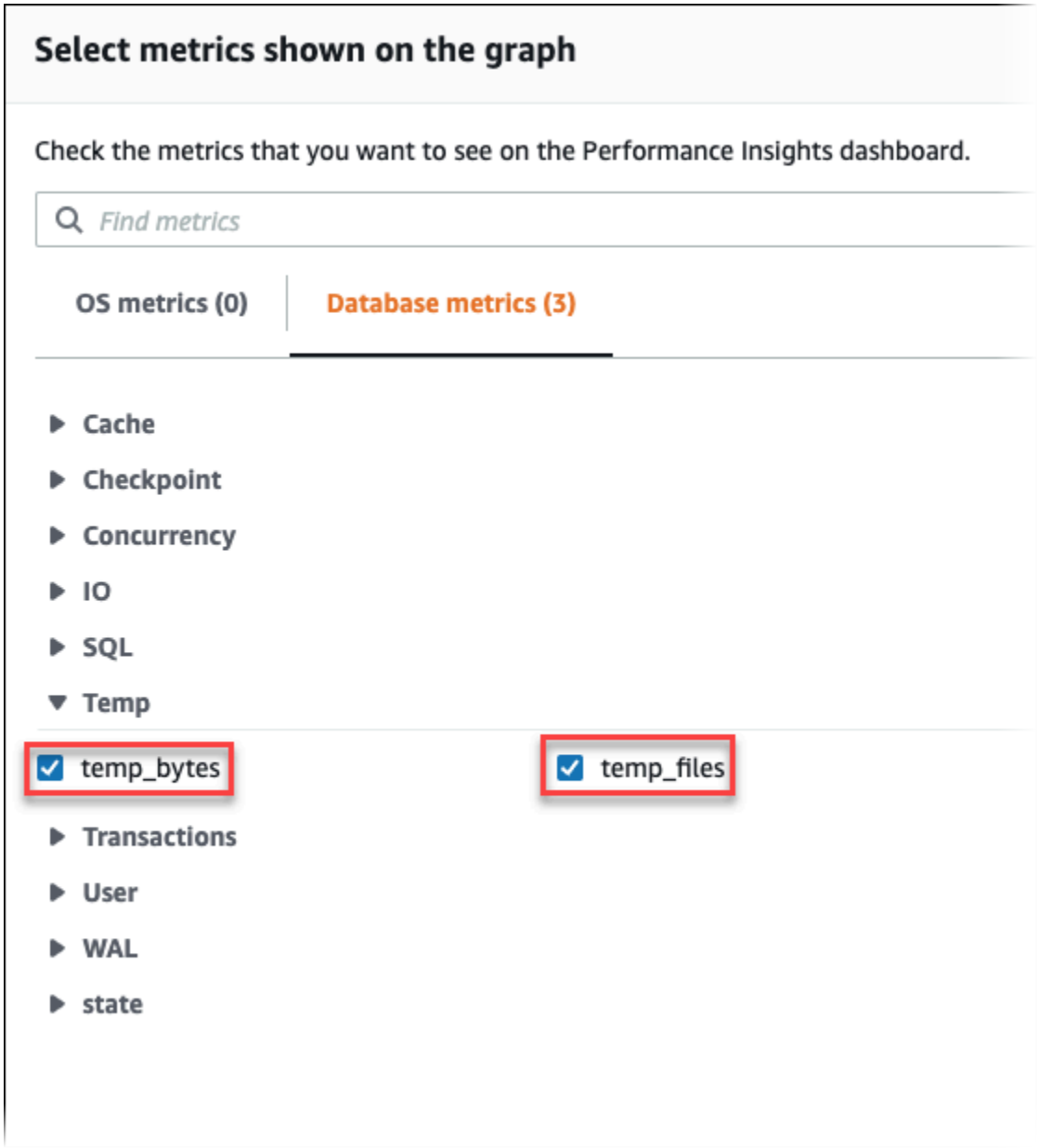
  有关如何使用性能详情分析指标和查询的更多信息，请参阅使用 Performance Insights 控制面板分析指标。

  有关使用性能详情查看临时文件使用情况的示例，请参阅使用性能详情查看临时文件使用情况

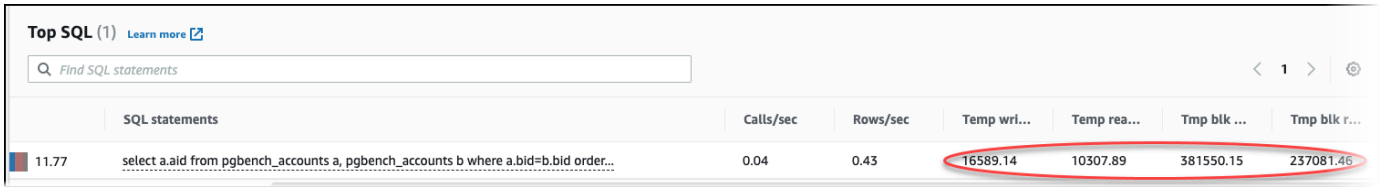
使用性能详情查看临时文件使用情况

可以使用性能详情，通过开启指标 temp_bytes 和 temp_files 来查看临时文件使用情况。性能详情中的视图不显示生成临时文件的特定查询，但是，当您将性能详情与针对 pg_ls_tmpdir 显示的查询相结合时，可以排查、分析并确定查询工作负载的变化。

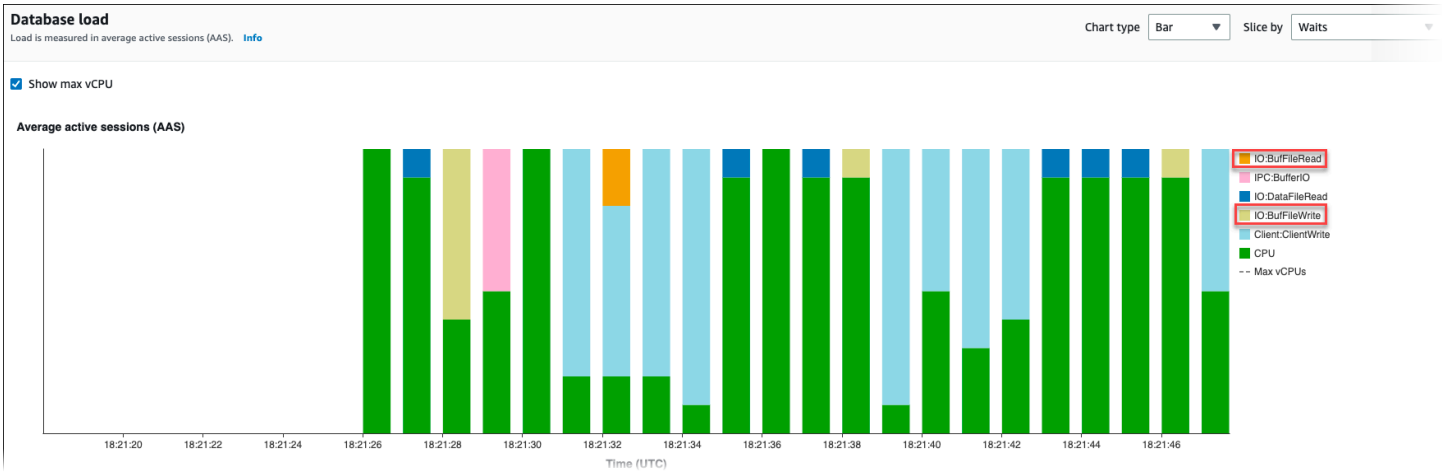
1. 在性能详情控制面板中，选择管理指标。
2. 选择数据库指标，然后选择 temp_bytes 和 temp_files 指标，如下图所示。



3. 在 Top SQL 选项卡中，选择首选项图标。
4. 在首选项窗口中，打开 Top SQL 选项卡中显示的以下统计数据，然后选择继续。
   - 临时写入次数/秒
   - 临时读取次数/秒
   - 临时批量写入/调用
   - 临时批量读取/调用
5. 当临时文件与针对 `pg_ls_tmpdir` 显示的查询相组合时，临时文件将被分解，如以下示例所示。



当您的工作负载中排名靠前的查询经常创建临时文件时，就会发生 IO:BufFileRead 和 IO:BufFileWrite 事件。通过查看“数据库负载”和“热门 SQL”部分中的平均活动会话（AAS），您可以使用性能详情来确定在 IO:BufFileRead 和 IO:BufFileWrite 上等待的热门 SQL。



有关如何使用性能详情按等待事件分析热门查询和负载的更多信息，请参阅“Top SQL”（主要 SQL）选项卡概览。您应该识别和调整导致临时文件使用量和相关等待事件增加的查询。有关这些等待事件和补救措施的更多信息，请参阅 IO:BufFileRead 和 IO:BufFileWrite。

> **Note**
>
> work_mem 参数控制排序操作何时耗尽内存以及结果何时写入临时文件中。我们建议您不要将此参数的设置更改为高于默认值，因为这将允许每个数据库会话消耗更多内存。此外，执行复杂联接和排序的单个会话可以执行并行操作，其中每个操作都会消耗内存。
> 作为最佳实践，当您有一个包含多个联接和排序的大型报告时，请使用 SET work_mem 命令在会话级别设置此参数。然后，更改仅应用于当前会话，而不会全局更改该值。

使用 Aurora PostgreSQL 的等待事件进行优化

等待事件是 Aurora PostgreSQL 的重要优化工具。当您能查明会话为什么在等待资源以及会话在做什么时，您就能更好地减少瓶颈。您可以使用本节中的信息来查找可能的原因和纠正措施。在深入研究本节之前，我们强烈建议您了解基本的 Aurora 概念，尤其是以下主题：

- Amazon Aurora 存储
- 管理 Aurora 数据库集群的性能和扩展

> **Important**
>
> 本节中的等待事件特定于 Aurora PostgreSQL。使用本节中的信息仅能优化 Amazon Aurora，而不能优化 RDS for PostgreSQL。
> 本节中的一些等待事件在这些数据库引擎的开源版本中没有类似内容。其他等待事件与开源引擎中的事件名称相同，但行为不同。例如，Amazon Aurora 存储的工作原理与开源存储不同，因此与存储相关的等待事件表明不同的资源状况。

主题

- Aurora PostgreSQL 优化的基本概念
- Aurora PostgreSQL 等待事件
- Client:ClientRead
- Client:ClientWrite
- CPU
- IO:BufFileRead 和 IO:BufFileWrite
- IO:DataFileRead
- IO:XactSync
- IPC:DamRecordTxAck
- IPC:并行等待事件
- Lock:advisory
- Lock:extend
- Lock:Relation
- Lock:transactionid
- Lock:tuple
- LWLock:buffer_content (BufferContent)
- LWLock:buffer_mapping
- LWLock:BufferIO (IPC:BufferIO)
- LWLock:lock_manager

- LWLock:MultiXact
- LWLock:pg_stat_statements
- Timeout:PgSleep

Aurora PostgreSQL 优化的基本概念

在优化 Aurora PostgreSQL 数据库之前，请务必了解等待事件以及它们发生的原因。还可以查看 Aurora PostgreSQL 的基本内存和磁盘架构。有关有用的架构图，请参阅 PostgreSQL wikibook。

主题

- Aurora PostgreSQL 等待事件
- Aurora PostgreSQL 内存
- Aurora PostgreSQL 过程

Aurora PostgreSQL 等待事件

等待事件表示会话正在等待的资源。例如，当 Aurora PostgreSQL 等待从客户端接收数据时，会发生等待事件 `Client:ClientRead`。会话等待的典型资源包括以下内容：

- 例如，当会话试图修改缓冲区时，对缓冲区的单线程访问
- 当前被另一个会话锁定的行
- 已读取一个数据文件
- 已写入一个日志文件

例如，为了满足查询，会话可能会执行完整的表扫描。如果数据尚未在内存中，会话将等待磁盘输入/输出完成。当缓冲区读取到内存时，会话可能需要等待，因为其他会话正在访问相同的缓冲区。数据库使用预定义的等待事件记录等待。这些事件按类别进行分组。

等待事件本身并不显示性能问题。例如，如果请求的数据不在内存中，则必须从磁盘读取数据。如果一个会话锁定行以进行更新，则另一个会话将等待解锁该行，以便它可以更新该行。提交需要等待对日志文件的写入完成。等待是数据库正常运行不可或缺的组成部分。

大量等待事件通常显示性能问题。在这种情况下，您可以使用等待事件数据来确定会话将时间花费在哪里。例如，如果通常在几分钟内运行的报告现在运行了数小时，则可以确定对总等待时间贡献最大的等待事件。如果您能确定顶级等待事件的原因，您有时就可以进行更改来提高性能。例如，如果您的会话正在等待已被另一个会话锁定的行，则可以结束锁定会话。

Aurora PostgreSQL 内存

Aurora PostgreSQL 内存分为共享内存和本地内存。

主题

- Aurora PostgreSQL 中的共享内存
- Aurora PostgreSQL 中的本地内存

Aurora PostgreSQL 中的共享内存

Aurora PostgreSQL 会在实例启动时分配共享内存。共享内存分为多个子区域。在下文中，您可以找到最重要子区域的说明。

主题

- 共享缓冲区
- 预写日志 (WAL) 缓冲区

共享缓冲区

共享缓冲池是一个 Aurora PostgreSQL 内存区域，它包含应用程序连接现在正在使用或过去正在使用的所有页面。分页是磁盘数据块的内存版本。共享缓冲池缓存从磁盘读取的数据块。该缓冲池减少了从磁盘重新读取数据的需求，从而提高了数据库的运行效率。

每个表和索引都存储为固定大小的页面数组。每个数据块包含多个元组，它们与行相对应。元组可以存储在任何页面中。

共享缓冲池的内存有限。如果新请求需要一个不在内存中的页面，并且没有更多的内存，Aurora PostgreSQL 会移除一个较少使用的页面来容纳请求。移出策略通过时钟扫描算法来实现。

`shared_buffers` 参数确定服务器用于缓存数据的内存量。

预写日志 (WAL) 缓冲区

预写日志 (WAL) 缓冲区保存 Aurora PostgreSQL 稍后写入持久存储的事务数据。使用 WAL 机制，Aurora PostgreSQL 可以执行以下操作：

- 发生故障后恢复数据
- 通过避免频繁写入磁盘来减少磁盘输入/输出

当客户端更改数据时，Aurora PostgreSQL 会将更改写入 WAL 缓冲区。当客户发出 COMMIT，WAL 写入器进程将事务数据写入 WAL 文件。

wal_level 参数决定向 WAL 写入多少信息。

Aurora PostgreSQL 中的本地内存

每个后端进程都会为查询处理分配本地内存。

主题

- 工作内存区域
- 维护工作内存区域
- 临时缓冲区

工作内存区域

工作内存区域保存执行排序和哈希的查询的临时数据。例如，包含 ORDER BY 子句的查询执行排序。查询在哈希联接和聚合中使用哈希表。

work_mem 参数表示在写入临时磁盘文件之前内部排序操作和哈希表要使用的内存量。原定设置值为 4MB。可以同时运行多个会话，且每个会话可以并行运行维护操作。出于这个原因，使用的总工作内存可以是 work_mem 设置的倍数。

维护工作内存区域

维护工作内存区域缓存数据以进行维护操作。这些操作包括 vacuum 操作、创建索引和添加外键。

maintenance_work_mem 参数指定维护操作要使用的最大内存量。原定设置值为 64MB。一个数据库会话一次只能运行一个维护操作。

临时缓冲区

临时缓冲区缓存每个数据库会话的临时表。

每个会话都根据需要分配临时缓冲区，但不超过您指定的限制。当会话结束时，服务器将清除缓冲区。

temp_buffers 参数设置每个会话使用的临时缓冲区的最大数量。在会话中首次使用临时表之前，您可以更改 temp_buffers 值。

Aurora PostgreSQL 过程

Aurora PostgreSQL 使用多个过程。

主题

- 邮件管理员过程
- 后端进程
- 后台进程

邮件管理员过程

邮件管理员过程是启动 Aurora PostgreSQL 时开始的第一个过程。邮件管理员过程负有以下主要责任：

- 分流并监控后台进程
- 接收来自客户端进程的身份验证请求，并在允许数据库为请求提供服务之前对这些请求进行身份验证

后端进程

如果邮件管理员对客户请求进行身份验证，邮件管理员会分流一个新的后端进程，也称为 postgres 进程。一个客户端进程只连接到一个后端进程。客户端进程和后端进程直接通信，而无需邮件管理员过程的干预。

后台进程

邮件管理员过程会分流执行不同后端任务的几个进程。其中一些更重要的事项包括：

- WAL 写入器

  Aurora PostgreSQL 会将 WAL（预写日志记录）缓冲区中的数据写入日志文件。预写日志记录的原则是，在数据库将描述这些更改的日志记录写入磁盘之后，数据库才能将更改写入数据文件。WAL 机制减少了磁盘输入/输出，并允许 Aurora PostgreSQL 在出现故障后使用日志恢复数据库。
- 后台写入器

  此进程会定期将内存缓冲区中的脏（已修改）分页写入数据文件。当后端进程在内存中修改分页时，分页会变脏。
- Autovacuum 守护进程

  守护进程由以下各项组成：

  - Autovacuum 启动程序
  - Autovacuum 工件进程

当 Autovacuum 开启时，它会检查包含大量插入的、更新的或删除的元组的表。守护进程要承担以下责任：

- 恢复或重复使用更新或删除的行占用的磁盘空间
- 更新计划人员使用的统计数据
- 防止因事务 ID 重叠而导致旧数据丢失

Autovacuum 功能自动执行 VACUUM 和 ANALYZE 命令。VACUUM 具有以下变体：标准和完整版。标准 vacuum 与其他数据库操作并行运行。VACUUM FULL 需要对您工作所在的表具有专有锁定。因此，它不能与访问同一表的操作并行运行。VACUUM 创建了大量的输入/输出流量，这可能会导致其他活动会话的性能不佳。

Aurora PostgreSQL 等待事件

下表列出了 Aurora PostgreSQL 最常指示性能问题的等待事件，并总结了最常见的原因和纠正措施。以下等待事件是 Amazon Aurora PostgreSQL 等待事件 中的列表子集。

等待事件	定义
Client:ClientRead	当 Aurora PostgreSQL 等待从客户端接收数据时，会发生此事件。
Client:ClientWrite	当 Aurora PostgreSQL 等待将数据写入客户端时，会发生此事件。
CPU	当线程在 CPU 中处于活动状态或正在等待 CPU 时，会发生此事件。
IO:BufFileRead 和 IO:BufFileWrite	这些事件发生在 Aurora PostgreSQL 创建临时文件时。
IO:DataFileRead	当由于分页在共享内存中不可用，连接等待后端进程从存储中读取所需分页时，会发生此事件。
IO:XactSync	当数据库等待 Aurora 存储子系统确认常规事务的提交，或者准备好的事务的提交或回滚时，会发生此事件。

等待事件	定义
IPC:DamRecordTxAck	当 Aurora PostgreSQL 在使用数据库活动流的会话中生成活动流事件，然后等待该事件变为持久事件时，会发生此事件。
Lock:advisory	当 PostgreSQL 应用程序使用锁定来协调多个会话之间的活动时，会发生此事件。
Lock:extend	当后端进程正在等待锁定关系以对其进行扩展，而另一个进程出于同样目的锁定该关系时，会发生此事件。
Lock:Relation	当查询等待获取当前被另一个事务锁定的表或视图上的锁定时，会发生此事件。
Lock:transactionid	当事务正在等待行级锁定时，会发生此事件。
Lock:tuple	在后端进程等待获取元组锁定时，会发生此事件。
LWLock:buffer_content (BufferContent)	当某个会话等待读取或写入内存中的某个数据页面，而另一个会话正锁定该页面以进行写入时，会发生此事件。
LWLock:buffer_mapping	当会话正在等待将数据块与共享缓冲池中的缓冲区关联起来时，会发生此事件。
LWLock:BufferIO (IPC:BufferIO)	当 Aurora PostgreSQL 或 RDS for PostgreSQL 正在等待其他进程在同时尝试访问页面时完成输入/输出 (I/O) 操作时，会发生此事件。
LWLock:lock_manager	当 Aurora PostgreSQL 引擎维护共享锁的内存区域以便在无法使用快速路径锁时分配、检查和取消分配锁时，会发生此事件。

等待事件	定义
LWLock:MultiXact	当 Aurora PostgreSQL 保持会话处于打开状态以完成涉及表中同一行的多个事务时，就会发生此类事件。等待事件表示多事务处理的哪个方面正在生成等待事件，即 LWLock:MultiXactOffsetSLRU、LWLock:MultiXactOffsetBuffer、LWLock:MultiXactMemberSLRU 或 LWLock:MultiXactMemberBuffer。
Timeout:PgSleep	当服务器进程调用 pg_sleep 函数并且正在等待睡眠超时过期时，会发生此事件。

Client:ClientRead

当 Aurora PostgreSQL 等待从客户端接收数据时，会发生 `Client:ClientRead` 事件。

主题

- 支持的引擎版本
- 上下文
- 等待次数增加的可能原因
- 操作

支持的引擎版本

Aurora PostgreSQL 版本 10 及更高版本支持此等待事件信息。

上下文

Aurora PostgreSQL 数据库集群正在等待从客户端接收数据。Aurora PostgreSQL 数据库集群必须先从客户端接收数据，然后才能向客户端发送更多数据。集群在从客户端接收数据之前等待的时间为 `Client:ClientRead` 事件。

等待次数增加的可能原因

`Client:ClientRead` 显示在主要等待中的常见原因包括以下各项：

网络延迟增加

Aurora PostgreSQL 数据库集群和客户端之间的网络延迟可能会增加。较高的网络延迟会增加数据库集群从客户端接收数据所需的时间。

客户端负载增加

客户端上可能存在 CPU 压力或网络饱和。客户端负载的增加可能会延迟从客户端向 Aurora PostgreSQL 数据库集群传输数据的时间。

过多的网络往返次数

Aurora PostgreSQL 数据库集群和客户端之间的大量网络往返可能会延迟将数据从客户端传输到 Aurora PostgreSQL 数据库集群的过程。

大型复制操作

在复制操作期间，数据将从客户端的文件系统传输到 Aurora PostgreSQL 数据库集群。向数据库集群发送大量数据可能会延迟从客户端向数据库集群传输数据的时间。

空闲客户端连接

与 Aurora PostgreSQL 数据库实例的连接在事务状态下处于空闲状态，正在等待客户端发送更多数据或发出命令。这种状态可能会导致 `Client:ClientRead` 事件增加。

用于连接池的 PgBouncer

PgBouncer 有一个名为 `pkt_buf` 的低级网络配置设置，预设情况下设置为 4096。如果工作负载通过 PgBouncer 发送大于 4096 字节的查询数据包，我们建议增加 `pkt_buf` 设置为 8192。如果新设置没有减少 `Client:ClientRead` 事件的数量，我们建议增加 `pkt_buf` 设置为较大的值，例如 16384 或 32768。如果查询文本很大，则较大的设置可能会特别有用。

操作

根据等待事件的原因，我们建议采取不同的操作。

主题

- 将客户端放置在与集群相同的可用区和 VPC 子网中。
- 扩展客户端
- 使用当前一代实例
- 提高网络带宽

- 监控网络性能的最大值
- 监控处于“空闲事务”状态的事务

将客户端放置在与集群相同的可用区和 VPC 子网中。

为了减少网络延迟并提高网络吞吐量，请将客户端放在与 Aurora PostgreSQL 数据库集群相同的可用区和 Virtual Private Cloud (VPC) 子网中。确保客户端在地理位置上尽可能靠近数据库集群。

扩展客户端

使用 Amazon CloudWatch 或其他主机指标，确定您的客户端当前是受 CPU 或网络带宽的限制，还是受此两者的限制。如果客户端受到限制，请相应地扩展您的客户端。

使用当前一代实例

在某些情况下，您可能没有使用支持巨型帧的数据库实例类。如果您在 Amazon EC2 上运行应用程序，请考虑为客户端使用当前一代实例。另外，在客户端操作系统上配置最大传输单位 (MTU)。这种技术可能会减少网络往返次数并提高网络吞吐量。有关更多信息，请参阅《Amazon EC2 用户指南》中的巨型帧（9001 MTU）。

有关数据库实例类的信息，请参阅 Amazon Aurora 数据库实例类。要确定等同于 Amazon EC2 实例类型的数据库实例类，请将 `db.` 放置在 Amazon EC2 实例类型名称之前。例如，`r5.8xlarge` Amazon EC2 实例等同于 `db.r5.8xlarge` 数据库实例类。

提高网络带宽

使用 `NetworkReceiveThroughput` 和 `NetworkTransmitThroughput` Amazon CloudWatch 指标监控数据库集群上的传入和传出网络流量。这些指标可以帮助您确定网络带宽是否足以满足您的工作负载。

如果您的网络带宽不够，请增加它。如果 AWS 客户端或您的数据库实例已达到网络带宽限制，增加带宽的唯一方法是增加数据库实例大小。

有关 CloudWatch 指标的更多信息，请参阅Amazon Aurora 的 Amazon CloudWatch 指标。

监控网络性能的最大值

如果您使用的是 Amazon EC2 客户端，Amazon EC2 会提供网络性能指标的最大值，包括聚合入站和出站网络带宽。它还提供连接跟踪功能，以确保按预期返回数据包以及域名系统 (DNS) 等服务的链接本地服务访问。要监控这些最大值，请使用当前的增强型联网驱动程序并监控客户端的网络性能。

有关更多信息，请参阅《Amazon EC2 用户指南》中的监控您的 Amazon EC2 实例的网络性能，以及《Amazon EC2 用户指南》中的监控您的 Amazon EC2 实例的网络性能。

监控处于“空闲事务”状态的事务

检查您是否有越来越多的 `idle in transaction` 连接。要做到这一点，请监控 `pg_stat_activity` 表中的 `state` 列。您可能能够通过运行类似于以下内容的查询来识别连接源。

```
select client_addr, state, count(1) from pg_stat_activity
where state like 'idle in transaction%'
group by 1,2
order by 3 desc
```

Client:ClientWrite

当 Aurora PostgreSQL 等待将数据写入客户端时，会发生 `Client:ClientWrite` 事件。

主题

- 支持的引擎版本
- 上下文
- 等待次数增加的可能原因
- 操作

支持的引擎版本

Aurora PostgreSQL 版本 10 及更高版本支持此等待事件信息。

上下文

客户端进程必须先读取从 Aurora PostgreSQL 数据库集群接收的所有数据，然后集群才能发送更多数据。集群在将更多数据发送给客户端之前等待的时间为 `Client:ClientWrite` 事件。

Aurora PostgreSQL 数据库集群与客户端之间的网络吞吐量降低可能会导致此事件。客户端的 CPU 压力和网络饱和也可能导致此事件。CPU 压力是 CPU 被充分利用并且有任务等待 CPU 时间的时间。网络饱和是当数据库和客户端之间的网络传输的数据超出其处理能力之时。

等待次数增加的可能原因

`Client:ClientWrite` 显示在主要等待中的常见原因包括以下各项：

网络延迟增加

Aurora PostgreSQL 数据库集群和客户端之间的网络延迟可能会增加。较高的网络延迟会增加客户端接收数据所需的时间。

客户端负载增加

客户端上可能存在 CPU 压力或网络饱和。客户端负载的增加会延迟从 Aurora PostgreSQL 数据库集群接收数据的过程。

发送到客户端的大量数据

Aurora PostgreSQL 数据库集群可能会向客户端发送大量数据。客户端可能无法像集群发送数据那样地快速接收数据。诸如大型表的副本之类的活动可能会导致 `Client:ClientWrite` 事件增加。

操作

根据等待事件的原因，我们建议采取不同的操作。

主题

- 将客户端放置在与集群相同的可用区和 VPC 子网中。
- 使用当前一代实例
- 减少发送到客户端的数据量
- 扩展客户端

将客户端放置在与集群相同的可用区和 VPC 子网中。

为了减少网络延迟并提高网络吞吐量，请将客户端放在与 Aurora PostgreSQL 数据库集群相同的可用区和 Virtual Private Cloud (VPC) 子网中。

使用当前一代实例

在某些情况下，您可能没有使用支持巨型帧的数据库实例类。如果您在 Amazon EC2 上运行应用程序，请考虑为客户端使用当前一代实例。另外，在客户端操作系统上配置最大传输单位 (MTU)。这种技术可能会减少网络往返次数并提高网络吞吐量。有关更多信息，请参阅《Amazon EC2 用户指南》中的巨型帧（9001 MTU）。

有关数据库实例类的信息，请参阅 Amazon Aurora 数据库实例类。要确定等同于 Amazon EC2 实例类型的数据库实例类，请将 `db.` 放置在 Amazon EC2 实例类型名称之前。例如，`r5.8xlarge` Amazon EC2 实例等同于 `db.r5.8xlarge` 数据库实例类。

减少发送到客户端的数据量

如果可能，请调整应用程序以减少 Aurora PostgreSQL 数据库集群发送给客户端的数据量。进行这样的调整可以减轻客户端上的 CPU 和网络争用。

扩展客户端

使用 Amazon CloudWatch 或其他主机指标，确定您的客户端当前是受 CPU 或网络带宽的限制，还是受此两者的限制。如果客户端受到限制，请相应地扩展您的客户端。

CPU

当线程在 CPU 中处于活动状态或正在等待 CPU 时，会发生此事件。

主题

- 支持的引擎版本
- 上下文
- 等待次数增加的可能原因
- 操作

支持的引擎版本

此等待时间信息与 Aurora PostgreSQL 版本 9.6 及更高版本相关。

上下文

中央处理单元 (CPU) 是运行指令的计算机的组件。例如，CPU 指令执行算术运算并在内存中交换数据。如果查询增加了通过数据库引擎执行的指令的数量，则运行查询所花费的时间将增加。CPU 调度正在为进程提供 CPU 时间。调度由操作系统的内核编排。

主题

- 如何判断此等待何时发生
- DBLoadCPU 指标
- os.cpuUtilization 指标
- CPU 调度的可能原因

如何判断此等待何时发生

该 CPU 等待事件表示后端进程在 CPU 中处于活动状态或正在等待 CPU。您知道，当查询显示以下信息时会发生这种情况：

- The pg_stat_activity.state column has the value active。
- pg_stat_activity 中的 wait_event_type 和 wait_event 列都是 null。

要查看正在使用或等待 CPU 的后端进程，请运行以下查询。

```
SELECT *
FROM   pg_stat_activity
WHERE  state = 'active'
AND    wait_event_type IS NULL
AND    wait_event IS NULL;
```

DBLoadCPU 指标

CPU 的性能详情指标为 DBLoadCPU。DBLoadCPU 的值可能与 Amazon CloudWatch 指标 CPUUtilization 的值不同。后一个指标是从 Hypervisor 中收集的，用于数据库实例。

os.cpuUtilization 指标

性能详情操作系统指标提供有关 CPU 利用率的详细信息。例如，您可以显示以下指标：

- os.cpuUtilization.nice.avg
- os.cpuUtilization.total.avg
- os.cpuUtilization.wait.avg
- os.cpuUtilization.idle.avg

性能详情将数据库引擎的 CPU 使用情况报告为 os.cpuUtilization.nice.avg。

CPU 调度的可能原因

从操作系统的角度来看，CPU 在未运行空闲线程时处于活动状态。CPU 在执行计算时处于活动状态，但在等待内存输入/输出时也处于活动状态。这种类型的输入/输出主导着典型的数据库工作负载。

满足以下条件时，进程可能会等待获得 CPU 调度：

- CloudWatch CPUUtilization 指标接近 100%。
- 平均负载大于 vCPU 的数量，表示负载过重。您可以在性能详情中的操作系统指标部分找到 loadAverageMinute 指标。

等待次数增加的可能原因

当此事件的发生率超过正常（可能表示性能问题）时，典型的原因包括以下几点。

主题

- 突然猛增的可能原因
- 长期高频的可能原因
- 极端状况

突然猛增的可能原因

突然猛增的最可能原因如下：

- 您的应用程序打开了太多与数据库同时连接。这种情况被称为“连接风暴”。
- 您的应用程序工作负载在以下任一方面发生变化：
  - 新查询
  - 数据集的大小增加
  - 索引维护或创建
  - 新函数
  - 新的运营商
  - 并行查询执行增加
- 您的查询执行计划已更改。在某些情况下，更改可能会导致缓冲区增加。例如，查询之前使用索引，而现在正在使用顺序扫描。在这种情况下，查询需要更多的 CPU 才能实现同样的目标。

长期高频的可能原因

长期反复出现的事件的最可能原因：

- CPU 上同时运行的后端进程太多。这些进程可以是并行工件。
- 查询执行不佳，因为它们需要大量缓冲区。

极端状况

如果所有可能的原因都不是实际原因，则可能会发生以下情况：

- CPU 正在换入和换出进程。
- CPU 上下文切换有所增加。
- Aurora PostgreSQL 代码缺少等待事件。

操作

如果 CPU 等待事件主导着数据库活动，它不一定表示性能问题。只在性能下降时应对此事件。

主题

- 调查数据库是否导致 CPU 增加
- 确定连接数量是否增加
- 响应工作负载变化

调查数据库是否导致 CPU 增加

检查性能详情中的 `os.cpuUtilization.nice.avg` 指标。如果此值远低于 CPU 使用率，则非数据库进程是 CPU 的主要贡献者。

确定连接数量是否增加

检查 Amazon CloudWatch 中的 `DatabaseConnections` 指标。您的操作取决于 CPU 等待事件增加期间该数量是增加还是减少。

连接增加

如果连接数量增加，请将消耗 CPU 的后端进程数与 vCPU 的数量进行比较。以下是可能的情况：

- 消耗 CPU 的后端进程数量少于 vCPU 的数量。

  在这种情况下，连接数量不是问题。但是，您仍然可以尝试降低 CPU 利用率。
- 消耗 CPU 的后端进程数量大于 vCPU 的数量。

  在这种情况下，需考虑以下选项：
  - 减少连接到数据库的后端进程的数量。例如，实施连接池解决方案，例如 RDS 代理。要了解更多信息，请参阅 适用于 Aurora 的 Amazon RDS 代理。

- 升级实例大小以获得更多 vCPU 数量。
- 如果适用，将一些只读工作负载重新导向到读取器节点。

连接未增加

检查性能详情中的 `blks_hit` 指标。寻找 `blks_hit` 增加与 CPU 使用率之间的相关性。以下是可能的情况：

- CPU 使用率和 `blks_hit` 具有相关性。

  在这种情况下，找到与 CPU 使用率相关联的主要 SQL 语句，然后查找计划更改。您可以使用下面的方法之一：

  - 手动解释计划并将其与预期的执行计划进行比较。
  - 寻找每秒数据块命中量和每秒局部数据块命中量的增加。在性能详情控制面板的主要 SQL 部分中，选择 Preferences（首选项）。
- CPU 使用率和 `blks_hit` 无关。

  在这种情况下，请确定是否出现以下任一情况：

  - 应用程序正在快速连接到数据库并断开与数据库的连接。

    通过打开 `log_connections` 和 `log_disconnections`，然后分析 PostgreSQL 日志来诊断此行为。考虑使用 `pgbadger` 日志分析器。有关更多信息，请参阅 https://github.com/darold/pgbadger。
  - 操作系统已超载。

    在这种情况下，性能详情显示，后端进程使用 CPU 的时间比平时更长。在性能详情 `os.cpuUtilization` 指标或 CloudWatch `CPUUtilization` 指标中寻找证据。如果操作系统过载，请查看增强监控指标以进一步诊断。具体来说，请查看进程列表以及每个进程占用的 CPU 百分比。
  - 主要 SQL 语句消耗的 CPU 太多。

    检查与 CPU 使用率相关联的语句，看看它们是否可以使用更少的 CPU。运行 `EXPLAIN` 命令，并将重点放在影响最大的计划节点上。考虑使用 PostgreSQL 执行计划可视化工具。要试用此工具，请参阅 http://explain.dalibo.com/。

响应工作负载变化

如果您的工作负载发生了变化，请查找以下类型的更改：

新查询

检查是否预计会出现新的查询。如果是，请确保他们的执行计划和每秒执行次数符合预期。

数据集的大小增加

确定分区（如果尚未实施）是否有帮助。此策略可能会减少查询需要检索的页数。

索引维护或创建

检查维护时间表是否符合预期。最佳实践是在高峰活动之外安排维护活动。

新函数

检查这些功能在测试期间是否按预期运行。具体来说，检查每秒执行次数是否符合预期。

新的运营商

检查它们在测试期间是否按预期运行。

运行并行查询的增加

确定是否出现以下任一情况：

- 所涉及的关系或索引的规模突然增长，因此它们与 `min_parallel_table_scan_size` 或 `min_parallel_index_scan_size` 显著不同。
- 最近对 `parallel_setup_cost` 或 `parallel_tuple_cost` 进行了更改。
- 最近对 `max_parallel_workers` 或 `max_parallel_workers_per_gather` 进行了更改。

IO:BufFileRead 和 IO:BufFileWrite

`IO:BufFileRead` 和 `IO:BufFileWrite` 事件发生在 Aurora PostgreSQL 创建临时文件时。当操作需要的内存超过当前定义的工作内存参数时，它们会将临时数据写入持久性存储。此操作有时被称为“溢出到磁盘”。

主题

- 支持的引擎版本
- 上下文
- 等待次数增加的可能原因
- 操作

支持的引擎版本

Aurora PostgreSQL 的所有版本均支持此等待事件信息。

上下文

IO:BufFileRead 和 IO:BufFileWrite 与工作内存区域和维护工作内存区域有关。有关这些本地内存区域的更多信息，请参阅 工作内存区域 和 维护工作内存区域。

work_mem 的原定设置值为 4MB。如果一个会话并行执行操作，则处理并行性的每个工件将使用 4MB 的内存。出于此原因，请仔细设置 work_mem。如果您将值增加的太大，则运行很多会话的数据库可能会占用太多内存。如果您将值设置得太低，Aurora PostgreSQL 会在本地存储中创建临时文件。这些临时文件的磁盘输入/输出可能会降低性能。

如果观察到以下事件顺序，则数据库可能正在生成临时文件：

1. 可用性突然急剧下降
2. 可用空间的快速恢复

您可能还会看到“chainsaw”模式。此模式可能表明您的数据库在不断创建小文件。

等待次数增加的可能原因

一般来说，这些等待事件是占用内存比 work_mem 或 maintenance_work_mem 参数分配的内存更多的操作造成。为了进行补偿，操作会写入临时文件。IO:BufFileRead 和 IO:BufFileWrite 事件的常见原因包括以下内容：

需要比工作内存区域中存在的内存更多的查询

具有以下特征的查询使用工作内存区域：

- 哈希联接
- ORDER BY 子句
- GROUP BY 子句
- DISTINCT
- 窗口函数
- CREATE TABLE AS SELECT
- 具体化视图刷新

需要比维护工作内存区域中存在的内存更多的语句

以下语句使用维护工作内存区域：

- CREATE INDEX
- CLUSTER

操作

根据等待事件的原因，我们建议采取不同的操作。

主题

- 识别问题
- 请检查您的联接查询
- 检查您的 ORDER BY 和 GROUP BY 查询
- 避免使用 DISTINCT 操作
- 考虑使用窗口函数而不是 GROUP BY 函数
- 调查具体化视图和 CTAS 语句
- 在创建索引时使用 pg_repack
- 聚集表时，增加 maintenance_work_mem
- 优化内存以防止 IO:BufFileRead 和 IO:BufFileWrite

识别问题

假设存在一种情况，性能详情尚未开启的一种情况，而您怀疑 IO:BufFileRead 和 IO:BufFileWrite 的发生频率比正常情况高。执行以下操作：

1. 检查 Amazon CloudWatch 中的 FreeLocalStorage 指标。
2. 寻找一种电锯模式，该模式为一系列的交错突增。

电锯模式表示存储的快速消耗和释放，通常与临时文件有关。如果您注意到这种模式，请开启性能详情。使用性能详情时，您可以确定等待事件的发生时间以及与这些事件关联的查询。您的解决方案取决于导致事件的特定查询。

或者设置参数 log_temp_files。此参数记录生成超出临时文件阈值 KB 的所有查询。如果值为 0，Aurora PostgreSQL 会记录所有临时文件。如果值为 1024，Aurora PostgreSQL 会记录生成大于

1MB 的临时文件的所有查询。有关 log_temp_files 更多信息，请参阅 PostgreSQL 文档中的错误报告和日志记录。

请检查您的联接查询

您的应用程序可能会使用联接。例如，以下查询将四个表联接到一起。

```
SELECT *
       FROM order
 INNER JOIN order_item
       ON (order.id = order_item.order_id)
 INNER JOIN customer
       ON (customer.id = order.customer_id)
 INNER JOIN customer_address
       ON (customer_address.customer_id = customer.id AND
           order.customer_address_id = customer_address.id)
 WHERE customer.id = 1234567890;
```

临时文件使用率激增的可能原因是查询本身存在问题。例如，中断的子句可能无法正确筛选联接。考虑以下示例中的第二个内联接。

```
SELECT *
       FROM order
 INNER JOIN order_item
       ON (order.id = order_item.order_id)
 INNER JOIN customer
       ON (customer.id = customer.id)
 INNER JOIN customer_address
       ON (customer_address.customer_id = customer.id AND
           order.customer_address_id = customer_address.id)
 WHERE customer.id = 1234567890;
```

前面的查询错误地将 customer.id 与 customer.id 进行了联接，在每个客户和每个订单之间生成了笛卡尔积。这种类型的意外联接会生成大型临时文件。根据表的大小，笛卡尔查询甚至可以填满存储空间。满足以下条件时，您的应用程序可能会有笛卡尔联接：

- 您可以看到存储可用性大幅下降，然后是快速恢复。
- 现在没有创建任何索引。
- 现在没有发布任何 CREATE TABLE FROM SELECT 语句。
- 没有进行任何具体化视图的刷新。

要查看是否使用正确的键联接表，请检查查询和对象关系映射指令。请记住，应用程序的某些查询不会总是被调用，而且有些查询是动态生成的。

检查您的 ORDER BY 和 GROUP BY 查询

在某些情况下，ORDER BY 子句可能会导致过多的临时文件。请考虑以下准则：

- 当需要对它们进行排序时，只包括 ORDER BY 子句中的列。本指南对于返回数千行并在 ORDER BY 子句中指定很多列的查询尤其重要。
- 考虑创建索引以在 ORDER BY 子句与具有相同升序或降序的列匹配时对它们进行加速。部分索引更可取，因为它们较小。较小的索引可以更快地读取和遍历。
- 如果为可以接受 null 值的列创建索引，请考虑是希望将 null 值存储在索引的末尾还是在索引的开头存储。

  如果可能，通过筛选结果集来减少需要排序的行数。如果您使用 WITH 子句语句或子查询，请记住，内部查询会生成一个结果集并会将其传递给外部查询。查询可以筛选出的行越多，查询需要进行的排序就越少。
- 如果您不需要获取完整的结果集，请使用 LIMIT 子句。例如，如果您只想要前五行，则使用 LIMIT 子句的查询不会继续生成结果。这样，查询需要更少的内存和临时文件。

使用 GROUP BY 子句的查询也可能需要临时文件。GROUP BY 查询通过使用以下函数汇总值：

- COUNT
- AVG
- MIN
- MAX
- SUM
- STDDEV

要优化 GROUP BY 查询，请按照 ORDER BY 查询的建议。

避免使用 DISTINCT 操作

如果可能的话，避免使用 DISTINCT 操作来删除重复的行。查询返回的不必要和重复的行越多，DISTINCT 操作就会越昂贵。如果可能，请在 WHERE 子句中添加筛选条件，即使您对不同的表使用相同的筛选条件。筛选查询并正确联接可以提高性能并减少资源使用。它还可以防止错误的报告和结果。

如果您需要将 DISTINCT 用于同一个表的多行，请考虑创建复合索引。将索引中的多个列进行分组可以缩短评估不同行的时间。此外，如果您使用 Amazon Aurora PostgreSQL 版本 10 或更高版本，则可以使用 CREATE STATISTICS 命令在多个列之间关联统计数据。

考虑使用窗口函数而不是 GROUP BY 函数

使用 GROUP BY，您可以更改结果集，然后检索聚合的结果。使用窗口函数，可以在不更改结果集的情况下聚合数据。窗口函数使用 OVER 子句来跨查询定义的集执行计算，从而将一行与另一行关联。您可以使用窗口函数中的所有 GROUP BY 函数，但也可以使用以下函数：

- RANK
- ARRAY_AGG
- ROW_NUMBER
- LAG
- LEAD

为了尽量减少窗口函数生成的临时文件的数量，请在需要两个不同的聚合时删除同一结果集的重复项。请考虑以下查询。

```
SELECT sum(salary) OVER (PARTITION BY dept ORDER BY salary DESC) as sum_salary
     , avg(salary) OVER (PARTITION BY dept ORDER BY salary ASC) as avg_salary
  FROM empsalary;
```

您可以使用如下 WINDOW 子句重新写入查询。

```
SELECT sum(salary) OVER w as sum_salary
         , avg(salary) OVER w as_avg_salary
    FROM empsalary
  WINDOW w AS (PARTITION BY dept ORDER BY salary DESC);
```

预设情况下，Aurora PostgreSQL 执行计划器会整合类似的节点，这样它就不会重复操作。但是，通过对窗口数据块使用显式声明，您可以更轻松地维护查询。您还可以通过防止重复来提高性能。

调查具体化视图和 CTAS 语句

当具体化视图刷新时，它会运行查询。此查询可以包含 GROUP BY、ORDER BY 或 DISTINCT 之类的操作。刷新期间，您可能会观察到大量临时文件以及等待事件 IO:BufFileWrite 和

IO:BufFileRead。同样地，当您根据 SELECT 语句创建表时，CREATE TABLE 语句会运行查询。要减少所需的临时文件，请优化查询。

在创建索引时使用 pg_repack

创建索引时，引擎会对结果集进行排序。随着表的大小增加以及索引列中的值变得更加多样化，临时文件需要更多的空间。在大多数情况下，如果不修改维护工作内存区域，就无法阻止为大型表创建临时文件。有关更多信息，请参阅 维护工作内存区域。

重新创建大型索引时可能的解决方法是使用 pg_repack 工具。有关更多信息，请参阅 pg_repack 文档中的用最少的锁定重新组织 PostgreSQL 数据库中的表。

聚集表时，增加 maintenance_work_mem

CLUSTER 命令基于 index_name 指定的现有索引聚集 table_name 指定的表。Aurora PostgreSQL 以物理方式重新创建表以匹配给定索引的顺序。

当磁性存储普遍存在时，集群很常见，因为存储吞吐量有限。由于基于 SSD 的存储已经很常见，因此集群不太受欢迎。但是，如果对表进行聚集，您仍然可以根据表大小、索引、查询等稍微提高性能。

如果您运行 CLUSTER 命令并观察到等待事件 IO:BufFileWrite 和 IO:BufFileRead，请优化 maintenance_work_mem。将内存大小增加到相当大的量。较高的值意味着引擎可以使用更多内存进行集群操作。

优化内存以防止 IO:BufFileRead 和 IO:BufFileWrite

在某些情况下，您需要优化内存。您的目标是平衡以下要求：

- work_mem 值（请参阅 工作内存区域）
- 折扣 shared_buffers 值后剩余的内存（请参阅 缓冲池）
- 已打开和使用中的最大连接数，受限于 max_connections

增加工作内存区域的大小

在某些情况下，唯一的选项是增加会话使用的内存。如果您的查询编写正确并且正在使用正确的键进行连接，请考虑增加 work_mem 值。有关更多信息，请参阅 工作内存区域。

要了解查询生成了多少个临时文件，请将 log_temp_files 设置为 0。如果您将 work_mem 值增加为日志中标识的最大值，则可以防止查询生成临时文件。但是，work_mem 为每个连接或并行工件设

置每个计划节点的最大值。如果数据库有 5000 个连接，并且每个连接使用 256MiB 内存，则引擎需要 1.2TiB 的 RAM。因此，您的实例可能会耗尽内存。

为共享缓冲池预留足够的内存

您的数据库使用很多内存区域，例如共享缓冲池，而不仅仅是工作内存区域。在增加 work_mem 之前考虑这些额外的内存区域的要求。有关缓冲池的更多信息，请参阅 缓冲池。

例如，假设您的 Aurora PostgreSQL 实例类为 db.r5.2xlarge。此实例类拥有 64GiB 的内存。预设情况下，75% 的内存为共享缓冲池预留。减去分配给共享内存区域的量后，仍然有 16384 MB。不要将剩余内存专门分配给工作内存区域，因为操作系统和引擎还需要内存。

您可以分配给 work_mem 的内存取决于实例类。如果您使用较大的实例类，则可用的内存更多。但是，在前面的示例中，您不能使用超过 16GiB 的内存。否则，当内存耗尽时，您的实例将变得不可用。要从不可用状态恢复实例，Aurora PostgreSQL 自动化服务会自动重新启动。

管理连接数

假设您的数据库实例具有 5000 个同时连接。每个连接至少使用 4MiB 的 work_mem 连接的内存消耗过高可能会降低性能。作为响应，您可进行以下选择：

- 升级到更大的实例类。
- 使用连接代理或池程序减少同时数据库连接的数量。

对于代理，请考虑 Amazon RDS 代理、pgBouncer 或基于您的应用程序的连接池程序。此解决方案减轻了 CPU 负载。它还可以降低所有连接都需要工作内存区域时的风险。当数据库连接较少时，您可以增加 work_mem 的值。通过这种方式，您可以减少 IO:BufFileRead 和 IO:BufFileWrite 等待事件的发生率。此外，等待工作内存区域的查询显著加速。

IO:DataFileRead

当由于分页在共享内存中不可用，连接等待后端进程从存储中读取所需分页时，会发生 IO:DataFileRead 事件。

主题

- 支持的引擎版本
- 上下文
- 等待次数增加的可能原因
- 操作

支持的引擎版本

Aurora PostgreSQL 的所有版本均支持此等待事件信息。

上下文

所有查询和数据处理 (DML) 操作都会访问缓冲池中的页面。可以诱导读取的语句包括 SELECT、UPDATE 和 DELETE。例如，UPDATE 可以从表或索引中读取页面。如果请求或更新的页面不在共享缓冲池中，则此读取可能会导致 IO:DataFileRead 事件。

由于共享缓冲池是有限的，所以它可以填满。在这种情况下，对不在内存中的页面的请求会强制数据库从磁盘中读取数据块。如果 IO:DataFileRead 事件频繁发生，您的共享缓冲池可能太小，从而无法容纳您的工作负载。对于读取大量不适合缓冲池的行的 SELECT 查询，此问题很严重。有关缓冲池的更多信息，请参阅 缓冲池。

等待次数增加的可能原因

IO:DataFileRead 事件的常见原因包括以下各项：

连接激增

您可能会发现多个连接生成相同数量的 IO:DataFileRead 等待事件。在这种情况下，IO:DataFileRead 事件可能会发生激增（突然大幅度增加）。

执行顺序扫描的 SELECT 和 DML 语句

您的应用程序可能正在执行新的操作。或者，现有的操作可能会因为新的执行计划而发生变化。在这种情况下，请查找具有更大的 seq_scan 值的表格（特别是大型表格）。通过查询 pg_stat_user_tables 查找它们。要跟踪生成更多读取操作的查询，请使用扩展 pg_stat_statements。

适用于大型数据集的 CTAS 和 CREATE INDEX

CTAS 是一个 CREATE TABLE AS SELECT 语句。如果您使用大型数据集作为源来运行 CTAS，或者在大型表上创建索引，则可能会发生 IO:DataFileRead 事件。创建索引时，数据库可能需要使用顺序扫描读取整个对象。当页面不在内存中时，CTAS 会生成 IO:DataFile 读取。

多个 vacuum 工件同时运行

vacuum 工件可以手动或自动触发。我们建议采取积极的 vacuum 策略。但是，当表中有许多更新或删除的行时，IO:DataFileRead 等待增加。回收空间后，花在 IO:DataFileRead 上的 vacuum 时间减少。

摄取大量数据

当您的应用程序提取大量数据时，ANALYZE 操作可能会更频繁地发生。ANALYZE 进程可以由 Autovacuum 启动程序触发，也可以手动调用。

ANALYZE 操作可以读取表的子集。必须扫描的页数通过将 30 乘以 default_statistics_target 值进行计算。有关更多信息，请参阅 PostgreSQL 文档。default_statistics_target 参数接受 1 到 10000 之间的值，其中原定设置值为 100。

资源匮乏

如果消耗了实例网络带宽或 CPU，IO:DataFileRead 事件可能会更频繁地发生。

操作

根据等待事件的原因，我们建议采取不同的操作。

主题

- 检查谓词筛选条件是否存在生成等待的查询
- 尽量减少维护操作的影响
- 响应大量连接

检查谓词筛选条件是否存在生成等待的查询

假设您确定了正在生成 IO:DataFileRead 等待事件的特定查询。您可以使用以下方法识别它们：

- 性能详情
- 目录视图，例如扩展程序 pg_stat_statements 提供的视图
- 目录视图 pg_stat_all_tables，如果它定期显示物理读取数量增加
- pg_statio_all_tables 视图，如果它显示 _read 计数器正在增加

我们建议您确定这些查询的谓词（WHERE 子句）中使用了哪些筛选条件。请遵循以下准则：

- 运行 EXPLAIN 命令。在输出中，确定使用的扫描类型。顺序扫描不一定表示存在问题。与使用筛选条件的查询相比，使用顺序扫描的查询自然会产生更多的 IO:DataFileRead 事件。

了解 WHERE 子句中列出的列是否已编入索引。如果没有，请考虑为此列创建索引。这种方法避免了顺序扫描并减少了 IO:DataFileRead 事件。如果某个查询具有限制性筛选条件并且仍然生成顺序扫描，请评估是否使用了正确的索引。

- 了解查询是否正在访问非常大的表。在某些情况下，对表进行分区可以提高性能，从而允许查询只读取必要的分区。
- 检查联接操作的基数（总行数）。请注意您在筛选条件中为您的 WHERE 子句传递的值的限制性。如果可能，请优化查询以减少在计划的每个步骤中传递的行数。

尽量减少维护操作的影响

维护操作（例如 VACUUM 和 ANALYZE）非常重要。我们建议您不要将其关闭，因为您会找到与这些维护操作相关的 IO:DataFileRead 等待事件。以下方法可以最大限度地减少这些操作的影响：

- 在非高峰时段手动运行维护操作。此方法可防止数据库达到自动操作的阈值。
- 对于非常大的表，请考虑对表进行分区。这种方法减少了维护操作的开销。数据库只访问需要维护的分区。
- 当您摄取大量数据时，请考虑禁用自动分析功能。

当以下公式为真时，系统会自动为表触发 Autovacuum 功能。

```
pg_stat_user_tables.n_dead_tup > (pg_class.reltuples x autovacuum_vacuum_scale_factor)
 + autovacuum_vacuum_threshold
```

视图 pg_stat_user_tables 和目录 pg_class 有多个行。一行可以对应于表中的一行。这个公式假设 reltuples 适用于特定的表。参数 autovacuum_vacuum_scale_factor（原定设置为 0.20）和 autovacuum_vacuum_threshold（原定设置为 50 个元组）通常在全局范围内为整个实例设置。但是，您可以为特定表设置不同的值。

主题

- 查找不必要地占用空间的表
- 查找占用不必要空间的索引
- 查找符合 Autovacuum 操作条件的表

查找不必要地占用空间的表

要查找占用超出所需空间的表，请运行以下查询。当此查询由不具有 rds_superuser 角色的数据库用户角色运行时，它只返回有关用户角色有权读取的那些表的信息。PostgreSQL 版本 12 及更高版本支持此查询。

```
WITH report AS (
   SELECT   schemaname
           ,tblname
           ,n_dead_tup
           ,n_live_tup
           ,block_size*tblpages AS real_size
           ,(tblpages-est_tblpages)*block_size AS extra_size
           ,CASE WHEN tblpages - est_tblpages > 0
              THEN 100 * (tblpages - est_tblpages)/tblpages::float
              ELSE 0
            END AS extra_ratio, fillfactor, (tblpages-est_tblpages_ff)*block_size AS
 bloat_size
           ,CASE WHEN tblpages - est_tblpages_ff > 0
              THEN 100 * (tblpages - est_tblpages_ff)/tblpages::float
              ELSE 0
            END AS bloat_ratio
           ,is_na
    FROM (
           SELECT  ceil( reltuples / ( (block_size-page_hdr)/tpl_size ) ) +
 ceil( toasttuples / 4 ) AS est_tblpages
                  ,ceil( reltuples / ( (block_size-page_hdr)*fillfactor/
(tpl_size*100) ) ) + ceil( toasttuples / 4 ) AS est_tblpages_ff
                  ,tblpages
                  ,fillfactor
                  ,block_size
                  ,tblid
                  ,schemaname
                  ,tblname
                  ,n_dead_tup
                  ,n_live_tup
                  ,heappages
                  ,toastpages
                  ,is_na
             FROM (
                    SELECT ( 4 + tpl_hdr_size + tpl_data_size + (2*ma)
                              - CASE WHEN tpl_hdr_size%ma = 0 THEN ma ELSE
 tpl_hdr_size%ma END
```

```
                                - CASE WHEN ceil(tpl_data_size)::int%ma = 0 THEN ma ELSE
 ceil(tpl_data_size)::int%ma END
                            ) AS tpl_size
                            ,block_size - page_hdr AS size_per_block
                            ,(heappages + toastpages) AS tblpages
                            ,heappages
                            ,toastpages
                            ,reltuples
                            ,toasttuples
                            ,block_size
                            ,page_hdr
                            ,tblid
                            ,schemaname
                            ,tblname
                            ,fillfactor
                            ,is_na
                            ,n_dead_tup
                            ,n_live_tup
                           FROM (
                                SELECT  tbl.oid                       AS tblid
                                       ,ns.nspname                    AS schemaname
                                       ,tbl.relname                   AS tblname
                                       ,tbl.reltuples                 AS reltuples
                                       ,tbl.relpages                  AS heappages
                                       ,coalesce(toast.relpages, 0)   AS toastpages
                                       ,coalesce(toast.reltuples, 0)  AS toasttuples
                                       ,psat.n_dead_tup               AS n_dead_tup
                                       ,psat.n_live_tup               AS n_live_tup
                                       ,24                            AS page_hdr
                                       ,current_setting('block_size')::numeric AS
 block_size

 ,coalesce(substring( array_to_string(tbl.reloptions, ' ') FROM
 'fillfactor=([0-9]+)')::smallint, 100) AS fillfactor
                                       ,CASE WHEN version()~'mingw32' OR version()~'64-
bit|x86_64|ppc64|ia64|amd64' THEN 8 ELSE 4 END        AS ma
                                       ,23 + CASE WHEN MAX(coalesce(null_frac,0)) > 0
 THEN ( 7 + count(*) ) / 8 ELSE 0::int END              AS tpl_hdr_size
                                       ,sum( (1-coalesce(s.null_frac, 0)) *
 coalesce(s.avg_width, 1024) )                                  AS tpl_data_size
                                       ,bool_or(att.atttypid =
 'pg_catalog.name'::regtype) OR count(att.attname) <> count(s.attname)        AS is_na
                                  FROM  pg_attribute       AS att
```

```
                                    JOIN  pg_class           AS tbl    ON (att.attrelid =
 tbl.oid)
                                    JOIN  pg_stat_all_tables AS psat   ON (tbl.oid =
 psat.relid)
                                    JOIN  pg_namespace       AS ns     ON (ns.oid =
 tbl.relnamespace)
                               LEFT JOIN  pg_stats           AS s      ON
 (s.schemaname=ns.nspname AND s.tablename = tbl.relname AND s.inherited=false AND
 s.attname=att.attname)
                               LEFT JOIN  pg_class           AS toast  ON
 (tbl.reltoastrelid = toast.oid)
                                   WHERE  att.attnum > 0
                                     AND  NOT att.attisdropped
                                     AND  tbl.relkind = 'r'
                                GROUP BY  tbl.oid, ns.nspname, tbl.relname,
 tbl.reltuples, tbl.relpages, toastpages, toasttuples, fillfactor, block_size, ma,
 n_dead_tup, n_live_tup
                                ORDER BY  schemaname, tblname
                              ) AS s
                  ) AS s2
       ) AS s3
 ORDER BY bloat_size DESC
)
  SELECT *
    FROM report
   WHERE bloat_ratio != 0
 -- AND schemaname = 'public'
 -- AND tblname = 'pgbench_accounts'
;

-- WHERE NOT is_na
--   AND tblpages*((pst).free_percent + (pst).dead_tuple_percent)::float4/100 >= 1
```

您可以在应用程序中检查表和索引膨胀。有关更多信息，请参阅 诊断表和索引膨胀。

查找占用不必要空间的索引

要查找占用不必要空间的索引，请运行以下查询。

```
-- WARNING: run with a nonsuperuser role, the query inspects
-- only indexes on tables you have permissions to read.
-- WARNING: rows with is_na = 't' are known to have bad statistics ("name" type is not
 supported).
-- This query is compatible with PostgreSQL 8.2 and later.
```

```
SELECT current_database(), nspname AS schemaname, tblname, idxname,
 bs*(relpages)::bigint AS real_size,
  bs*(relpages-est_pages)::bigint AS extra_size,
  100 * (relpages-est_pages)::float / relpages AS extra_ratio,
  fillfactor, bs*(relpages-est_pages_ff) AS bloat_size,
  100 * (relpages-est_pages_ff)::float / relpages AS bloat_ratio,
  is_na
  -- , 100-(sub.pst).avg_leaf_density, est_pages, index_tuple_hdr_bm,
  -- maxalign, pagehdr, nulldatawidth, nulldatahdrwidth, sub.reltuples, sub.relpages
  -- (DEBUG INFO)
FROM (
  SELECT coalesce(1 +
       ceil(reltuples/floor((bs-pageopqdata-pagehdr)/(4+nulldatahdrwidth)::float)), 0
       -- ItemIdData size + computed avg size of a tuple (nulldatahdrwidth)
    ) AS est_pages,
    coalesce(1 +
       ceil(reltuples/floor((bs-pageopqdata-pagehdr)*fillfactor/
(100*(4+nulldatahdrwidth)::float))), 0
    ) AS est_pages_ff,
    bs, nspname, table_oid, tblname, idxname, relpages, fillfactor, is_na
    -- , stattuple.pgstatindex(quote_ident(nspname)||'.'||quote_ident(idxname)) AS
 pst,
    -- index_tuple_hdr_bm, maxalign, pagehdr, nulldatawidth, nulldatahdrwidth,
 reltuples
    -- (DEBUG INFO)
  FROM (
    SELECT maxalign, bs, nspname, tblname, idxname, reltuples, relpages, relam,
 table_oid, fillfactor,
      ( index_tuple_hdr_bm +
          maxalign - CASE -- Add padding to the index tuple header to align on MAXALIGN
            WHEN index_tuple_hdr_bm%maxalign = 0 THEN maxalign
            ELSE index_tuple_hdr_bm%maxalign
          END
        + nulldatawidth + maxalign - CASE -- Add padding to the data to align on
 MAXALIGN
            WHEN nulldatawidth = 0 THEN 0
            WHEN nulldatawidth::integer%maxalign = 0 THEN maxalign
            ELSE nulldatawidth::integer%maxalign
          END
      )::numeric AS nulldatahdrwidth, pagehdr, pageopqdata, is_na
      -- , index_tuple_hdr_bm, nulldatawidth -- (DEBUG INFO)
    FROM (
      SELECT
```

```
       i.nspname, i.tblname, i.idxname, i.reltuples, i.relpages, i.relam, a.attrelid
AS table_oid,
       current_setting('block_size')::numeric AS bs, fillfactor,
       CASE -- MAXALIGN: 4 on 32bits, 8 on 64bits (and mingw32 ?)
         WHEN version() ~ 'mingw32' OR version() ~ '64-bit|x86_64|ppc64|ia64|amd64'
THEN 8
         ELSE 4
       END AS maxalign,
       /* per page header, fixed size: 20 for 7.X, 24 for others */
       24 AS pagehdr,
       /* per page btree opaque data */
       16 AS pageopqdata,
       /* per tuple header: add IndexAttributeBitMapData if some cols are null-able */
       CASE WHEN max(coalesce(s.null_frac,0)) = 0
         THEN 2 -- IndexTupleData size
         ELSE 2 + (( 32 + 8 - 1 ) / 8)
         -- IndexTupleData size + IndexAttributeBitMapData size ( max num filed per
index + 8 - 1 /8)
       END AS index_tuple_hdr_bm,
       /* data len: we remove null values save space using it fractionnal part from
stats */
       sum( (1-coalesce(s.null_frac, 0)) * coalesce(s.avg_width, 1024)) AS
nulldatawidth,
       max( CASE WHEN a.atttypid = 'pg_catalog.name'::regtype THEN 1 ELSE 0 END ) > 0
AS is_na
     FROM pg_attribute AS a
       JOIN (
         SELECT nspname, tbl.relname AS tblname, idx.relname AS idxname,
           idx.reltuples, idx.relpages, idx.relam,
           indrelid, indexrelid, indkey::smallint[] AS attnum,
           coalesce(substring(
             array_to_string(idx.reloptions, ' ')
              from 'fillfactor=([0-9]+)')::smallint, 90) AS fillfactor
         FROM pg_index
           JOIN pg_class idx ON idx.oid=pg_index.indexrelid
           JOIN pg_class tbl ON tbl.oid=pg_index.indrelid
           JOIN pg_namespace ON pg_namespace.oid = idx.relnamespace
         WHERE pg_index.indisvalid AND tbl.relkind = 'r' AND idx.relpages > 0
       ) AS i ON a.attrelid = i.indexrelid
       JOIN pg_stats AS s ON s.schemaname = i.nspname
         AND ((s.tablename = i.tblname AND s.attname =
pg_catalog.pg_get_indexdef(a.attrelid, a.attnum, TRUE))
         -- stats from tbl
         OR  (s.tablename = i.idxname AND s.attname = a.attname))
```

```
          -- stats from functionnal cols
        JOIN pg_type AS t ON a.atttypid = t.oid
      WHERE a.attnum > 0
      GROUP BY 1, 2, 3, 4, 5, 6, 7, 8, 9
    ) AS s1
  ) AS s2
    JOIN pg_am am ON s2.relam = am.oid WHERE am.amname = 'btree'
) AS sub
-- WHERE NOT is_na
ORDER BY 2,3,4;
```

查找符合 Autovacuum 操作条件的表

要查找符合 Autovacuum 操作条件的表，请运行以下查询。

```
--This query shows tables that need vacuuming and are eligible candidates.
--The following query lists all tables that are due to be processed by autovacuum.
-- During normal operation, this query should return very little.
WITH  vbt AS (SELECT setting AS autovacuum_vacuum_threshold
              FROM pg_settings WHERE name = 'autovacuum_vacuum_threshold')
    , vsf AS (SELECT setting AS autovacuum_vacuum_scale_factor
              FROM pg_settings WHERE name = 'autovacuum_vacuum_scale_factor')
    , fma AS (SELECT setting AS autovacuum_freeze_max_age
              FROM pg_settings WHERE name = 'autovacuum_freeze_max_age')
    , sto AS (SELECT opt_oid, split_part(setting, '=', 1) as param,
                split_part(setting, '=', 2) as value
              FROM (SELECT oid opt_oid, unnest(reloptions) setting FROM pg_class) opt)
SELECT
    '"'||ns.nspname||'"."'||c.relname||'"' as relation
    , pg_size_pretty(pg_table_size(c.oid)) as table_size
    , age(relfrozenxid) as xid_age
    , coalesce(cfma.value::float, autovacuum_freeze_max_age::float)
 autovacuum_freeze_max_age
    , (coalesce(cvbt.value::float, autovacuum_vacuum_threshold::float) +
         coalesce(cvsf.value::float,autovacuum_vacuum_scale_factor::float) *
 c.reltuples)
         as autovacuum_vacuum_tuples
    , n_dead_tup as dead_tuples
FROM pg_class c
JOIN pg_namespace ns ON ns.oid = c.relnamespace
JOIN pg_stat_all_tables stat ON stat.relid = c.oid
JOIN vbt on (1=1)
JOIN vsf ON (1=1)
JOIN fma on (1=1)
```

```
LEFT JOIN sto cvbt ON cvbt.param = 'autovacuum_vacuum_threshold' AND c.oid =
 cvbt.opt_oid
LEFT JOIN sto cvsf ON cvsf.param = 'autovacuum_vacuum_scale_factor' AND c.oid =
 cvsf.opt_oid
LEFT JOIN sto cfma ON cfma.param = 'autovacuum_freeze_max_age' AND c.oid = cfma.opt_oid
WHERE c.relkind = 'r'
AND nspname <> 'pg_catalog'
AND (
    age(relfrozenxid) >= coalesce(cfma.value::float, autovacuum_freeze_max_age::float)
    or
    coalesce(cvbt.value::float, autovacuum_vacuum_threshold::float) +
      coalesce(cvsf.value::float,autovacuum_vacuum_scale_factor::float) * c.reltuples
 <= n_dead_tup
    -- or 1 = 1
)
ORDER BY age(relfrozenxid) DESC;
```

响应大量连接

当您监控 Amazon CloudWatch 时，您可能会发现 `DatabaseConnections` 指标激增。这种增加表示与数据库的连接数量有所增加。我们建议采取以下方法：

- 限制应用程序可与每个实例一起打开的连接数。如果您的应用程序具有嵌入式连接池功能，请设置合理数量的连接。根据实例中的 vCPU 可以有效并行处理的数量来确定数量。

  如果您的应用程序没有使用连接池功能，请考虑使用 Amazon RDS 代理或替代方案。这种方法允许您的应用程序打开与负载均衡器的多个连接。然后，均衡器可以打开与数据库的数量有限的连接。由于并行运行的连接减少，您的数据库实例在内核中执行的上下文切换会减少。查询的进度应该更快，从而导致等待事件减少。有关更多信息，请参阅 适用于 Aurora 的 Amazon RDS 代理。
- 尽可能利用 Aurora PostgreSQL 的读取器节点和 RDS for PostgreSQL 的只读副本。当您的应用程序运行只读操作时，将这些请求发送到只读端点。此方法将应用程序请求分布到所有读取器节点，从而减少了写入器节点的输入/输出压力。
- 请考虑纵向扩展数据库实例。更高容量的实例类可提供更多内存，这为 Aurora PostgreSQL 提供了一个更大的共享缓冲池来容纳页面。较大的大小还为数据库实例提供了更多的 vCPU 来处理连接。当生成 `IO:DataFileRead` 等待事件的操作为写入时，更多的 vCPU 会特别有用。

IO:XactSync

当数据库等待 Aurora 存储子系统确认常规事务的提交，或者准备好的事务的提交或回滚时，会发生 `IO:XactSync` 事件。准备好的事务是 PostgreSQL 支持两阶段提交的一部分。当一个查询正在等待另

一个事务提交时，也可能发生此事件，尤其是在关闭自动提交的情况下。在此类情况下，即使更新尚未提交，XactSync 上也可能看起来正在等待更新。

主题

- 支持的引擎版本
- 上下文
- 等待次数增加的可能原因
- 操作

支持的引擎版本

Aurora PostgreSQL 的所有版本均支持此等待事件信息。

上下文

事件 `IO:XactSync` 表示实例在花时间等待 Aurora 存储子系统确认事务数据已处理。

等待次数增加的可能原因

当 `IO:XactSync` 事件的发生率超过正常（可能表示性能问题）时，典型原因包括以下几点：

网络饱和

客户端与数据库实例之间的流量或至存储子系统的流量对于网络带宽来说可能太大。

CPU 压力

繁重的工作负载可能会阻止 Aurora 存储守护进程获得足够的 CPU 时间。

操作

根据等待事件的原因，我们建议采取不同的操作。

主题

- 监控您的资源。
- 纵向扩展 CPU
- 提高网络带宽
- 减少提交的数量

监控您的资源。

要确定 IO:XactSync 事件增加的原因，请检查以下指标：

- WriteThroughput 和 CommitThroughput – 写入吞吐量或提交吞吐量的变化可能显示工作负载增加。
- WriteLatency 和 CommitLatency – 写入延迟或提交延迟的变化可能表明系统要求存储子系统执行更多工作。
- CPUUtilization – 如果实例的 CPU 利用率超过 90%，Aurora 存储守护进程可能没有足够的 CPU 时间。在这种情况下，输入/输出性能会下降。

有关这些指标的信息，请参阅 Amazon Aurora 的实例级指标。

纵向扩展 CPU

要解决 CPU 匮乏问题，请考虑更改为具有更多 CPU 容量的实例类型。有关数据库实例类的 CPU 容量的信息，请参阅 适用于 Aurora 的数据库实例类的硬件规格。

提高网络带宽

要确定实例是否达到其网络带宽限制，请检查以下其他等待事件：

- IO:DataFileRead、IO:BufferRead、IO:BufferWrite 和 IO:XactWrite – 使用大量输入/输出的查询可以生成更多这类等待事件。
- Client:ClientRead 和 Client:ClientWrite – 具有大量客户端通信的查询可以生成更多这类等待事件。

如果网络带宽是一个问题，请考虑更改为具有更多网络带宽的实例类型。有关数据库实例类的网络性能的信息，请参阅 适用于 Aurora 的数据库实例类的硬件规格。

减少提交的数量

为了减少提交的数量，请将语句合并到事务数据块中。

IPC:DamRecordTxAck

当 Aurora PostgreSQL 在使用数据库活动流的会话中生成活动流事件，然后等待该事件变为持久事件时，会发生 IPC:DamRecordTxAck 事件。

主题

- 相关引擎版本
- 上下文
- 原因
- 操作

相关引擎版本

此等待事件信息与所有的 Aurora PostgreSQL 10.7 及更高的 10.x 版本、11.4 及更高的 11.x 版本以及所有的 12.x 和 13.x 版本相关。

上下文

在同步模式下，活动流事件的持久性优于数据库性能。在等待持久写入事件时，会话会阻止其他数据库活动，从而导致 `IPC:DamRecordTxAck` 等待事件。

原因

显示在主要等待中的 `IPC:DamRecordTxAck` 事件的最常见原因是数据库活动流 (DAS) 功能是一项全面审计。较高的 SQL 活动会生成需要记录的活动流事件。

操作

根据等待事件的原因，我们建议采取不同的操作：

- 减少 SQL 语句的数量或关闭数据库活动流。这样可以减少需要持久写入的事件数量。
- 更改为异步模式。这样做有助于减少对 `IPC:DamRecordTxAck` 等待事件的争用。

  但是，DAS 功能无法保证每个事件在异步模式下的持久性。

IPC:并行等待事件

以下 `IPC:parallel wait events` 表明会话正在等待与并行查询执行操作相关的进程间通信。

- `IPC:BgWorkerStartup`：进程正在等待并行工作进程完成其启动序列。在初始化工作线程以执行并行查询时会发生这种情况。
- `IPC:BgWorkerShutdown`：进程正在等待并行工作进程完成其关闭序列。这发生在并行查询执行的清理阶段。

- `IPC:ExecuteGather`：在查询执行期间，进程正在等待从并行工作进程接收数据。当领导进程需要从其工作线程中收集结果时，会发生这种情况。
- `IPC:ParallelFinish`：进程正在等待并行工作线程完成其执行并报告其最终结果。这种情况发生在并行查询执行的完成阶段。

主题

- 支持的引擎版本
- 上下文
- 等待次数增加的可能原因
- 操作

支持的引擎版本

Aurora PostgreSQL 的所有版本均支持此等待事件信息。

上下文

PostgreSQL 中的并行查询执行涉及多个进程协同工作以处理单个查询。当确定查询适合并行化时，领导进程会根据 `max_parallel_workers_per_gather` 参数设置与一个或多个并行工作进程进行协调。领导进程在各工作线程之间划分工作，每个工作线程独立处理自己的那部分数据，然后将结果收集回领导进程。

> **Note**
>
> 每个并行工作线程均作为一个单独的进程运行，其资源需求类似于完整的用户会话。这意味着，与非并行查询相比，具有 4 个工作线程的并行查询消耗的资源（CPU、内存、I/O 带宽）最多可达 5 倍，因为领导进程和每个工作进程都保持自己的资源分配。例如，诸如 `work_mem` 的设置单独应用于每个工作线程，可能会使所有进程的总内存使用量成倍增加。

并行查询架构由三个主要部分组成：

- 领导进程：启动并行操作、划分工作负载并与工作进程进行协调的主进程。
- 工作进程：并行执行查询各部分的后台进程。
- 收集/收集合并：将来自多个工作进程的结果合并回领导进程的操作

在并行执行期间，进程需要通过进程间通信（IPC）机制相互通信。这些 IPC 等待事件发生在不同的阶段：

- 工作线程启动：初始化并行工作线程时
- 数据交换：当工作线程处理数据并将结果发送给领导进程时
- 工作线程关闭：当并行执行完成且终止工作线程时
- 同步点：当进程需要协调或等待其它进程完成其任务时

了解这些等待事件对于诊断与并行查询执行相关的性能问题至关重要，尤其是在可能同时执行多个并行查询的高并发环境中。

等待次数增加的可能原因

有几个因素可能导致与并行相关的 IPC 等待事件增加：

并行查询的高并发性

当多个并行查询同时运行时，可能会导致资源争用和 IPC 操作的等待时间增加。这在事务量或分析工作负载较高的系统中尤其常见。

并行查询计划不够理想

如果查询计划器选择效率低下的并行计划，则可能会导致不必要的并行化或工作线程间的工作分配不佳。这可能会导致 IPC 等待增加，尤其是对于 IPC:ExecuteGather 和 IPC:ParallelFinish 事件。这些规划问题通常源于过时的统计数据和表/索引膨胀。

并行工作线程频繁启动和关闭

频繁启动和终止并行工作线程的短期查询可能会导致 `IPC:BgWorkerStartup` 和 `IPC:BgWorkerShutdown` 事件增加。这种情况经常出现在具有许多小型可并行查询的 OLTP 工作负载中。

资源约束

有限的 CPU、内存或 I/O 容量可能会导致并行执行出现瓶颈，从而导致所有 IPC 事件的等待时间增加。例如，如果 CPU 饱和，则工作进程可能需要更长的时间才能启动或处理自己的那部分工作。

复杂的查询结构

具有多个并行性级别的查询（例如，并行联接后跟并行聚合）可能会导致更复杂的 IPC 模式并可能增加等待时间，尤其是对于 `IPC:ExecuteGather` 事件。

大型结果集

生成大型结果集的查询可能会导致 IPC:ExecuteGather 等待时间增加，因为领导进程将花费更多的时间来收集和处理来自工作进程的结果。

了解这些因素有助于诊断和解决与 Aurora PostgreSQL 中的并行查询执行相关的性能问题。

操作

当您看到与并行查询相关的等待时，这通常表示后端进程正在协调或等待并行工作进程。在执行并行计划期间，这些等待很常见。您可以通过监控并行工作线程使用情况、查看参数设置以及调整查询执行和资源分配，来调查并缓解这些等待的影响。

主题

- 分析查询计划是否存在并行性效率低下问题
- 监控并行查询使用情况
- 查看和调整并行查询设置
- 优化资源分配
- 调查连接管理
- 查看和优化维护操作
- 利用查询计划管理（QPM）

分析查询计划是否存在并行性效率低下问题

并行查询执行可能经常导致系统不稳定、CPU 峰值和不可预测的查询性能变化。彻底分析并行性是否确实可以改善特定工作负载至关重要。使用 EXPLAIN ANALYZE 查看并行查询执行计划。

在会话级别暂时禁用并行性以比较计划效率：

```
SET max_parallel_workers_per_gather = 0;
EXPLAIN ANALYZE <your_query>;
```

重新启用并行性并进行比较：

```
RESET max_parallel_workers_per_gather;
```

```
EXPLAIN ANALYZE <your_query>;
```

如果禁用并行性会产生更好或更一致的结果，则考虑使用 SET 命令在会话级别为特定查询禁用并行性。为了获得更广泛的影响，您可能需要通过调整集群或实例参数组中的相关参数，在实例级别禁用并行性。有关更多信息，请参阅Amazon Aurora PostgreSQL 参数。

监控并行查询使用情况

使用以下查询来深入了解并行查询活动和容量：

检查活动的并行工作进程：

```
SELECT
    COUNT(*)
FROM
    pg_stat_activity
WHERE
    backend_type = 'parallel worker';
```

此查询显示活动的并行工作进程的数量。较高的值可能表示为“max_parallel_workers”配置了较高的值，您可能需要考虑将其降低。

检查并发并行查询：

```
SELECT
    COUNT(DISTINCT leader_pid)
FROM
    pg_stat_activity
WHERE
    leader_pid IS NOT NULL;
```

此查询返回已启动并行查询的不同领导进程的数量。此处的数字较高表示多个会话正在并发运行并行查询，这可能会增加对 CPU 和内存的需求。

查看和调整并行查询设置

查看以下参数以确保它们与您的工作负载一致：

- max_parallel_workers：所有会话中并行工作线程的总数。
- max_parallel_workers_per_gather：每个查询的最大工作线程数。

对于 OLAP 工作负载，增加这些值可以提高性能。对于 OLTP 工作负载，通常首选较低的值。

```
SHOW max_parallel_workers;
SHOW max_parallel_workers_per_gather;
```

优化资源分配

监控 CPU 利用率，如果 vCPU 的数量一直很高，并且应用程序从并行查询中受益，请考虑调整 vCPU 的数量。确保有足够的内存可用于并行操作。

- 使用性能详情指标来确定系统是否受到 CPU 限制。
- 每个并行工作线程都使用自己的 work_mem。确保总内存使用量在实例限制范围内。

并行查询可能比非并行查询消耗的资源要多得多，因为每个工作进程都是一个完全独立的进程，它对系统的影响与额外的用户会话大致相同。在为此设置选择值时，以及在配置其它用于控制资源利用率的设置（例如 work_mem）时，应考虑到这一点。有关更多信息，请参阅 PostgreSQL 文档。资源限制（例如 work_mem）单独应用于每个工作线程，这意味着跨所有进程的总利用率可能比针对任何单个进程的正常利用率高得多。

如果工作负载高度并行化，可以考虑增加 vCPU 数量或调整内存参数。

调查连接管理

如果遇到连接耗尽的情况，请查看应用程序连接池策略。考虑在应用程序级别实现连接池（如果尚未使用）。

查看和优化维护操作

协调索引创建和其它维护任务，以防止资源争用。考虑将这些操作安排在非高峰时段。避免在用户查询负载较高的时期安排繁重的维护工作（例如并行索引构建）。这些操作可能会消耗并行工作线程并影响常规查询的性能。

利用查询计划管理（QPM）

在 Aurora PostgreSQL 中，查询计划管理（QPM）功能旨在确保计划的适应性和稳定性，而无论数据库环境如何变化（此类变化可能导致查询计划回归）。有关更多信息，请参阅Aurora PostgreSQL 查询计划管理概览。QPM 对优化器提供某种程度的控制。在 QPM 中查看已批准的计划，以确保它们与当前的并行性设置保持一致。更新或移除过时的计划，这些计划可能会强制执行不够理想的并行执行。

也可以使用 pg_hint_plan 修复计划。有关更多信息，请参阅使用 pg_hint_plan 修复计划。可以使用名为 `Parallel` 的提示来强制并行执行。有关更多信息，请参阅 Hints for parallel plans。

Lock:advisory

当 PostgreSQL 应用程序使用锁定来协调多个会话之间的活动时，会发生 `Lock:advisory` 事件。

主题

- 相关引擎版本
- 上下文
- 原因
- 操作

相关引擎版本

此等待事件信息与 Aurora PostgreSQL 版本 9.6 及更高版本相关。

上下文

PostgreSQL 咨询锁是由用户的应用程序代码显式锁定和解锁的应用程序级别的合作锁。应用程序可以使用 PostgreSQL 咨询锁来协调多个会话之间的活动时。与常规锁、对象级锁或行级锁不同的是，应用程序对锁的生命周期拥有完全控制权。有关更多信息，请参阅 PostgreSQL 文档中的咨询锁。

咨询锁可以在事务结束之前释放，也可以在事务中由会话持有。对于隐式的、系统强制的锁，例如 `CREATE INDEX` 语句获取的对表的互斥访问锁，情况并非如此。

有关用于获取（锁定）和释放（解锁）咨询锁的函数的说明，请参阅 PostgreSQL 文档中的咨询锁函数。

咨询锁是在常规的 PostgreSQL 锁定系统之上实施的，并且在 `pg_locks` 系统视图中可见。

原因

这种锁类型由显式使用它的应用程序完全控制。作为查询的一部分为每个行获取的咨询锁可能会导致锁激增或长期积累。

当以获得比查询返回的行更多的锁的方式运行查询时，会发生这些影响。应用程序最终必须释放每个锁，但是如果在未返回的行上获取锁，则应用程序无法找到所有锁。

以下示例来自 PostgreSQL 文档中的咨询锁。

```
SELECT pg_advisory_lock(id) FROM foo WHERE id > 12345 LIMIT 100;
```

在此示例中，LIMIT 子句只能在内部选择行并锁定其 ID 值后停止查询的输出。当数据量不断增长导致计划人员选择在开发过程中未测试的其他执行计划时，可能会突然发生这种情况。在这种情况下，由于应用程序会为锁定的每个 ID 值调用 pg_advisory_unlock，会发生累积。但是，在这种情况下，它找不到在未返回的行上获取的锁集合。由于锁是在会话级别获取的，因此它们不会在事务结束时自动释放。

阻止锁定尝试激增的另一个可能原因是意外的冲突。在这些冲突中，应用程序的不相关部分错误地共享了相同的锁 ID 空间。

操作

查看应用程序对咨询锁的使用情况，并详细说明在应用程序流中获取和释放每种类型的咨询锁的位置和时间。

确定会话是获取太多锁定还是长时间运行的会话没有尽早释放锁，从而导致锁缓慢累积。您可以通过使用 pg_terminate_backend(pid) 结束会话来纠正会话级别锁定的缓慢累积。

正在等待咨询锁的客户端显示在带有 wait_event_type=Lock 和 wait_event=advisory 的 pg_stat_activity 中。您可以通过查询相同 pid 的 pg_locks 系统视图来获取特定的锁定值，以寻找 locktype=advisory 和 granted=f。

然后您可以通过查询具有 granted=t 的相同咨询锁的 pg_locks 来识别阻止的会话，如以下示例所示。

```
SELECT blocked_locks.pid AS blocked_pid,
         blocking_locks.pid AS blocking_pid,
         blocked_activity.usename AS blocked_user,
```

```
        blocking_activity.usename AS blocking_user,
        now() - blocked_activity.xact_start AS blocked_transaction_duration,
        now() - blocking_activity.xact_start AS blocking_transaction_duration,
        concat(blocked_activity.wait_event_type,':',blocked_activity.wait_event) AS
blocked_wait_event,
        concat(blocking_activity.wait_event_type,':',blocking_activity.wait_event) AS
blocking_wait_event,
        blocked_activity.state AS blocked_state,
        blocking_activity.state AS blocking_state,
        blocked_locks.locktype AS blocked_locktype,
        blocking_locks.locktype AS blocking_locktype,
        blocked_activity.query AS blocked_statement,
        blocking_activity.query AS blocking_statement
   FROM pg_catalog.pg_locks blocked_locks
   JOIN pg_catalog.pg_stat_activity blocked_activity ON blocked_activity.pid =
blocked_locks.pid
   JOIN pg_catalog.pg_locks blocking_locks
       ON blocking_locks.locktype = blocked_locks.locktype
       AND blocking_locks.DATABASE IS NOT DISTINCT FROM blocked_locks.DATABASE
       AND blocking_locks.relation IS NOT DISTINCT FROM blocked_locks.relation
       AND blocking_locks.page IS NOT DISTINCT FROM blocked_locks.page
       AND blocking_locks.tuple IS NOT DISTINCT FROM blocked_locks.tuple
       AND blocking_locks.virtualxid IS NOT DISTINCT FROM blocked_locks.virtualxid
       AND blocking_locks.transactionid IS NOT DISTINCT FROM
blocked_locks.transactionid
       AND blocking_locks.classid IS NOT DISTINCT FROM blocked_locks.classid
       AND blocking_locks.objid IS NOT DISTINCT FROM blocked_locks.objid
       AND blocking_locks.objsubid IS NOT DISTINCT FROM blocked_locks.objsubid
       AND blocking_locks.pid != blocked_locks.pid
   JOIN pg_catalog.pg_stat_activity blocking_activity ON blocking_activity.pid =
blocking_locks.pid
   WHERE NOT blocked_locks.GRANTED;
```

所有的咨询锁 API 函数都有两组参数，一个 bigint 参数或两个 integer 参数：

- 对于具有一个 bigint 参数的 API 函数，上面的 32 位在 pg_locks.classid 中，下面的 32 位在 pg_locks.objid 中。
- 对于具有两个 integer 参数的 API 函数，第一个参数是 pg_locks.classid，第二个参数是 pg_locks.objid。

pg_locks.objsubid 值表示使用了哪个 API 表单：1 表示一个 bigint 参数；2 表示两个 integer 参数。

Lock:extend

当后端进程正在等待锁定关系以对其进行扩展，而另一个进程出于同样目的锁定该关系时，会发生 Lock:extend 事件。

主题

- 支持的引擎版本
- 上下文
- 等待次数增加的可能原因
- 操作

支持的引擎版本

Aurora PostgreSQL 的所有版本均支持此等待事件信息。

上下文

事件 Lock:extend 表示后端进程正在等待扩展另一个后端进程在扩展该关系时保持锁定的关系。由于每次只有一个进程可以扩展关系，因此系统会生成 Lock:extend 等待事件。INSERT、COPY 和 UPDATE 操作可以生成此事件。

等待次数增加的可能原因

当 Lock:extend 事件的发生率超过正常（可能表示性能问题）时，典型原因包括以下几点：

对同一表的并发插入或更新激增

插入或更新同一表的查询的并发会话数可能会增加。

网络带宽不足

数据库实例上的网络带宽可能不足以满足当前工作负载的存储通信需求。这可能会导致存储延迟，从而导致 Lock:extend 事件增加。

操作

根据等待事件的原因，我们建议采取不同的操作。

主题

- 减少同一关系的并发插入和更新

- 提高网络带宽

减少同一关系的并发插入和更新

首先，确定 tup_inserted 和 tup_updated 指标是否有增加，以及此等待事件是否伴随增加。如果是这样，请检查哪些关系在插入和更新操作中处于高争用状态。要确定这一点，请查询 pg_stat_all_tables 视图，以了解 n_tup_ins 和 n_tup_upd 字段中的值。有关 pg_stat_all_tables 视图的信息，请参阅 PostgreSQL 文档中的 pg_stat_all_tables。

要获取有关正在阻止和已阻止的查询的更多信息，请如以下示例所示查询 pg_stat_activity：

```
SELECT
    blocked.pid,
    blocked.usename,
    blocked.query,
    blocking.pid AS blocking_id,
    blocking.query AS blocking_query,
    blocking.wait_event AS blocking_wait_event,
    blocking.wait_event_type AS blocking_wait_event_type
FROM pg_stat_activity AS blocked
JOIN pg_stat_activity AS blocking ON blocking.pid = ANY(pg_blocking_pids(blocked.pid))
where
blocked.wait_event = 'extend'
and blocked.wait_event_type = 'Lock';

   pid  | usename  |            query             | blocking_id |
            blocking_query                          | blocking_wait_event |
 blocking_wait_event_type
  ------+----------+------------------------------+-------------
+-------------------------------------------------------------------
+---------------------+--------------------------
   7143 |  myuser  | insert into tab1 values (1); |        4600 | INSERT INTO tab1 (a)
 SELECT s FROM generate_series(1,1000000) s; | DataFileExtend      | IO
```

在您确定有助于增加 Lock:extend 事件的关系后，请使用以下方法来减少争用：

- 查明是否可以使用分区来减少同一个表的争用。将插入或更新的元组分成不同的分区可以减少争用。有关分区的信息，请参阅 使用 pg_partman 扩展管理 PostgreSQL 分区。
- 如果等待事件主要是由于更新活动造成的，请考虑减少关系的 fillfactor 值。这可以减少更新期间对新数据块的请求。fillfactor 是表的存储参数，用于确定打包表页面的最大空间量。它表示为页面总空间的百分比。有关 fillfactor 参数的更多信息，请参阅 PostgreSQL 文档中的 CREATE TABLE。

> **Important**
>
> 我们强烈建议您在更改 fillfactor 时测试系统，因为更改此值可能会对性能产生负面影响，这具体取决于您的工作负载。

提高网络带宽

要查看写入延迟是否增加，请检查 CloudWatch 中的 `WriteLatency` 指标。如果有，请使用 `WriteThroughput` 和 `ReadThroughput` Amazon CloudWatch 指标监控数据库集群上与存储相关的流量。这些指标可以帮助您确定网络带宽是否足以满足您的工作负载的存储活动。

如果您的网络带宽不够，请增加它。如果您的数据库实例已达到网络带宽限制，增加带宽的唯一方法是增加数据库实例大小。

有关 CloudWatch 指标的更多信息，请参阅Amazon Aurora 的 Amazon CloudWatch 指标。有关每个数据库实例类的网络性能的信息，请参阅 适用于 Aurora 的数据库实例类的硬件规格。

Lock:Relation

当查询等待获取当前被另一个事务锁定的表或视图（关系）上的锁定时，会发生 `Lock:Relation` 事件。

主题

- 支持的引擎版本
- 上下文
- 等待次数增加的可能原因
- 操作

支持的引擎版本

Aurora PostgreSQL 的所有版本均支持此等待事件信息。

上下文

大多数 PostgreSQL 命令隐式使用锁来控制对表中数据的并发访问。您还可以通过 `LOCK` 命令在应用程序代码中显式使用这些锁。许多锁定模式彼此不兼容，它们可以在尝试访问同一对象时阻止事务。发生这种情况时，Aurora PostgreSQL 会生成一个 `Lock:Relation` 事件。以下是一些常见的示例：

- ACCESS EXCLUSIVE 之类的独占锁可以阻止所有并发访问。数据定义语言 (DDL) 操作（例如 DROP TABLE、TRUNCATE、VACUUM FULL 和 CLUSTER）隐式获取 ACCESS EXCLUSIVE 锁定。ACCESS EXCLUSIVE 也是用于未显式指定模式的 LOCK TABLE 语句的原定设置锁定模式。
- 在表上使用 CREATE INDEX (without CONCURRENT) 与数据操作语言 (DML) 语句 UPDATE、DELETE 和 INSERT 有冲突，这些语句可获取 ROW EXCLUSIVE 锁定。

有关表级锁和冲突锁模式的更多信息，请参阅 PostgreSQL 文档中的显式锁定。

阻止查询和事务通常通过以下方式之一解锁阻止：

- 阻止查询 – 应用程序可以取消查询或者用户可以结束该过程。引擎还可以因会话的语句超时或死锁检测机制而强制结束查询。
- 阻止事务 – 事务在运行 ROLLBACK 或 COMMIT 语句时停止阻止。当会话被客户端或网络问题断开连接或结束时，也会自动发生回滚。当数据库引擎关闭、系统内存不足等时，可以结束会话。

等待次数增加的可能原因

当 Lock:Relation 事件的发生频率高于正常值时，可能表明存在性能问题。典型的原因包括：

增加与表锁冲突的并发会话

用冲突锁定模式锁定相同表格的查询的并发会话数可能会增加。

维护操作

VACUUM 和 ANALYZE 之类的运行状况维护操作可以显著增加冲突锁的数量。VACUUM FULL 获取 ACCESS EXCLUSIVE 锁，ANALYZE 获取 SHARE UPDATE EXCLUSIVE锁。这两种类型的锁都可能导致 Lock:Relation 等待事件。应用程序数据维护操作（例如刷新具体化视图）也可以增加阻止的查询和事务。

读取器实例的锁定

写入器和读取器持有的关系锁之间可能存在冲突。目前，仅 ACCESS EXCLUSIVE 关系锁复制到读取器实例。但是，ACCESS EXCLUSIVE 关系锁将与读取器持有的任何 ACCESS SHARE 关系锁冲突。这可能会导致读取器上的锁定关系等待事件增加。

操作

根据等待事件的原因，我们建议采取不同的操作。

主题

- 减少阻止 SQL 语句的影响
- 尽量减少维护操作的影响
- 检查读取器锁

减少阻止 SQL 语句的影响

为了减少阻止 SQL 语句的影响，请尽可能修改应用程序代码。以下是减少数据块的两种常见方法：

- 使用 NOWAIT 选项 – 一些 SQL 命令，例如 SELECT 和 LOCK 语句，支持此选项。如果无法立即获取锁定，则 NOWAIT 指令将取消请求锁定的查询。这种方法可以帮助防止阻止会话导致其后面堆积阻止的会话。

  例如：假设事务 A 正在等待事务 B 所持有的锁定。现在，如果 B 请求对被事务 C 锁定的表进行锁，那么事务 A 可能会被阻止，直到事务 C 完成。但是，如果事务 B 在请求对 C 进行锁定时使用 NOWAIT，它可能会很快失败，并确保事务 A 不必无限期等待。
- 使用 SET lock_timeout – 设置 lock_timeout 值，以限制 SQL 语句等待获取关系锁的时间。如果未在指定的超时内获取锁定，则请求锁定的事务将被取消。在会话级别设置此值。

尽量减少维护操作的影响

维护操作（例如 VACUUM 和 ANALYZE）非常重要。我们建议您不要将其关闭，因为您会找到与这些维护操作相关的 Lock:Relation 等待事件。以下方法可以最大限度地减少这些操作的影响：

- 在非高峰时段手动运行维护操作。
- 要减少由 Autovacuum 任务导致的 Lock:Relation 等待，执行任何需要的 Autovacuum 优化。有关优化 Autovacuum 的信息，请参阅《Amazon RDS 用户指南》中的在 Amazon RDS 上使用 PostgreSQL Autovacuum。

检查读取器锁

您可以看到写入器和读取器的并发会话如何持有互相阻止的锁。执行此操作的一种方法是运行返回锁定类型和关系的查询。在该表中，可以找到对两个此类并发会话（一个写入器会话和一个读取器会话）的一系列查询。

重播过程会在取消读取器查询之前等待持续时间 max_standby_streaming_delay。如示例中所示，100 毫秒的锁定超时远低于 max_standby_streaming_delay 默认值 30 秒。锁定在出现问题之前已超时。

序列事件	会话	命令或输出

使用指定的值设置名为 READER 的环境变量，并尝试使用此端点连接到数据库实例。	读取器会话	CLI 命令：
		`export READER=aurorapg2.12345678910.us-west-1.rds.amazonaws.com`
		`psql -h $READER`
		输出：
		`psql (15devel, server 10.14)`
		`Type "help" for help.`

设置名为 WRITER 的环境变量，并尝试使用此端点连接到数据库实例。	写入器会话	CLI 命令：
		`export WRITER=aurorapg1.12345678910.us-west-1.rds.amazonaws.com`
		`psql -h $WRITER`
		输出：
		`psql (15devel, server 10.14)`
		`Type "help" for help.`

写入器会话在写入器实例上创建表 t1。	写入器会话	PostgreSQL 查询： `postgres=> CREATE TABLE t1(b integer);`

序列事件	会话	命令或输出
		`CREATE TABLE`
如果写入器上没有冲突的查询，则会立即在写入器上获取 ACCESS EXCLUSIVE 锁定。	写入器会话	`ACCESS EXCLUSIVE` 锁定已启用

读取器会话将锁定超时间隔设置为 100 毫秒。	读取器会话	PostgreSQL 查询：
		`postgres=> SET lock_timeout=100;`
		`SET`

读取器会话尝试从读取器实例上的表 t1 中读取数据。	读取器会话	PostgreSQL 查询：
		`postgres=> SELECT * FROM t1;`
		示例输出：
		`b`
		`---`
		`(0 rows)`

写入器会话删除 t1。	写入器会话	PostgreSQL 查询：
		`postgres=> BEGIN;`
		`BEGIN`
		`postgres=> DROP TABLE t1;`
		`DROP TABLE`
		`postgres=>`

序列事件	会话	命令或输出

查询超时并在读取器上被取消。	读取器会话	PostgreSQL 查询：
		`postgres=> SELECT * FROM t1;`
		示例输出：
		`ERROR: canceling statement due to lock timeout`
		`LINE 1: SELECT * FROM t1;`
		`^`

为确定错误的原因，读取器会话查询 pg_locks 和 pg_stat_activity 。	读取器会话	PostgreSQL 查询：
		`postgres=> SELECT locktype, relation, mode, backend_type`
		`postgres=> FROM pg_locks l, pg_stat_activity t1`
		`postgres=> WHERE l.pid=t1.pid AND relation = 't1'::regclass::oid;`

序列事件	会话	命令或输出

结果表明，aurora wal replay 进程对表 t1 持有 ACCESS EXCLUSIVE 锁定。	读取器会话	查询结果：
		`locktype | relation | mode | backend_type`
		`----------+----------+-------------------+---------------`
		`relation | 68628525 | AccessExclusiveLock | aurora wal replay`
		`(1 row)`

Lock:transactionid

当事务正在等待行级锁定时，会发生 Lock:transactionid 事件。

主题

- 支持的引擎版本
- 上下文
- 等待次数增加的可能原因
- 操作

支持的引擎版本

Aurora PostgreSQL 的所有版本均支持此等待事件信息。

上下文

当事务试图获取已被授予同时运行的事务的行级锁时，会发生事件 Lock:transactionid。显示 Lock:transactionid 等待事件的会话因此锁定被阻止。当阻止事务在 COMMIT 或 ROLLBACK 语句中结束后，被阻止的事务可以继续进行。

Aurora PostgreSQL 的多版本并发控制语义保证读取器不会阻止写入器，写入器也不会阻止读取器。为了发生行级冲突，正在阻止和已阻止的事务必须发出以下类型的冲突语句：

- UPDATE
- SELECT … FOR UPDATE
- SELECT … FOR KEY SHARE

语句 SELECT … FOR KEY SHARE 是一种特殊情况。数据库使用子句 FOR KEY SHARE 以优化参照完整性的性能。一行上的行级锁定可以组织引用该行的其他表上的 INSERT、UPDATE 和 DELETE 命令。

等待次数增加的可能原因

当此事件出现频率超过正常时，原因通常是 UPDATE、SELECT … FOR UPDATE 或 SELECT … FOR KEY SHARE 语句结合以下条件。

主题

- 高并发性
- 空闲事务
- 长时间运行的事务

高并发性

Aurora PostgreSQL 可以使用精细的行级锁定语义。满足以下条件时，行级冲突的可能性会增加：

- 高度并发的工作负载争用相同的行。
- 并发性增加。

空闲事务

有时，pg_stat_activity.state 列会显示值 idle in transaction。对于已开始事务但尚未发布 COMMIT 或 ROLLBACK 的会话，会显示此值。如果 pg_stat_activity.state 值不为 active，pg_stat_activity 中显示的查询是完成运行的最新版本。阻止会话没有主动处理查询，因为未完成的事务持有锁定。

如果空闲事务获得了行级锁，则可能会阻止其他会话获取它。这种情况导致等待事件 Lock:transactionid 频繁发生。要诊断问题，请检查来自的 pg_stat_activity 和 pg_locks 的输出。

长时间运行的事务

长时间运行的事务会获得很长一段时间的锁定。这些长期保留的锁定可以阻止其他事务运行。

操作

行锁定是 UPDATE、SELECT … FOR UPDATE 或 SELECT … FOR KEY SHARE 语句之间发生的冲突。在尝试解决方案之前，请先了解这些语句何时在同一行上运行。使用此信息选择以下各部分所述的策略。

主题

- 响应高并发
- 回应空闲事务
- 响应长期运行的事务

响应高并发

如果并发性是问题，请尝试以下方法之一：

- 降低应用程序中的并发率。例如，减少活动会话的数量。
- 实施连接池。要了解如何使用 RDS 代理进行池连接，请参阅 适用于 Aurora 的 Amazon RDS 代理。
- 设计应用程序或数据模型以避免争用 UPDATE 和 SELECT … FOR UPDATE 语句。您还可以减少 SELECT … FOR KEY SHARE 语句访问的外键的数量。

回应空闲事务

如果 pg_stat_activity.state 显示 idle in transaction，请使用以下策略：

- 尽可能开启自动提交。这种方法可防止事务在等待 COMMIT 或 ROLLBACK 时阻止其他事务。
- 搜索缺失 COMMIT、ROLLBACK 或 END 的代码路径。
- 确保应用程序中的异常处理逻辑始终具有通向有效 end of transaction 的路径。
- 确保您的应用程序在结束与COMMIT 或 ROLLBACK 的事务后处理查询结果。

响应长期运行的事务

如果长时间运行的事务导致频繁发生 Lock:transactionid，请尝试以下策略：

- 在长时间运行的事务中保持行锁定。
- 尽可能通过实现自动提交来限制查询的长度。

Lock:tuple

在后端进程等待获取元组锁定时，会发生 Lock:tuple 事件。

主题

- 支持的引擎版本
- 上下文
- 等待次数增加的可能原因
- 操作

支持的引擎版本

Aurora PostgreSQL 的所有版本均支持此等待事件信息。

上下文

事件 Lock:tuple 表示一个后端正在等待获取元组上的锁定，而另一个后端在同一个元组上保持冲突锁定。下表说明了会话生成 Lock:tuple 事件的场景。

时间	会话 1	会话 2	会话 3
t1	开始事务。		
t2	更新第 1 行。		
t3		更新第 1 行。会话获取元组上的独占锁定，然后等待会话 1 通过提交或回滚来释放锁。	
t4			更新第 1 行。会话等待会话 2 才能释放元组上的独占锁定。

或者您可以使用基准测试工具 pgbench 来模拟此等待事件。配置大量并发会话以使用自定义 SQL 文件更新表中的同一行。

要了解冲突锁模式的更多信息，请参阅 PostgreSQL 文档中的显式锁定。要了解有关 pgbench 的更多信息，请参阅 PostgreSQL 文档中的 pgbench。

等待次数增加的可能原因

当此事件的发生率超过正常（可能表示性能问题）时，典型原因包括以下几点：

- 大量并发会话试图通过运行 UPDATE 或 DELETE 语句获取相同元组的冲突锁定。
- 高度并发的会话正在使用 FOR UPDATE 或 FOR NO KEY UPDATE 锁定模式运行 SELECT 语句。
- 各种因素促使应用程序或连接池打开更多会话以执行相同的操作。由于新会话正在尝试修改相同的行，数据库负载可能会激增，Lock:tuple 可以出现。

有关更多信息，请参阅 PostgreSQL 文档中的行级锁定。

操作

根据等待事件的原因，我们建议采取不同的操作。

主题

- 调查应用程序逻辑
- 查找阻止器会话
- 在并发性高时降低并发性
- 排查瓶颈

调查应用程序逻辑

了解阻止器会话是否已经处于 idle in transaction 状态很长一段时间。如果是这样，请考虑结束阻止器会话，作为短期解决方案。您可以使用 pg_terminate_backend 函数。有关此函数的更多信息，请参阅 PostgreSQL 文档中的服务器信号函数。

要获得长期解决方案，请执行以下操作：

- 调整应用程序逻辑。
- 使用 idle_in_transaction_session_timeout 参数。此参数可结束空闲时间超过指定时间的已打开事务的任何会话。有关更多信息，请参阅 PostgreSQL 文档中的客户端连接原定设置。

- 尽可能多地使用自动提交。有关更多信息，请参阅 PostgreSQL 文档中的 SET AUTOCOMMIT。

查找阻止器会话

在 Lock:tuple 等待事件发生时，通过找出哪些锁相互依赖来识别阻止器和已阻止的会话。有关更多信息，请参阅 PostgreSQL wiki 中的锁定依赖项信息。要分析过去的 Lock:tuple 事件，请使用 Aurora 函数 aurora_stat_backend_waits。

以下示例查询所有会话，并对 tuple 进行筛选，通过 wait_time 进行排序。

```
--AURORA_STAT_BACKEND_WAITS
      SELECT a.pid,
             a.usename,
             a.app_name,
             a.current_query,
             a.current_wait_type,
             a.current_wait_event,
             a.current_state,
             wt.type_name AS wait_type,
             we.event_name AS wait_event,
             a.waits,
             a.wait_time
        FROM (SELECT pid,
                     usename,
                     left(application_name,16) AS app_name,
                     coalesce(wait_event_type,'CPU') AS current_wait_type,
                     coalesce(wait_event,'CPU') AS current_wait_event,
                     state AS current_state,
                     left(query,80) as current_query,
                     (aurora_stat_backend_waits(pid)).*
                FROM pg_stat_activity
               WHERE pid <> pg_backend_pid()
                 AND usename<>'rdsadmin') a
NATURAL JOIN aurora_stat_wait_type() wt
NATURAL JOIN aurora_stat_wait_event() we
WHERE we.event_name = 'tuple'
    ORDER BY a.wait_time;

  pid  | usename | app_name |                 current_query                  |
 current_wait_type | current_wait_event | current_state | wait_type | wait_event |
 waits | wait_time
```

```
-------+---------+----------+-------------------------------------------------
+-------------------+--------------------+--------------+-----------+-----------
+-------+-----------
 32136 | sys     | psql     | /*session3*/ update tab set col=1 where col=1; | Lock
           | tuple              | active        | Lock      | tuple      |     1 |
 1000018
 11999 | sys     | psql     | /*session4*/ update tab set col=1 where col=1; | Lock
           | tuple              | active        | Lock      | tuple      |     1 |
 1000024
```

在并发性高时降低并发性

Lock:tuple 事件可能会不断发生，特别是在繁忙的工作负载时间。在这种情况下，考虑降低非常繁忙的行的高并发率。通常，只有几个行控制队列或布尔逻辑，这使得这些行非常繁忙。

您可以根据业务需求、应用程序逻辑和工作负载类型使用不同的方法来降低并发性。例如，您可以执行以下操作：

- 重新设计表和数据逻辑以降低高并发性。
- 更改应用程序逻辑以降低行级别的高并发性。
- 使用行级锁定利用和重新设计查询。
- 使用具有重试操作的 NOWAIT 子句。
- 考虑使用乐观和混合锁定逻辑并发控制。
- 考虑更改数据库隔离级别。

排查瓶颈

当出现诸如 CPU 匮乏或 Amazon EBS 带宽的最大使用率的瓶颈时，可能会发生 Lock:tuple。要减少瓶颈，请考虑以下方法：

- 纵向扩展您的实例类类型。
- 优化资源密集型查询。
- 更改应用程序逻辑。
- 存档很少访问的数据。

LWLock:buffer_content (BufferContent)

当某个会话等待读取或写入内存中的某个数据页面，而另一个会话正锁定该页面以进行写入时，会发生 LWLock:buffer_content 事件。在 Aurora PostgreSQL 13 及更高版本中，此等待事件被称为 BufferContent。

主题

- 支持的引擎版本
- 上下文
- 等待次数增加的可能原因
- 操作

支持的引擎版本

Aurora PostgreSQL 的所有版本均支持此等待事件信息。

上下文

要读取或操作数据，PostgreSQL 会通过共享内存缓冲区访问数据。要从缓冲区读取，进程会在共享模式下获取缓冲区内容的轻量级锁 (LWLock)。要写入缓冲区，它会在独占模式下获得该锁。共享锁允许其他进程同时获取对该内容的共享锁。独占锁可防止其他进程获取对该内容的任何类型的锁定。

LWLock:buffer_content (BufferContent) 事件表示多个进程试图获取对特定缓冲区的内容的锁定。

等待次数增加的可能原因

当 LWLock:buffer_content (BufferContent) 事件的发生率超过正常（可能表示性能问题）时，典型原因包括以下几点：

增加了对同一数据的并发更新

更新相同缓冲区内容的查询的并发会话数可能会增加。在具有大量索引的表中，这种争用可能更加明显。

工作负载数据不在内存中

当活动工作负载正在处理的数据不在内存中时，这些等待事件可能会增加。这种影响是因为持有锁的进程可以在执行磁盘输入/输出操作时保持更长时间。

过度使用外键约束

外键约束可能会增加进程在缓冲区内容锁上保留的时间。这种影响是因为读取操作需要在更新引用的键时对该键进行共享缓冲区内容锁定。

操作

根据等待事件的原因，我们建议采取不同的操作。您可以通过使用 Amazon RDS 性能详情或查询视图 `pg_stat_activity` 来识别 `LWLock:buffer_content` (`BufferContent`) 事件。

主题

- 提高内存中的效率
- 减少对外键约束的使用
- 删除未使用的索引

提高内存中的效率

为了增加活动工作负载数据在内存中的可能性，请对表进行分区或纵向扩展您的实例类。有关数据库实例类的信息，请参阅 Amazon Aurora 数据库实例类。

减少对外键约束的使用

调查在使用外键约束时遇到大量`LWLock:buffer_content` (`BufferContent`) 等待事件的工作负载。删除不必要的外键约束。

删除未使用的索引

对于遇到大量 `LWLock:buffer_content`(`BufferContent`) 等待事件的工作负载，识别未使用的索引并删除它们。

LWLock:buffer_mapping

当会话正在等待将数据块与共享缓冲池中的缓冲区关联起来时，会发生此事件。

> **Note**
>
> 此事件在 Aurora PostgreSQL 版本 12 及更低版本中显示为 `LWLock:buffer_mapping`，在版本 13 及更高版本中显示为 `LWLock:BufferMapping`。

主题

- 支持的引擎版本
- 上下文
- 原因
- 操作

支持的引擎版本

此等待时间信息与 Aurora PostgreSQL 版本 9.6 及更高版本相关。

上下文

共享缓冲池是一个 Aurora PostgreSQL 内存区域，它包含进程现在正在使用或过去正在使用的所有页面。当进程需要页面时，它会将页面读入共享缓冲池中。`shared_buffers` 参数会设置共享缓冲区大小并保留一个内存区域来存储表和索引页。如果更改此参数，请确保重新启动数据库。有关更多信息，请参阅 共享缓冲区。

以下情况下回发生 `LWLock:buffer_mapping` 等待事件：

- 进程在缓冲区表中搜索页面并获取共享缓冲区映射锁。
- 进程将页面加载到缓冲池中并获取独占缓冲区映射锁。
- 进程从缓冲池中删除页面并获取独占缓冲区映射锁。

原因

当此事件发生超过正常时（可能表示性能问题），数据库正在共享缓冲池中移入和移出分页。典型的原因包括：

- 大型查询
- 臃肿的索引和表
- 完整的表扫描
- 小于工作集的共享池大小

操作

根据等待事件的原因，我们建议采取不同的操作。

主题

- 监控缓冲区相关的指标
- 评估您的索引策略
- 减少必须快速分配的缓冲区数量

监控缓冲区相关的指标

当 LWLock:buffer_mapping 等待激增时，调查缓冲区命中率。您可以使用这些指标更好地了解缓冲区缓存中发生的情况。检查以下指标：

BufferCacheHitRatio

此 Amazon CloudWatch 指标测量数据库集群中的数据库实例的缓冲区缓存处理的请求百分比。您可以会在 LWLock:buffer_mapping 等待事件的前面看到此指标降低。

blks_hit

此性能详情计数器指标表示从共享缓冲池中检索的数据块的数量。当 LWLock:buffer_mapping 等待事件出现后，您可能会观察到 blks_hit 激增。

blks_read

此性能详情计数器指标表示需要将输入/输出读入共享缓冲池的数据块的数量。您可能会在 LWLock:buffer_mapping 等待事件的前面观察到 blks_read 激增。

评估您的索引策略

要确认您的索引策略不会降低性能，请检查以下各项：

索引膨胀

确保索引和表膨胀不会导致不必要的分页被读入共享缓冲区。如果表中包含未使用的行，请考虑存档数据并从表中删除这些行。然后，您可以为调整大小的表重建索引。

常用查询的索引

要确定您是否拥有最佳索引，请在性能详情中监控数据库引擎指标。tup_returned 指标显示读取的行数。tup_fetched 指标显示返回到客户端的行数量。如果 tup_returned 明显大于 tup_fetched，可能无法正确编制数据索引。此外，您的表统计数据可能不是最新的。

减少必须快速分配的缓冲区数量

要减少 LWLock:buffer_mapping 等待事件，请尝试减少必须快速分配的缓冲区数量。一种策略是执行较小的批处理操作。通过对表进行分区，也许能够实现较小的批处理。

LWLock:BufferIO (IPC:BufferIO)

当 Aurora PostgreSQL 或 RDS for PostgreSQL 正在等待其他进程在同时尝试访问页面时完成输入/输出 (I/O) 操作时，会发生 LWLock:BufferIO 事件。它的目的是将同一个分页读入共享缓冲区中。

主题

- 相关引擎版本
- 上下文
- 原因
- 操作

相关引擎版本

此等待事件信息与所有的 Aurora PostgreSQL 版本相关。对于 Aurora PostgreSQL 12 及更早版本，此等待事件命名为 lwlock:buffer_io，而在 Aurora PostgreSQL 13 版本中，则命名为 lwlock:bufferio。从 Aurora PostgreSQL 14 版本开始，BufferIO 等待事件从 LWLock 移到 IPC 等待事件类型（IPC:BufferIO）。

上下文

每个共享缓冲区都有一个与 LWLock:BufferIO 等待事件相关的输入/输出锁，每次必须在共享缓冲池外检索数据块（或分页）。

此锁定用于处理多个会话，而这些会话都需要访问同一个数据块。必须从共享缓冲池外部读取此数据库块，该缓冲池由 shared_buffers 参数定义。

一旦在共享缓冲池内读取分页，LWLock:BufferIO 锁即被释放。

> **Note**
>
> LWLock:BufferIO 等待事件发生在 IO:DataFileRead 等待事件之前。IO:DataFileRead 事件在从存储中读取数据时发生。

有关轻量级锁定的更多信息，请参阅锁定概览。

原因

LWLock:BufferIO 显示在主要等待中的常见原因包括以下各项：

- 多个后端或连接试图访问同样在等待输入/输出操作的同一页面
- 共享缓冲池大小之间的比率（由 shared_buffers 参数定义）以及当前工作负载所需的缓冲区数量
- 共享缓冲池的大小与其他操作移出的分页数量没有很好地平衡
- 需要引擎在共享缓冲池中读取更多页面的臃肿的大索引
- 缺乏强制数据库引擎从表中读取更多页面的索引
- 试图在同一页面上执行操作的数据库连接突增

操作

根据等待事件的原因，我们建议采取不同的操作：

- 观察 Amazon CloudWatch 指标，了解 BufferCacheHitRatio 突然减少和 LWLock:BufferIO 等待事件之间的关系。此影响可能意味着您有一个较小的共享缓冲区设置。您可能需要增加数据库实例类或对其进行纵向扩展。您可以将工作负载拆分为更多的读取器节点。
- 验证是否有未使用的索引，然后将其删除。
- 使用分区表（也具有分区索引）。这样做有助于保持较低的指数重新排序并降低其影响。
- 避免对列进行不必要的索引编制。
- 使用连接池防止数据库连接突增。
- 作为最佳实践，限制与数据库的最大连接数。

LWLock:lock_manager

当 Aurora PostgreSQL 引擎维护共享锁的内存区域以便在无法使用快速路径锁时分配、检查和取消分配锁时，会发生此事件。

主题

- 支持的引擎版本
- 上下文
- 等待次数增加的可能原因

- 操作

支持的引擎版本

此等待时间信息与 Aurora PostgreSQL 版本 9.6 及更高版本相关。

上下文

发布 SQL 语句时，Aurora PostgreSQL 会记录锁，以在并发操作期间保护数据库的结构、数据和完整性。引擎可以使用快速路径锁或不快的路径锁来实现这个目标。不快的路径锁定更昂贵，并且比快速路径锁定造成的开销更大。

快速路径锁定

为了减少频繁获取和释放但很少发生冲突的锁的开销，后端进程可以使用快速路径锁定。数据库对满足以下条件的锁定使用此机制：

- 他们使用 DEFAULT 锁定方法。
- 它们代表数据库关系的锁，而不是共享关系。
- 它们是不太可能发生冲突的弱锁。
- 引擎可以快速确认不可能存在发生冲突的锁。

在以下任一条件为真时，引擎无法使用快速路径锁定：

- 锁不满足上述标准。
- 没有更多的插槽可用于后端进程。

有关快速路径锁定的更多信息，请参阅 PostgreSQL 锁管理器 README 中的快速路径和 PostgreSQL 文档中的 pg-locks。

锁管理器的扩缩问题示例

在此示例中，名为 `purchases` 的表存储五年的数据，按天进行分区。每个分区有两个索引。将发生以下一系列事件：

1. 您查询了许多天的数据，这需要数据库读取许多分区。
2. 数据库为每个分区创建一个锁条目。如果分区索引是优化程序访问路径的一部分，则数据库也会为它们创建一个锁条目。

3. 当同一后端进程请求的锁条目数大于 16 时（这是 FP_LOCK_SLOTS_PER_BACKEND 的值），锁管理器会使用非快速路径锁定方法。

现代应用程序可能有数百个会话。如果并发会话在没有适当的分区修剪的情况下查询父级数据库，则数据库可能会创建数百甚至数千个非快速路径锁。通常，当此并发性高于 vCPU 的数量时，会出现 LWLock:lock_manager 等待事件。

> **Note**
>
> LWLock:lock_manager 等待事件与数据库架构中的分区或索引数量无关。相反，它与数据库必须控制的非快速路径锁的数量有关。

等待次数增加的可能原因

当 LWLock:lock_manager 等待事件发生率超出正常（可能表明性能问题）时，突增的最可能原因如下：

- 并发活动会话正在运行不使用快速路径锁的查询。这些会话还超出了最大 vCPU。
- 大量并发活动会话正在访问严重分区的表。每个分区都有多个索引。
- 数据库正在经历连接风暴。预设情况下，当数据库缓慢时，某些应用程序和连接池软件会创建更多连接。这种做法使问题变得更糟。优化连接池软件，以免发生连接风暴。
- 大量会话在不修剪分区的情况下查询父级表。
- 数据定义语言 (DDL)、数据操作语言 (DML) 或维护命令专门锁定经常访问或修改的繁忙关系或元组。

操作

根据等待事件的原因，我们建议采取不同的操作。

主题

- 使用分区修剪
- 删除不必要的索引
- 优化查询以实现快速路径锁定
- 优化其他等待事件
- 减少硬件瓶颈

- 使用连接池程序。
- 升级您的 Aurora PostgreSQL 版本

使用分区修剪

分区修剪是一种查询优化策略，它从表扫描中排除不需要的分区，从而提高性能。预设情况下，分区修剪处于开启状态。如果它已关闭，请按如下方式将其打开。

```
SET enable_partition_pruning = on;
```

查询可以在其 WHERE 子句包含用于分区的列时利用分区修建。有关更多信息，请参阅 PostgreSQL 文档中的分区修建。

删除不必要的索引

数据库可能包含未使用或很少使用的索引。如果是，请考虑删除它们。请执行以下任一操作：

- 通过阅读 PostgreSQL wiki 中的未使用索引了解如何查找不必要的索引。
- 运行 PG 收集器。此 SQL 脚本会收集数据库信息并将其显示在整合的 HTML 报告中。检查“未使用的索引”部分。有关更多信息，请参阅 AWS Labs GitHub 存储库中的 pg-collector。

优化查询以实现快速路径锁定

要了解您的查询是否使用快速路径锁定，请查询 pg_locks 表中的 fastpath 列。如果您的查询没有使用快速路径锁定，请尝试将每个查询的关系数减少到 16 个以下。

优化其他等待事件

如果 LWLock:lock_manager 排在主要等待列表中的第一个或第二个，请检查列表中是否也显示了以下等待事件：

- Lock:Relation
- Lock:transactionid
- Lock:tuple

如果前面的事件显示在列表的前面，请考虑首先优化这些等待事件。这些事件可以是 LWLock:lock_manager 的驱动因素。

减少硬件瓶颈

您可能存在硬件瓶颈，例如 CPU 匮乏或 Amazon EBS 带宽的最大使用率。在这样的情况下，需考虑减少硬件瓶颈。请考虑以下操作：

- 纵向扩展您的实例类。
- 优化占用大量 CPU 和内存的查询。
- 更改应用程序逻辑。
- 将您的数据存档。

有关 CPU、内存和 EBS 网络带宽的更多信息，请参阅 Amazon RDS 实例类型。

使用连接池程序。

如果您的活动连接总数超过最大 vCPU，则需要 CPU 的操作系统进程超过实例类型所能支持的数量。在这种情况下，需考虑使用或优化连接池。有关您的实例类型 vCPU 数量的更多信息，请参阅 Amazon RDS 实例类型。

有关为连接池的更多信息，请参阅以下资源：

- 适用于 Aurora 的 Amazon RDS 代理
- pgbouncer
- PostgreSQL 文档中的连接池和数据源

升级您的 Aurora PostgreSQL 版本

如果您当前的 Aurora PostgreSQL 版本低于 12，请升级到版本 12 或更高版本。PostgreSQL 版本 12 和 13 具有改进的分区机制。有关版本 12 的更多信息，请参阅 PostgreSQL 12.0 发布说明。有关升级 Aurora PostgreSQL 的更多信息，请参阅 Amazon Aurora PostgreSQL 的数据库引擎更新。

LWLock:MultiXact

`LWLock:MultiXactMemberBuffer`、`LWLock:MultiXactOffsetBuffer`、`LWLock:MultiXactMembe` 和 `LWLock:MultiXactOffsetSLRU` 等待事件表示会话正在等待检索修改给定表中同一行的事务列表。

- `LWLock:MultiXactMemberBuffer` – 进程正在等待 multixact 成员的最近使用最少的（SLRU）简单缓冲区上的输入/输出。

- `LWLock:MultiXactMemberSLRU` – 进程正在等待访问 multixact 成员的最近使用最少的（SLRU）简单缓存。
- `LWLock:MultiXactOffsetBuffer` – 进程正在等待 multixact offset 的最近使用最少的（SLRU）简单缓冲区上的输入/输出。
- `LWLock:MultiXactOffsetSLRU` – 进程正在等待访问 multixact offset 的最近使用最少的（SLRU）简单缓存。

主题

- 支持的引擎版本
- 上下文
- 等待次数增加的可能原因
- 操作

支持的引擎版本

Aurora PostgreSQL 的所有版本均支持此等待事件信息。

上下文

multixact 是一种数据结构，用于存储修改同一表行的事务 ID（XID）列表。当单个事务引用表中的一行时，事务 ID 存储在表标题行中。当多个事务引用表中的同一行时，事务 ID 列表存储在 multixact 数据结构中。Multixact 等待事件表示会话正在从数据结构中检索引用表中给定行的事务列表。

等待次数增加的可能原因

使用 multixact 的三个常见原因如下所示：

- 来自显式保存点的子事务：在事务中显式创建保存点会为同一行生成新事务。例如，使用 `SELECT FOR UPDATE`，接着使用 `SAVEPOINT`，然后使用 `UPDATE`。

  某些驱动程序、对象关联映射器（ORM）和抽象层具有配置选项，可以使用保存点自动包装所有操作。这会在某些工作负载中生成许多 multixact 等待事件。PostgreSQL JDBC 驱动程序的 `autosave` 选项就是这样一个例子。有关更多信息，请参阅 PostgreSQL JDBC 文档中的 pgJDBC。另一个例子是 PostgreSQL ODBC 驱动程序及其 `protocol` 选项。有关更多信息，请参阅 PostgreSQL ODBC 驱动程序文档中的 psqlODBC 配置选项。
- 来自 PL/pgSQL EXCEPTION 子句的子事务：您在 PL/pgSQL 函数或过程中编写的每个 `EXCEPTION` 子句都会在内部创建一个 `SAVEPOINT`。

- 外键：多个事务在父行上获取共享锁定。

当给定行包含在多事务操作中时，处理该行要求从 `multixact` 列表中检索事务 ID。如果查找无法从内存缓存中获取 multixact，则必须从 Aurora 存储层读取数据结构。存储中的这一输入/输出意味着 SQL 查询可能需要更长的时间。由于大量的多事务，可能会在使用量很大时开始发生内存缓存未命中。所有这些因素都导致这种等待事件的增加。

操作

根据等待事件的原因，我们建议采取不同的操作。其中一些操作有助于立即减少等待事件。但是，其它一些操作可能需要进行调查和更正以扩展您的工作负载。

主题

- 使用此等待事件对表执行 vacuum 冻结
- 增加具有此等待事件的表上的 autovacuum 频率
- 增加内存参数
- 减少长时间运行的事务
- 长期操作

使用此等待事件对表执行 vacuum 冻结

如果此等待事件突然激增并影响您的生产环境，则可以使用以下任何临时方法减少其数量。

- 在受影响的表或表分区上使用 VACUUM FREEZE 以立即解决问题。有关更多信息，请参阅 VACUUM。
- 使用 VACUUM (FREEZE, INDEX_CLEANUP FALSE) 子句通过跳过索引来执行快速 vacuum 操作。有关更多信息，请参阅尽快对表执行 vacuum 操作。

增加具有此等待事件的表上的 autovacuum 频率

在扫描所有数据库中的所有表后，VACUUM 最终将删除 multixact，并前移其最旧的 multixact 值。有关更多信息，请参阅 Multixacts and Wraparound。要将 LWLock:MultiXact 等待事件数降至其最低值，必须根据需要尽可能多运行 VACUUM。为此，请确保对 Aurora PostgreSQL 数据库集群中的 VACUUM 进行了最佳配置。

如果在受影响的表或表分区上使用 VACUUM FREEZE 可以解决等待事件问题，我们建议使用调度器（例如 `pg_cron`）来执行 VACUUM，而不是在实例级别调整 autovacuum 操作。

为了使 autovacuum 更频繁地发生，您可以减少受影响表中的存储参数 `autovacuum_multixact_freeze_max_age` 的值。有关更多信息，请参阅 autovacuum_multixact_freeze_max_age。

增加内存参数

可以通过调整以下参数来优化 multixact 缓存的内存使用情况。这些设置控制为这些缓存预留多少内存，这有助于减少工作负载中的 multixact 等待事件。建议从以下值开始：

对于 Aurora PostgreSQL 17 和更高版本：

- `multixact_offset_buffers` = 128
- `multixact_member_buffers` = 256

对于 Aurora PostgreSQL 16 及更早版本：

- `multixact_offsets_cache_size` = 128
- `multixact_members_cache_size` = 256

> **Note**
>
> 在 Aurora PostgreSQL 17 中，为了与社区 PostgreSQL 17 保持一致，参数名称已从 `multixact_offsets_cache_size` 更改为 `multixact_offset_buffers` 和从 `multixact_members_cache_size` 更改为 `multixact_member_buffers`。

可以在集群级别设置这些参数，以使集群中的所有实例保持一致。建议您测试和调整这些值，使其最适合您的特定工作负载要求和实例类。必须重启写入器实例才能使参数更改生效。

这些参数以 multixact 缓存条目表示。每个缓存条目使用 8 KB 内存。要计算预留的总内存，请将每个参数值乘以 8 KB。例如，如果将参数设置为 128，则预留内存总量为 `128 * 8 KB = 1 MB`。

减少长时间运行的事务

长时间运行的事务会导致 vacuum 在事务提交或只读事务关闭之前保留其信息。建议您主动监视和管理长时间运行的事务。有关更多信息，请参阅 数据库具有长时间运行的事务空闲连接。尝试修改您的应用程序，以避免或尽量减少使用长时间运行的事务。

长期操作

检查您的工作负载，以找出 multixact 溢出的原因。为了扩展工作负载并减少等待事件，您必须修复该问题。

- 您必须分析用于创建表的 DDL（数据定义语言）。确保表结构和索引设计良好。
- 当受影响的表有外键时，请确定是否需要外键，或者是否有其它方法可以强制执行引用完整性。
- 当表有大量未使用的索引时，可能会导致 autovacuum 不适合您的工作负载，并可能阻止其运行。为避免这种情况，请检查是否有未使用的索引并将其完全删除。有关更多信息，请参阅使用大型索引管理 autovacuum。
- 减少在事务中使用保存点。

LWLock:pg_stat_statements

当 `pg_stat_statements` 扩展对跟踪 SQL 语句的哈希表进行独占锁定时，就会发生 LWLock:pg_stat_statements 等待事件。如果存在以下情形，可能会发生这种情况：

- 当跟踪的语句数量达到配置的 `pg_stat_statements.max` 参数值，并且需要为更多条目腾出空间时，此扩展会对调用次数执行排序，移除 5% 的执行次数最少的语句，然后用剩余的条目重新填充哈希表。
- 当 `pg_stat_statements` 对磁盘上的 `pgss_query_texts.stat` 文件执行 `garbage collection` 操作并重新写入该文件时。

主题

- 支持的引擎版本
- 上下文
- 等待次数增加的可能原因
- 操作

支持的引擎版本

Aurora PostgreSQL 的所有版本均支持此等待事件信息。

上下文

了解 pg_stat_statements 扩展：pg_stat_statements 扩展在哈希表中跟踪 SQL 语句执行统计数据。该扩展可跟踪 SQL 语句，直至达到由 `pg_stat_statements.max` 参数定义的限制。此参数确定可以跟踪的最大语句数，该数量对应于 pg_stat_statements 视图中的最大行数。

语句统计数据持久性：该扩展通过以下方式，在实例重新启动时持久保留语句统计数据：

- 将数据写入名为 pg_stat_statements.stat 的文件
- 使用 pg_stat_statements.save 参数来控制持久行为

当 pg_stat_statements.save 设为以下设置时：

- 开启（默认）：关闭时保存统计数据，并在服务器启动时重新加载它们
- 关闭：既不会在服务器关闭时保存统计数据，也不会在服务器启动时重新加载它们

查询文本存储：该扩展将所跟踪的查询的文本存储在一个名为 pgss_query_texts.stat 的文件中。在进行垃圾回收之前，该文件可以增长到所有跟踪的 SQL 语句平均大小的两倍。该扩展需要在清理操作和重新写入 pgss_query_texts.stat 文件期间对哈希表实施独占锁定。

语句取消分配流程：当所跟踪语句的数量达到 pg_stat_statements.max 限制并且需要跟踪新的语句时，此扩展会：

- 对哈希表实施独占锁定（LWLock:pg_stat_statements）。
- 将现有数据加载到本地内存中。
- 根据调用次数执行快速排序。
- 移除调用次数最少的语句（底部 5%）。
- 用剩余的条目重新填充哈希表。

监控语句取消分配：在 PostgreSQL 14 及更高版本中，可以使用 pg_stat_statements_info 视图来监控语句取消分配。此视图包括一个 dealloc 列，该列显示对语句取消分配来为新语句腾出空间的次数

如果频繁地对语句取消分配，则会导致更频繁地对磁盘上的 pgss_query_texts.stat 文件进行垃圾回收。

等待次数增加的可能原因

LWLock:pg_stat_statements 等待次数增加的典型原因包括：

- 应用程序使用的唯一查询的数量有所增加。
- 与正在使用的唯一查询的数量相比，pg_stat_statements.max 参数值较小。

操作

根据等待事件的原因，我们建议采取不同的操作。可以通过使用 Amazon RDS 性能详情或查询视图 `pg_stat_activity` 来识别 `LWLock:pg_stat_statements` 事件。

调整以下 `pg_stat_statements` 参数，以控制跟踪行为并减少 LWLock:pg_stat_ statements 等待事件。

主题

- 禁用 pg_stat_statements.track 参数
- 提高 pg_stat_statements.max 参数的值
- 禁用 pg_stat_statements.track_utility 参数

禁用 pg_stat_statements.track 参数

如果 LWLock:pg_stat_statements 等待事件对数据库性能产生不利影响，并且在进一步分析 `pg_stat_statements` 视图以确定根本原因之前需要快速的解决方案，则可以通过将 `pg_stat_statements.track` 参数设置为 `none` 来禁用该参数。这将禁用对语句统计数据的回收。

提高 pg_stat_statements.max 参数的值

要减少取消分配并最大限度地减少磁盘上 `pgss_query_texts.stat` 文件的垃圾回收，请提高 `pg_stat_statements.max` 参数的值。默认值为 `5,000`。

> **Note**
>
> `pg_stat_statements.max` 参数为静态参数。必须重新启动数据库实例，才能应用对此参数的任何更改。

禁用 pg_stat_statements.track_utility 参数

您可以分析 pg_stat_statements 视图，以确定哪些实用程序命令正在消耗由 `pg_stat_statements` 跟踪的大部分资源。

`pg_stat_statements.track_utility` 参数控制模块是否跟踪实用程序命令，包括除 SELECT、INSERT、UPDATE、DELETE 和 MERGE 之外的所有命令。默认情况下，此参数设置为 on。

例如，当应用程序使用许多本质上为唯一的保存点查询时，它可能会增加语句取消分配。要解决此问题，您可以禁用 pg_stat_statements.track_utility 参数以停止 pg_stat_statements 跟踪保存点查询。

> **Note**
>
> pg_stat_statements.track_utility 参数是一个动态参数。无需重新启动数据库实例即可更改其值。

Example pg_stat_statements 中的唯一保存点查询示例

```
                    query                         |       queryid
--------------------------------------------------+---------------------
 SAVEPOINT JDBC_SAVEPOINT_495701                  | -7249565344517699703
 SAVEPOINT JDBC_SAVEPOINT_1320                    | -1572997038849006629
 SAVEPOINT JDBC_SAVEPOINT_26739                   |  54791337410474486
 SAVEPOINT JDBC_SAVEPOINT_1294466                 |  8170064357463507593
 ROLLBACK TO SAVEPOINT JDBC_SAVEPOINT_65016       | -33608214779996400
 SAVEPOINT JDBC_SAVEPOINT_14185                   | -2175035613806809562
 SAVEPOINT JDBC_SAVEPOINT_45837                   | -6201592986750645383
 SAVEPOINT JDBC_SAVEPOINT_1324                    |  6388797791882029332
```

PostgreSQL 17 引入了多项增强功能来进行实用程序命令跟踪：

- 保存点名称现在显示为常量。
- 现在，两阶段提交命令的全局事务 ID（GID）显示为常量。
- DEALLOCATE 语句的名称显示为常量。
- CALL 参数现在显示为常量。

Timeout:PgSleep

当服务器进程调用 pg_sleep 函数并且等待睡眠超时过期时，会发生 Timeout:PgSleep 事件。

主题

- 支持的引擎版本
- 等待次数增加的可能原因

- 操作

支持的引擎版本

Aurora PostgreSQL 的所有版本均支持此等待事件信息。

等待次数增加的可能原因

当应用程序、存储函数或用户发出调用以下函数之一的 SQL 语句时，会发生此等待事件：

- `pg_sleep`
- `pg_sleep_for`
- `pg_sleep_until`

前面的函数会延迟执行，直到经过指定的秒数为止。例如，`SELECT pg_sleep(1)` 暂停 1 秒。有关更多信息，请参阅 PostgreSQL 文档中的延迟执行。

操作

确定正在运行 `pg_sleep` 函数的语句。确定使用该功能是否合适。

使用 Amazon DevOps Guru 主动见解优化 Aurora PostgreSQL

DevOps Guru 主动见解可检测 Aurora PostgreSQL 数据库集群上可能导致问题的情况，并在问题发生之前告知您。主动的洞察可以提醒您具有长时间运行的事务空闲连接。有关排查长时间运行的事务空闲连接的更多信息，请参阅数据库具有长时间运行的事务空闲连接。

DevOps Guru 可以执行以下操作：

- 通过对照常见的建议设置交叉检查数据库配置，可以防止许多常见的数据库问题。
- 提醒您注意实例集中的关键问题，如果不加以检查，以后可能会导致更大的问题。
- 提醒您注意新发现的问题。

每项主动见解都包含对问题原因的分析和纠正措施建议。

有关适用于 Amazon RDS 的 Amazon DevOps Guru 的更多信息，请参阅使用 Amazon DevOps Guru for Amazon RDS 分析 Aurora 性能异常。

数据库具有长时间运行的事务空闲连接

与数据库的连接处于 `idle in transaction` 状态的时间已超过 1800 秒。

主题

- 支持的引擎版本
- 上下文
- 这个问题的可能原因
- 操作
- 相关指标

支持的引擎版本

Aurora PostgreSQL 的所有版本都支持这些见解信息。

上下文

处于 `idle in transaction` 状态的事务可能持有旨在阻止其他查询的锁。它还可以防止 VACUUM（包括 autovaum）清理死行，从而导致索引或表膨胀或事务 ID 重叠。

这个问题的可能原因

在使用 BEGIN 或 START TRANSATION 的交互式会话中启动的事务尚未通过使用 COMMIT、ROLLBACK 或 END 命令结束。这会导致事务移至 `idle in transaction` 状态。

操作

您可以通过查询 `pg_stat_activity` 来查找空闲的事务。

在 SQL 客户端中，运行以下查询以列出所有处于 `idle in transaction` 状态的连接，并按持续时间对它们进行排序：

```
SELECT now() - state_change as idle_in_transaction_duration, now() - xact_start as
 xact_duration,*
FROM  pg_stat_activity
WHERE state  = 'idle in transaction'
AND   xact_start is not null
ORDER BY 1 DESC;
```

根据见解的原因，我们建议采取不同的操作。

主题

- 结束事务
- 终止连接
- 配置 idle_in_transaction_session_timeout 参数
- 检查 AUTOCOMMIT 状态
- 检查应用程序代码中的事务逻辑

结束事务

当您在交互式会话中使用 BEGIN 或 START TRANSACTION 启动事务时，它会转入 `idle in transaction` 状态。它会一直保持此状态，直到您通过发出 COMMIT、ROLLBACK、END 命令结束事务，或完全断开连接以回滚事务。

终止连接

使用以下查询终止与空闲事务的连接：

```
SELECT pg_terminate_backend(pid);
```

pid 是连接的进程 ID。

配置 idle_in_transaction_session_timeout 参数

在参数组中配置 `idle_in_transaction_session_timeout` 参数。配置此参数的优势在于，它不需要手动干预即可终止长时间空闲的事务。有关此参数的更多信息，请参阅 PostgreSQL 文档。

当事务处于 idle_in_transaction 状态超过指定时间时，连接终止后，PostgreSQL 日志文件中将报告以下消息。

```
FATAL: terminating connection due to idle in transaction timeout
```

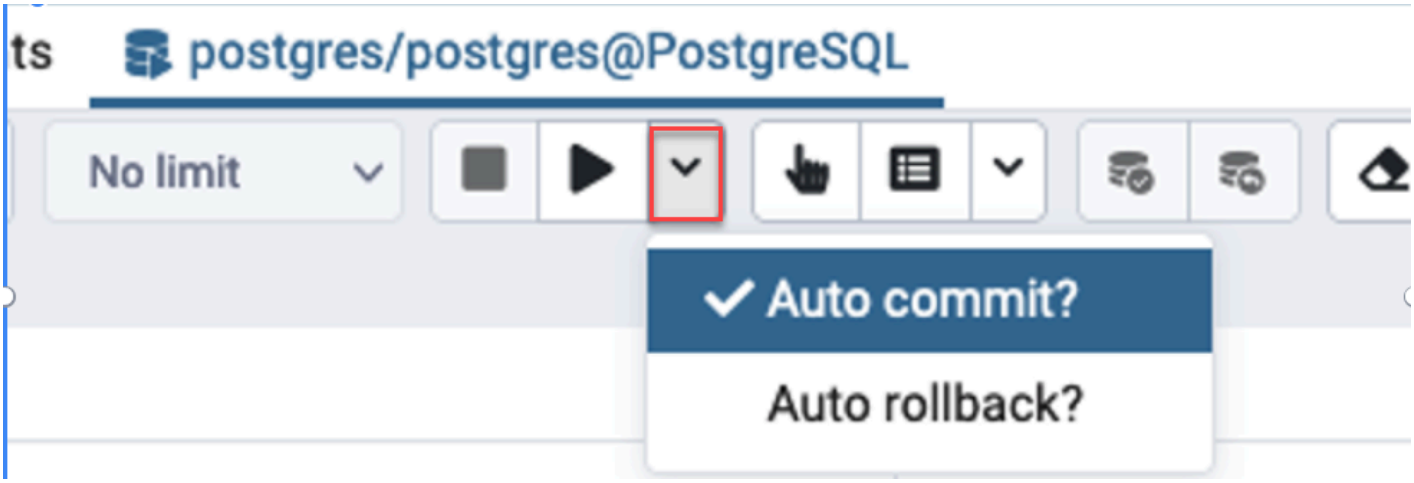
检查 AUTOCOMMIT 状态

原定设置情况下，AUTOCOMMIT 处于开启状态。但是，如果在客户端中意外将其关闭，请确保将其重新开启。

- 在 psql 客户端中，运行以下命令：

```
postgres=> \set AUTOCOMMIT on
```

- 在 pgadmin 中，通过从向下箭头中选择 AUTOCOMMIT 选项将其开启。



检查应用程序代码中的事务逻辑

调查应用程序逻辑中可能存在的问题。请考虑以下操作：

- 检查应用程序中 JDBC 自动提交是否设置为 true。另外，可以考虑在代码中使用显式 `COMMIT` 命令。
- 检查错误处理逻辑，看看它是否会在错误后关闭事务。
- 在事务打开时，检查您的应用程序处理查询所返回的行是否花费了很长时间。如果是，请考虑在处理行之前对应用程序进行编码以关闭事务。
- 检查事务是否包含许多长时间运行的操作。如果是，请将单个事务分成多个事务。

相关指标

以下 PI 指标与此见解相关：

- idle_in_transaction_count - 处于 `idle in transaction` 状态的会话数。
- idle_in_transaction_max_time - 处于 `idle in transaction` 状态的运行时间最长的事务的持续时间。

Amazon Aurora PostgreSQL 的最佳实践

接下来，您可以找到管理 Amazon Aurora PostgreSQL 数据库集群的几个最佳实践。还请务必查看基本维护任务。有关更多信息，请参阅 Amazon Aurora PostgreSQL 的性能和扩展。

主题

- 避免 Aurora PostgreSQL 数据库实例性能降低、自动重启和失效转移
- 诊断表和索引膨胀
- 改进了 Aurora PostgreSQL 中的内存管理
- Amazon Aurora PostgreSQL 的快速故障转移
- 通过 Aurora PostgreSQL 的集群缓存管理提供故障转移后的快速恢复
- 使用池管理 Aurora PostgreSQL 连接流失
- PostgreSQL 中的失效连接处理
- 调整 Aurora PostgreSQL 的内存参数
- 使用 Amazon CloudWatch 指标分析 Aurora PostgreSQL 的资源使用情况
- 使用逻辑复制对 Aurora PostgreSQL 执行主要版本升级
- 排查 Aurora PostgreSQL 中的存储问题

避免 Aurora PostgreSQL 数据库实例性能降低、自动重启和失效转移

如果您正在运行繁重的工作负载，或峰值工作负载超出为数据库实例分配的资源，则可能会耗尽运行应用程序和 Aurora 数据库的资源。要获取有关数据库实例的指标，例如 CPU 利用率、内存使用率和使用的数据库连接数，您可以参考 Amazon CloudWatch、Performance Insights 和增强型监控所提供的指标。有关监控数据库实例的更多信息，请参阅 监控 Amazon Aurora 集群中的指标。

如果您的工作负载耗尽了您正在使用的资源，则数据库实例可能会变慢、重新启动，甚至故障转移到另一个数据库实例。为避免这种情况，请监控您的资源利用率，检查数据库实例上运行的工作负载，并在必要时进行优化。如果优化不能改善实例指标和缓解资源耗尽问题，请考虑在达到数据库实例的限制之前对其进行纵向扩展。有关可用数据库实例类及其规格的更多信息，请参阅 Amazon Aurora 数据库实例类。

诊断表和索引膨胀

可以使用 PostgreSQL 多版本并发控制（MVCC）来帮助保持数据的完整性。PostgreSQL MVCC 的工作原理是保存已更新或已删除行（也称为元组）的内部副本，直到事务提交或回滚为止。这个保存的

内部副本对用户不可见。但是，当 VACUUM 或 AUTOVACUUM 实用程序未定期清理这些不可见副本时，可能会出现表膨胀。如果不加以控制，表膨胀可能会增加存储成本并降低处理速度。

在许多情况下，Aurora 上的 VACUUM 或 AUTOVACUUM 的原定设置足以处理不必要的表膨胀。但是，如果您的应用程序遇到以下情况，则可能需要检查是否存在膨胀：

- 在 VACUUM 进程之间的相对较短的时间内处理大量事务。
- 性能不佳，存储空间不足。

首先，收集最准确的信息，了解失效元组占用了多少空间，以及通过清理表和索引膨胀可以恢复多少空间。为此，请使用 pgstattuple 扩展来收集 Aurora 集群的统计数据。有关更多信息，请参阅 pgstattuple。使用 pgstattuple 扩展的权限仅限于 pg_stat_scan_tables 角色和数据库超级用户。

要在 Aurora 上创建 pgstattuple 扩展，请将客户端会话连接到集群，例如 psql 或 pgAdmin，然后使用以下命令：

```
CREATE EXTENSION pgstattuple;
```

在要分析的每个数据库中创建此扩展。创建扩展后，使用命令行界面（CLI）来衡量可以回收多少不可用的空间。在收集统计数据之前，通过将 AUTOVACUUM 设置为 0 来修改集群参数组。设置为 0 会阻止 Aurora 自动清理应用程序留下的任何失效元组，这可能会影响结果的准确性。输入以下命令以创建简单表：

```
postgres=> CREATE TABLE lab AS SELECT generate_series (0,100000);
SELECT 100001
```

在以下示例中，我们在为数据库集群开启 AUTOVACUUM 的情况下运行查询。dead_tuple_count 为 0，这表明 AUTOVACUUM 已经从 PostgreSQL 数据库中删除了过时的数据或元组。

要使用 pgstattuple 收集有关表的信息，请在查询中指定表的名称或对象标识符（OID）：

```
postgres=> SELECT * FROM pgstattuple('lab');
```

```


table_len  | tuple_count | tuple_len | tuple_percent | dead_tuple_count |
 dead_tuple_len | dead_tuple_percent | free_space | free_percent
```

```
-----------+-------------+-----------+---------------+------------------
+----------------+--------------------+------------+--------------
3629056    | 100001      | 2800028   | 77.16         | 0                | 0
  | 0                  | 16616      | 0.46
(1 row)
```

在以下查询中，我们关闭 AUTOVACUUM 并输入从表中删除 25000 行的命令。结果，dead_tuple_count 增加到 25000。

```
postgres=> DELETE FROM lab WHERE generate_series < 25000;

DELETE 25000
```

```
SELECT * FROM pgstattuple('lab');
```

```
table_len | tuple_count | tuple_len | tuple_percent | dead_tuple_count | dead_tuple_len
 | dead_tuple_percent | free_space | free_percent
-----------+-------------+-----------+---------------+------------------
+----------------+--------------------+------------+--------------
3629056 | 75001 | 2100028 | 57.87 | 25000 | 700000 | 19.29 | 16616 | 0.46
(1 row)
```

要回收那些失效的元组，请启动 VACUUM 进程。

在不中断应用程序的情况下观察膨胀

Aurora 集群上的设置经过优化，可为大多数工作负载提供最佳实践。但是，您可能需要优化集群以更好地适合您的应用程序和使用模式。在这种情况下，可以使用 pgstattuple 扩展而不必中断繁忙的应用程序。为此，请执行以下步骤：

1. 克隆您的 Aurora 实例。

2. 修改参数文件以在克隆中关闭 AUTOVACUUM。
3. 在使用示例工作负载或 pgbench（一个用于在 PostgreSQL 上运行基准测试的程序）测试克隆时执行 `pgstattuple` 查询。有关更多信息，请参阅 pgbench。

运行应用程序并查看结果后，在还原的副本上使用 pg_repack 或 VACUUM FULL 并比较差异。如果您看到 dead_tuple_count、dead_tuple_len 或 dead_tuple_percent 显著下降，请调整生产集群的 vacuum 时间表以最大限度地减少膨胀。

避免临时表出现膨胀

如果您的应用程序会创建临时表，请确保应用程序在不再需要这些临时表时将其删除。Autovacuum 进程无法找到临时表。如果不加以控制，临时表会迅速造成数据库膨胀。此外，膨胀可能扩展到系统表，这些表是跟踪 PostgreSQL 对象和属性的内部表，如 pg_attribute 和 pg_depend。

当不再需要临时表时，可以使用 TRUNCATE 语句清空该表并释放空间。然后，手动对 pg_attribute 表和 pg_depend 表执行 vacuum 操作。对这些表执行 vacuum 操作可确保持续创建和截断/删除临时表不会增加元组和导致系统膨胀。

在创建临时表时，可以通过包含以下语法来避免此问题，这些语法用于在提交内容时删除新行：

```
CREATE TEMP TABLE IF NOT EXISTS table_name(table_description) ON COMMIT DELETE ROWS;
```

提交事务时，`ON COMMIT DELETE ROWS` 子句会截断临时表。

避免索引膨胀

当您更改表中已编制索引的字段时，索引更新会导致该索引中出现一个或多个无效元组。原定设置情况下，autovacuum 进程会清理索引中的膨胀，但这种清理会消耗大量的时间和资源。要在创建表时指定索引清理首选项，请包括 vacuum_index_cleanup 子句。原定设置情况下，在创建表时，该子句设置为 AUTO，这意味着服务器在对表执行 vacuum 操作时决定您的索引是否需要清理。您可以将该子句设置为 ON 以开启特定表的索引清理，或者将该子句设置为 OFF 以关闭该表的索引清理。请记住，关闭索引清理可能会节省时间，但可能会导致索引膨胀。

在命令行中对表执行 VACUUM 时，可以手动控制索引清理。要对表执行 vacuum 操作并从索引中删除无效元组，请包括值为 ON 的 INDEX_CLEANUP 子句和表名称：

```
acctg=> VACUUM (INDEX_CLEANUP ON) receivables;
```

```
INFO: aggressively vacuuming "public.receivables"
VACUUM
```

要在不清理索引的情况下对表执行 vacuum 操作，请将值指定为 OFF：

```
acctg=> VACUUM (INDEX_CLEANUP OFF) receivables;

INFO: aggressively vacuuming "public.receivables"
VACUUM
```

改进了 Aurora PostgreSQL 中的内存管理

Aurora PostgreSQL 现在包含高级内存管理功能，这些功能可在各种工作负载下优化数据库性能和弹性。这些改进有助于 Aurora PostgreSQL 保持一致的可用性和响应能力，即使在内存需求高的时期也是如此。

在以下 Aurora PostgreSQL 版本的预置实例中，该功能可用并默认启用：

- 15.3 及所有更高的次要版本
- 14.8 及更高的次要版本
- 13.11 及更高的次要版本
- 12.15 及更高的次要版本
- 11.20 及更高的次要版本

在以下 Aurora PostgreSQL 版本的 Aurora Serverless 实例中，该功能可用并默认启用：

- 16.3 及所有更高的次要版本
- 15.7 及所有更高的次要版本
- 14.12 及更高的次要版本
- 13.15 及更高的次要版本

当客户的工作负载用完所有可用的空闲内存时，操作系统可能会重新启动数据库以保护资源，导致数据库暂时不可用。当系统遇到高内存压力时，Aurora PostgreSQL 中新的内存管理改进功能会主动取消一些事务，来帮助保持数据库的稳定性。

主要的内存管理改进功能如下：

- 当系统接近临界内存压力时，取消请求更多内存的数据库事务。
- 当系统耗尽所有物理内存并即将耗尽交换空间时，系统将被认为处于临界内存压力之下。在这些情况下，任何请求内存的事务都将被取消，以立即降低数据库实例中的内存压力。
- 必不可少的 PostgreSQL 启动器和后台工件（例如 autovacuum 工件）始终受到保护。

处理内存管理参数

开启内存管理

默认情况下，此功能处于启用状态。当由于内存不足而取消事务时，会显示一条错误消息，如以下示例所示：

```
ERROR: out of memory Detail: Failed on request of size 16777216.
```

关闭内存管理

要关闭此功能，请使用 psql 连接到 Aurora PostgreSQL 数据库集群，并对参数值使用 SET 语句，如下所述。

> **Note**
>
> 建议您保持启用内存管理。这有助于防止潜在的内存不足错误，而这些错误可能由于内存耗尽而导致工作负载引发的数据库重启。

下表显示了如何关闭不同 Aurora PostgreSQL 版本的内存管理功能：

Aurora PostgreSQL 版本	参数	默认值	在会话级别关闭内存管理的命令
11.20、11.21、12.15、12.16、13.11、13.12、14.8、14.9、15.3、15.4	`rds.memory_allocation_guard`	`false`	`SET rds.memory_allocation_guard = true;`

Aurora PostgreSQL 版本	参数	默认值	在会话级别关闭内存管理的命令
12.17、13.13、14.10、15.5 及更高版本	`rds.enable_memory_management`	`true`	`SET rds.enable_memory_management = false;`

> **Note**
>
> `rds.memory_allocation_guard` 参数在 Aurora PostgreSQL 12.17、13.13、14.10、15.5 及更高版本中已弃用。

在数据库集群参数组中设置这些参数的值可以防止查询被取消。有关数据库集群参数组的更多信息，请参阅Amazon Aurora 的参数组。

限制

- db.t3.medium 实例类不支持此功能。

Amazon Aurora PostgreSQL 的快速故障转移

接下来，您可以学习如何确保故障转移尽快进行。要在故障转移后快速恢复，您可以对 Aurora PostgreSQL 数据库集群使用集群缓存管理。有关更多信息，请参阅 通过 Aurora PostgreSQL 的集群缓存管理提供故障转移后的快速恢复。

为使故障转移快速执行，可以采取的一些步骤包括：

- 将传输控制协议（TCP）keepalive 设置为较短的时间范围，以便在出现故障时，在读取超时到期之前停止运行时间较长的查询。
- 前瞻性地设置 Java 域名系统（DNS）缓存的超时。这样做有助于确保 Aurora 只读端点在以后进行连接尝试时，能够在只读节点之间进行正常的循环切换。
- 将在 JDBC 连接字符串中使用的超时变量设置得尽可能低。对短时间和长时间运行的查询，使用单独的连接对象。
- 使用提供的读取和写入 Aurora 端点连接到集群。

- 使用 RDS API 操作测试在服务器端发生故障时的应用程序响应。此外，使用丢包工具测试在客户端发生故障时的应用程序响应。
- 使用 AWS JDBC 驱动程序充分利用 Aurora PostgreSQL 的故障转移功能。有关 AWS JDBC 驱动程序的更多信息及其完整使用说明，请参阅 Amazon Web Services (AWS) JDBC Driver GitHub 存储库。

下文将详述其中这些内容。

主题

- 设置 TCP keepalive 参数
- 配置应用程序以实现快速故障转移
- 测试故障转移
- 快速故障转移 Java 示例

设置 TCP keepalive 参数

在设置 TCP 连接时，会有一组计时器与该连接相关联。当 keepalive 计时器计时到零时，将向连接端点发送一个 keepalive 探测数据包。如果探测收到回复，则可认为连接仍在正常运行。

开启并前瞻性地设置 TCP keepalive 参数，可确保在客户端无法连接到数据库时，任何活动的连接都能很快关闭。然后，应用程序可以连接到新的端点。

确保设置以下 TCP keepalive 参数：

- `tcp_keepalives_idle` 控制一个以秒为单位的时间，如果套接字未发送数据的时间达到此时间段，则发送 keepalive 数据包。不将 ACK 视为数据。建议进行以下设置：

  ```
  tcp_keepalives_idle = 1
  ```

- `tcp_keepalives_interval` 控制发送初始数据包后到发送后续 keepalive 数据包之间的时间，以秒为单位。应使用 `tcp_keepalives_idle` 参数设置此时间。建议进行以下设置：

  ```
  tcp_keepalives_interval = 1
  ```

- `tcp_keepalives_count` 是在应用程序收到通知之前未获确认的 keepalive 探测包的数量。建议进行以下设置：

  ```
  tcp_keepalives_count = 5
  ```

这些设置应在数据库停止响应的五秒内通知应用程序。如果在应用程序的网络中经常出现 keepalive 数据包丢包，则可以设置较大的 `tcp_keepalives_count` 值。这样做允许在不太可靠的网络中使用更多缓冲区，尽管它增加了检测实际故障所需的时间。

在 Linux 上设置 TCP keepalive 参数

1. 测试如何配置 TCP keepalive 参数。

   我们建议使用命令行以及以下命令来执行此操作。这个建议的配置是系统范围的。换句话说，它还会影响在 `SO_KEEPALIVE` 选项开启的情况下创建套接字的所有其他应用程序。

   ```
   sudo sysctl net.ipv4.tcp_keepalive_time=1
   sudo sysctl net.ipv4.tcp_keepalive_intvl=1
   sudo sysctl net.ipv4.tcp_keepalive_probes=5
   ```

2. 在找到对应用程序有效的配置后，必须通过向 `/etc/sysctl.conf` 中添加以下各行（包括所做的任何更改）来保留这些设置：

   ```
   tcp_keepalive_time = 1
   tcp_keepalive_intvl = 1
   tcp_keepalive_probes = 5
   ```

配置应用程序以实现快速故障转移

接下来，您可以找到有关 Aurora PostgreSQL 的几项配置更改的讨论，您可以通过这些更改实现快速故障转移。要了解有关 PostgreSQL JDBC 驱动程序设置和配置的更多信息，请参阅 PostgreSQL JDBC 驱动程序文档。

主题

- 减少 DNS 缓存超时
- 设置 Aurora PostgreSQL 连接字符串以实现快速故障转移
- 用于获取主机字符串的其他选项

减少 DNS 缓存超时

在故障转移后，在应用程序尝试建立连接时，新的 Aurora PostgreSQL 写入器将是以前的读取器。您可以在完全传播 DNS 更新之前，使用 Aurora 只读端点找到该写入器。将 Java DNS 生存时间

（TTL）设置为较小的值（例如小于 30 秒），有助于在以后尝试连接时在读取器节点之间进行循环切换。

```
// Sets internal TTL to match the Aurora RO Endpoint TTL
java.security.Security.setProperty("networkaddress.cache.ttl" , "1");
// If the lookup fails, default to something like small to retry
java.security.Security.setProperty("networkaddress.cache.negative.ttl" , "3");
```

设置 Aurora PostgreSQL 连接字符串以实现快速故障转移

要使用 Aurora PostgreSQL 快速故障转移，请确保应用程序的连接字符串具有一系列主机，而不仅仅是单个主机。下面是可用来连接到 Aurora PostgreSQL 集群的连接字符串示例。在本示例中，主机以粗体显示。

```
jdbc:postgresql://myauroracluster.cluster-c9bfei4hjlrd.us-east-1-
beta.rds.amazonaws.com:5432,
myauroracluster.cluster-ro-c9bfei4hjlrd.us-east-1-beta.rds.amazonaws.com:5432
/postgres?user=<primaryuser>&password=<primarypw>&loginTimeout=2
&connectTimeout=2&cancelSignalTimeout=2&socketTimeout=60
&tcpKeepAlive=true&targetServerType=primary
```

为获得最佳可用性并避免对 RDS API 的依赖，我们建议您保留一个要用来进行连接的文件。该文件包含一个主机字符串，在建立与数据库的连接时，您的应用程序从该字符串进行读取。此主机字符串具有集群可用的所有 Aurora 端点。有关 Aurora 端点的更多信息，请参阅 Amazon Aurora 端点连接。

例如，您可以将端点存储在本地文件中，如下所示。

```
myauroracluster.cluster-c9bfei4hjlrd.us-east-1-beta.rds.amazonaws.com:5432,
myauroracluster.cluster-ro-c9bfei4hjlrd.us-east-1-beta.rds.amazonaws.com:5432
```

应用程序从该文件进行读取，以填充 JDBC 连接字符串的主机部分。重命名数据库集群将导致这些端点发生更改。如果发生此事件，请确保您的应用程序处理该事件。

另一个选择是使用数据库实例节点的列表，如下所示。

```
my-node1.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com:5432,
my-node2.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com:5432,
my-node3.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com:5432,
my-node4.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com:5432
```

这种方法的优点是 PostgreSQL JDBC 连接驱动程序将在此列表中的所有节点中循环查找有效连接。相比之下，当您使用 Aurora 端点时，每次尝试连接时只尝试两个节点。但是，使用数据库实例节点也有缺点。如果在集群中添加或删除节点，并且实例端点列表就变得陈旧，连接驱动程序可能始终找不到要连接的正确主机。

为帮助确保应用程序不用等太长时间就能连接任意一台主机，应前瞻性地设置以下参数：

- `targetServerType` – 控制驱动程序是连接到写入节点，还是读取节点。要确保应用程序仅重新连接到写入节点，请将 `targetServerType` 值设置为 `primary`。

  `targetServerType` 参数的值包括 `primary`、`secondary`、`any` 和 `preferSecondary`。值为 `preferSecondary` 将首先尝试与读取器建立连接。如果无法与读取器建立连接，则它连接到写入器。
- `loginTimeout` – 控制应用程序在建立套接字连接之后等待多长时间才能登录到数据库。
- `connectTimeout` – 控制套接字等待多长时间才能与数据库建立连接。

还可以修改其他应用程序参数以加速连接过程，具体取决于应用程序需要多快建立连接：

- `cancelSignalTimeout` – 在某些应用程序中，可能需要对已经超时的查询发送尽力取消信号。如果此取消信号在故障转移路径中，则考虑前瞻性地设置它，以免将此信号发送给已停止运行的主机。
- `socketTimeout` – 该参数控制套接字在执行读取操作之前等待多长时间。该参数可用作全局“查询超时”，以确保任何查询等待的时间都不会超过该值。一个好的做法是使用两个连接处理程序。一个连接处理程序运行短时查询并将该值设置为较低的值。另一个连接处理程序用于长时间运行的查询，应将此值设置为高得多。采用这种方法，如果服务器出现故障，可以依靠 TCP keepalive 参数来停止长时间运行的查询。
- `tcpKeepAlive` – 开启该参数可确保所设置的 TCP keepalive 参数有效。
- `loadBalanceHosts` – 如果设置为 `true`，则该参数让应用程序连接到从候选主机列表中随机选择的主机。

用于获取主机字符串的其他选项

您可以从多个来源获取主机字符串，包括 `aurora_replica_status` 函数以及使用 Amazon RDS API。

在许多情况下，您需要确定集群的写入器是谁，或者在集群中查找其他读取器节点。为此，您的应用程序可以连接到数据库集群中的任何数据库实例，并查询 `aurora_replica_status`

函数。您可以使用该函数来减少查找要连接到的主机所需的时间。但是，在某些网络故障场景中，aurora_replica_status 函数可能显示过时或不完整的信息。

要确保应用程序能够找到要连接的节点，一个好办法是尝试连接集群写入器端点，然后连接集群读取器端点。请执行此操作，直至建立可读的连接。除非您重命名数据库集群，否则这些端点不会发生更改。这样，一般情况下，您可以将它们保留为应用程序的静态成员，或将它们存储在应用程序会从中读取的资源文件中。

在使用其中一个端点建立连接后，您可以获取有关集群中的其余端点的信息。为此，请调用 aurora_replica_status 函数。例如，以下命令使用 aurora_replica_status 检索信息。

```
postgres=> SELECT server_id, session_id, highest_lsn_rcvd, cur_replay_latency_in_usec,
 now(), last_update_timestamp
FROM aurora_replica_status();

server_id | session_id | highest_lsn_rcvd | cur_replay_latency_in_usec | now |
 last_update_timestamp
-----------+--------------------------------------+-----------------
+----------------------------+-------------------------------+-----------------------
mynode-1 | 3e3c5044-02e2-11e7-b70d-95172646d6ca | 594221001 | 201421 | 2017-03-07
 19:50:24.695322+00 | 2017-03-07 19:50:23+00
mynode-2 | 1efdd188-02e4-11e7-becd-f12d7c88a28a | 594221001 | 201350 | 2017-03-07
 19:50:24.695322+00 | 2017-03-07 19:50:23+00
mynode-3 | MASTER_SESSION_ID | | | 2017-03-07 19:50:24.695322+00 | 2017-03-07
 19:50:23+00
(3 rows)
```

例如，连接字符串的主机部分可能以写入器和读取器集群端点开始，如下所示。

```
myauroracluster.cluster-c9bfei4hjlrd.us-east-1-beta.rds.amazonaws.com:5432,
myauroracluster.cluster-ro-c9bfei4hjlrd.us-east-1-beta.rds.amazonaws.com:5432
```

在这种情况下，应用程序将尝试与任意节点类型（主节点或辅助节点）建立连接。连接应用程序后，一个好办法是先检查节点的读/写状态。为此，请查询 SHOW transaction_read_only 命令的结果。

如果查询的返回值为 OFF，则表示已成功连接到主节点。但是，假设返回值是 ON，并且您的应用程序需要读/写连接。在这种情况下，您可以调用 aurora_replica_status 函数来确定具有 session_id='MASTER_SESSION_ID' 的 server_id。此函数为您提供主节点的名称。您可以将其与 endpointPostfix 结合使用，如下文所述。

当您连接到具有陈旧数据的副本时，请确保您知道这一情况。这时，aurora_replica_status 函数可能会显示过时的信息。您可以在应用程序级别设置陈旧阈值。要检查这一点，您可以查看服务器时间和 last_update_timestamp 值之间的差异。一般来说，您的应用程序应避免由于 aurora_replica_status 函数返回的冲突信息而在两台主机之间来回切换。您的应用程序应该首先尝试所有已知主机，而不是遵从由 aurora_replica_status 返回的数据。

使用 DescribeDBClusters API 列出实例，Java 示例

您可以使用 适用于 Java 的 AWS SDK（特别是 DescribeDBClusters API 操作），以编程方式查找实例列表。

下面的小示例介绍如何通过 Java 8 来实现这一点。

```
AmazonRDS client = AmazonRDSClientBuilder.defaultClient();
DescribeDBClustersRequest request = new DescribeDBClustersRequest()
   .withDBClusterIdentifier(clusterName);
DescribeDBClustersResult result =
rdsClient.describeDBClusters(request);

DBCluster singleClusterResult = result.getDBClusters().get(0);

String pgJDBCEndpointStr =
singleClusterResult.getDBClusterMembers().stream()
   .sorted(Comparator.comparing(DBClusterMember::getIsClusterWriter)
   .reversed()) // This puts the writer at the front of the list
   .map(m -> m.getDBInstanceIdentifier() + endpointPostfix + ":" +
 singleClusterResult.getPort()))
   .collect(Collectors.joining(","));
```

此处，pgJDBCEndpointStr 包含端点的格式化列表，如下所示。

```
my-node1.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com:5432,
my-node2.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com:5432
```

变量 endpointPostfix 可以是您的应用程序设置的常量。或者，您的应用程序可以通过查询集群中单个实例的 DescribeDBInstances API 操作来获得此变量。对于单个客户，该值在 AWS 区域内保持不变。因此，可以省去一次 API 调用，只需将此常量保存在您的应用程序从中进行读取的资源文件中。在上一个示例中，它设置为下面的值。

```
.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com
```

考虑到可用性，如果 API 不响应或响应时间过长，则原定设置使用数据库集群的 Aurora 端点是一个好方法。在端点更新 DNS 记录所用时间之内，应确保其最新。使用端点更新 DNS 记录通常可在不到 30 秒的时间内完成。您可以将此端点存储在应用程序使用的资源文件中。

测试故障转移

在任何情况下，数据库集群都必须包含两个或更多数据库实例。

在服务器端，某些 API 操作可能引起中断，您可以利用此中断来测试应用程序的响应情况：

- FailoverDBCluster – 此操作尝试将数据库集群中的新数据库实例提升为写入器。

  以下代码示例显示了如何可以使用 `failoverDBCluster` 导致中断。有关设置 Amazon RDS 客户端的更多详细信息，请参阅使用 AWS SDK for Java。

  ```
  public void causeFailover() {

      final AmazonRDS rdsClient = AmazonRDSClientBuilder.defaultClient();

      FailoverDBClusterRequest request = new FailoverDBClusterRequest();
      request.setDBClusterIdentifier("cluster-identifier");

      rdsClient.failoverDBCluster(request);
  }
  ```

- RebootDBInstance – 使用此 API 操作无法保证故障转移。但是，它会关闭写入器上的数据库。您可以使用它来测试应用程序如何响应连接断开。`ForceFailover` 参数不适用于 Aurora 引擎。而应使用 `FailoverDBCluster` API 操作。
- ModifyDBCluster – 修改 `Port` 参数会在集群中的节点开始侦听新端口时引起中断。一般而言，您的应用程序可以通过确保只有您的应用程序控制端口更改，来首先对此故障做出响应。此外，请确保它可以适当地更新它所依赖的端点。为此，您可以让某人在 API 级别进行修改时手动更新端口。或者，您可以在应用程序中使用 RDS API 来确定端口是否已更改，从而达到上述目的。
- ModifyDBInstance – 修改 `DBInstanceClass` 参数会导致中断。
- DeleteDBInstance – 删除主实例（写入器）会导致数据库集群中的新数据库实例提升为写入器。

从应用程序或客户端，如果您使用 Linux，您可以测试应用程序如何响应突然的数据包丢失。您可以根据端口、主机或是否使用 iptables 命令发送或接收 TCP keepalive 数据包来执行上述操作。

快速故障转移 Java 示例

以下代码示例说明了应用程序如何设置 Aurora PostgreSQL 驱动程序管理器。

当应用程序需要连接时，它会调用 `getConnection` 函数。调用 `getConnection` 可能找不到有效的主机。一个例子是：找不到写入器，但 `targetServerType` 参数设置为 `primary`。在这种情况下，发出调用的应用程序只需重试调用函数。

为了避免将重试行为推送到应用程序，可以将此重试调用包装到连接池程序中。对于大多数连接池程序，您可以指定 JDBC 连接字符串。因此，您的应用程序可以调用 `getJdbcConnectionString`，然后将其传递给连接池程序。这样做意味着您可以使用 Aurora PostgreSQL 更快地进行故障转移。

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.IntStream;

import org.joda.time.Duration;

public class FastFailoverDriverManager {
   private static Duration LOGIN_TIMEOUT = Duration.standardSeconds(2);
   private static Duration CONNECT_TIMEOUT = Duration.standardSeconds(2);
   private static Duration CANCEL_SIGNAL_TIMEOUT = Duration.standardSeconds(1);
   private static Duration DEFAULT_SOCKET_TIMEOUT = Duration.standardSeconds(5);

   public FastFailoverDriverManager() {
       try {
            Class.forName("org.postgresql.Driver");
       } catch (ClassNotFoundException e) {
            e.printStackTrace();
       }

       /*
         * RO endpoint has a TTL of 1s, we should honor that here. Setting this
 aggressively makes sure that when
         * the PG JDBC driver creates a new connection, it will resolve a new different
 RO endpoint on subsequent attempts
         * (assuming there is > 1 read node in your cluster)
         */
```

```
        java.security.Security.setProperty("networkaddress.cache.ttl" , "1");
       // If the lookup fails, default to something like small to retry
       java.security.Security.setProperty("networkaddress.cache.negative.ttl" , "3");
  }

  public Connection getConnection(String targetServerType) throws SQLException {
      return getConnection(targetServerType, DEFAULT_SOCKET_TIMEOUT);
  }

  public Connection getConnection(String targetServerType, Duration queryTimeout)
throws SQLException {
       Connection conn =
DriverManager.getConnection(getJdbcConnectionString(targetServerType, queryTimeout));

      /*
        * A good practice is to set socket and statement timeout to be the same thing
since both
        * the client AND server will stop the query at the same time, leaving no
running queries
        * on the backend
        */
       Statement st = conn.createStatement();
       st.execute("set statement_timeout to " + queryTimeout.getMillis());
       st.close();

      return conn;
  }

  private static String urlFormat = "jdbc:postgresql://%s"
          + "/postgres"
          + "?user=%s"
          + "&password=%s"
          + "&loginTimeout=%d"
          + "&connectTimeout=%d"
          + "&cancelSignalTimeout=%d"
          + "&socketTimeout=%d"
          + "&targetServerType=%s"
          + "&tcpKeepAlive=true"
          + "&ssl=true"
          + "&loadBalanceHosts=true";
  public String getJdbcConnectionString(String targetServerType, Duration
queryTimeout) {
      return String.format(urlFormat,
               getFormattedEndpointList(getLocalEndpointList()),
```

```
                CredentialManager.getUsername(),
                CredentialManager.getPassword(),
                LOGIN_TIMEOUT.getStandardSeconds(),
                CONNECT_TIMEOUT.getStandardSeconds(),
                CANCEL_SIGNAL_TIMEOUT.getStandardSeconds(),
                queryTimeout.getStandardSeconds(),
                targetServerType
        );
    }

    private List<String> getLocalEndpointList() {
        /*
          * As mentioned in the best practices doc, a good idea is to read a local
 resource file and parse the cluster endpoints.
          * For illustration purposes, the endpoint list is hardcoded here
          */
         List<String> newEndpointList = new ArrayList<>();
         newEndpointList.add("myauroracluster.cluster-c9bfei4hjlrd.us-east-1-
beta.rds.amazonaws.com:5432");
         newEndpointList.add("myauroracluster.cluster-ro-c9bfei4hjlrd.us-east-1-
beta.rds.amazonaws.com:5432");

        return newEndpointList;
    }

    private static String getFormattedEndpointList(List<String> endpoints) {
        return IntStream.range(0, endpoints.size())
                .mapToObj(i -> endpoints.get(i).toString())
                .collect(Collectors.joining(","));
    }
}
```

通过 Aurora PostgreSQL 的集群缓存管理提供故障转移后的快速恢复

要快速恢复 Aurora PostgreSQL 集群中的写入器数据库实例（如果进行了故障转移），请使用 Amazon Aurora PostgreSQL 的集群缓存管理。集群缓存管理可以确保在进行故障转移时保持应用程序的性能。

在典型的故障转移情况下，您可能会在故障转移后看到暂时的但很明显的性能下降。发生性能下降，是因为当故障转移数据库实例启动时，缓冲区缓存是空的。空缓存也称为冷缓存。冷缓存会降低性能，是因为数据库实例必须从更慢的磁盘读取，而不是利用缓冲区缓存中存储的值。

通过集群缓存管理，您将特定的读取器数据库实例设置为故障转移目标。集群缓存管理可以确保指定读取器缓存中的数据与写入器数据库实例缓存中的数据保持同步。预填充值的指定读取器缓存也称为热缓存。如果发生故障转移，在提升为新写入器数据库实例时，指定读取器将立即使用其热缓存中的值。这种方法会为您的应用程序提供好很多的恢复性能。

集群缓存管理要求指定的读取器实例具有与写入器相同类型和大小的实例类（例如 `db.r5.2xlarge` 或 `db.r5.xlarge`）。创建 Aurora PostgreSQL 数据库集群时请记住这一点，以便集群可以在故障转移期间恢复。有关实例类的类型和大小列表，请参阅适用于 Aurora 的数据库实例类的硬件规格。

> **Note**
>
> 作为 Aurora Global Database 一部分的 Aurora PostgreSQL 辅助数据库集群不支持集群缓存管理。对于支持此功能的主集群，建议不要在指定的第 0 层读取器上运行任何工作负载。

目录

- 配置集群缓存管理
  - 启用集群缓存管理
  - 为写入器数据库实例设置提升层优先级
  - 为读取器数据库实例设置提升层优先级
- 监控缓冲区缓存
- 排查 CCM 配置问题

配置集群缓存管理

要配置集群缓存管理，请按顺序执行以下过程。

主题

- 启用集群缓存管理
- 为写入器数据库实例设置提升层优先级
- 为读取器数据库实例设置提升层优先级

> **Note**
>
> 完成这些步骤后，至少需要 1 分钟才能使集群缓存管理完全正常运行。

启用集群缓存管理

要启用集群缓存管理，请执行以下描述的步骤。

控制台

启用集群缓存管理

1. 登录AWS Management Console并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择参数组。
3. 在列表中，为您的 Aurora PostgreSQL 数据库集群选择参数组。

   数据库集群必须使用默认参数组以外的参数组，因为您无法更改默认参数组中的值。
4. 对于 Parameter group actions (参数组操作)，选择 Edit (编辑)。
5. 将 `apg_ccm_enabled` 集群参数的值设置为 1。
6. 选择保存更改。

AWS CLI

要为 Aurora PostgreSQL 数据库集群启用集群缓存管理，请使用包含以下必要参数的 AWS CLI `modify-db-cluster-parameter-group` 命令：

- `--db-cluster-parameter-group-name`
- `--parameters`

Example

对于 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster-parameter-group \
    --db-cluster-parameter-group-name my-db-cluster-parameter-group \
    --parameters "ParameterName=apg_ccm_enabled,ParameterValue=1,ApplyMethod=immediate"
```

对于 Windows：

```
aws rds modify-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name my-db-cluster-parameter-group ^
    --parameters "ParameterName=apg_ccm_enabled,ParameterValue=1,ApplyMethod=immediate"
```

为写入器数据库实例设置提升层优先级

要进行集群缓存管理，请确保 Aurora PostgreSQL 数据库集群的写入器数据库实例的提升优先级为 0 层。提升层优先级 是指定在发生故障后将 Aurora 读取器提升为写入器数据库实例的顺序。有效值为 0–15，0 是第一个优先级，15 是最后一个。有关提升层的更多信息，请参阅 Aurora 数据库集群的容错能力。

控制台

将写入器数据库实例的提升优先级设置为“0 层”

1. 登录AWS Management Console并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Databases (数据库)。
3. 选择 Aurora PostgreSQL 数据库集群的写入器数据库实例。
4. 选择修改。将显示 Modify DB Instance (修改数据库实例) 页面。
5. 在其他配置面板上，选择第 0 层作为故障转移优先级。
6. 选择继续，查看修改摘要。
7. 要在保存更改后立即应用这些更改，请选择 Apply immediately (立即应用)。
8. 选择修改数据库实例以保存更改。

AWS CLI

要使用 AWS CLI 将写入器数据库实例的提升层优先级设置为 0，请调用包含以下必需参数的 modify-db-instance 命令：

- `--db-instance-identifier`
- `--promotion-tier`
- `--apply-immediately`

Example

对于 Linux、macOS 或 Unix：

```
aws rds modify-db-instance \
    --db-instance-identifier writer-db-instance \
    --promotion-tier 0 \
```

```
    --apply-immediately
```

对于 Windows：

```
aws rds modify-db-instance ^
    --db-instance-identifier writer-db-instance ^
    ---promotion-tier 0  ^
    --apply-immediately
```

为读取器数据库实例设置提升层优先级

您必须仅设置一个读取器数据库实例来进行集群缓存管理。要执行此操作，请从实例类和大小与写入器数据库实例相同的 Aurora PostgreSQL 集群选择读取器。例如，如果写入器使用 `db.r5.xlarge`，请选择使用相同类型和大小的实例类的读取器。然后，将其提升层优先级设置为 0。

提升层优先级 是指定在发生故障后将 Aurora 读取器提升为写入器数据库实例的顺序。有效值为 0-15，0 是第一优先级，15 是最后一个。

控制台

将读取器数据库实例的提升优先级设置为“0 层”

1. 登录AWS Management Console并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Databases (数据库)。
3. 选择实例类与写入器数据库实例相同的 Aurora PostgreSQL 数据库集群的 Reader (读取器) 数据库实例。
4. 选择修改。将显示 Modify DB Instance (修改数据库实例) 页面。
5. 在其他配置面板上，选择第 0 层作为故障转移优先级。
6. 选择继续，查看修改摘要。
7. 要在保存更改后立即应用这些更改，请选择 Apply immediately (立即应用)。
8. 选择修改数据库实例以保存更改。

AWS CLI

要使用 AWS CLI 将读取器数据库实例的提升层优先级设置为 0，请调用包含以下必需参数的 modify-db-instance 命令：

- `--db-instance-identifier`
- `--promotion-tier`
- `--apply-immediately`

Example

对于 Linux、macOS 或 Unix：

```
aws rds modify-db-instance \
    --db-instance-identifier reader-db-instance \
    --promotion-tier 0 \
    --apply-immediately
```

对于 Windows：

```
aws rds modify-db-instance ^
    --db-instance-identifier reader-db-instance ^
    ---promotion-tier 0  ^
    --apply-immediately
```

监控缓冲区缓存

在设置集群缓存管理后，您可以监控写入器数据库实例的缓冲区缓存与指定读取器的热缓冲区缓存之间的同步状态。要检查写入器数据库实例和指定读取器数据库实例上的缓冲区缓存内容，请使用 PostgreSQL pg_buffercache 模块。有关更多信息，请参阅 PostgreSQL pg_buffercache 文档。

使用 **aurora_ccm_status** 函数

集群缓存管理还提供 aurora_ccm_status 函数。在写入器数据库实例上使用 aurora_ccm_status 函数可以获取以下有关指定读取器上的缓存预热的进度的信息：

- buffers_sent_last_minute – 在过去一分钟内，有多少缓冲区发送到指定读取器。
- buffers_found_last_minute – 在过去一分钟内识别的频繁访问缓冲区的数量。
- buffers_sent_last_scan – 在上次完整扫描缓冲区缓存期间，有多少缓冲区发送到了指定读取器。

- buffers_found_last_scan – 有多少缓冲区被识别为经常访问，并需要在上一次完整扫描缓冲区缓存期间发送。已在指定读取器上缓存的缓冲区不会发送。
- buffers_sent_current_scan – 在当前的扫描期间，到现在已经发送了多少缓冲区。
- buffers_found_current_scan – 在当前的扫描中，有多少缓冲区被识别为经常访问。
- current_scan_progress – 在当前的扫描期间，到现在有多少缓冲区已被访问。

以下示例演示如何使用 aurora_ccm_status 函数将其一部分输出转换为热速率和热百分比。

```
SELECT buffers_sent_last_minute*8/60 AS warm_rate_kbps,
   100*(1.0-buffers_sent_last_scan::float/buffers_found_last_scan) AS warm_percent
   FROM aurora_ccm_status();
```

排查 CCM 配置问题

启用 apg_ccm_enabled 集群参数后，将在 Aurora PostgreSQL 数据库集群上的写入器数据库实例和一个读取器数据库实例上，以实例级别自动启用集群缓存管理。写入器和读取器实例应使用相同的实例类类型和大小。它们的提升层优先级设置为 0。数据库集群中的其它读取器实例应具有非零的提升层，并且集群缓存管理对于这些实例处于禁用状态。

以下原因可能会导致配置出现问题并禁用集群缓存管理：

- 当没有单个读取器数据库实例设置为提升层 0 时。
- 当写入器数据库实例未设置为提升层 0 时。
- 当多个读取器数据库实例设置为提升层 0 时。
- 当提升层为 0 的写入器和一个读取器数据库实例的实例大小不相同时。

使用池管理 Aurora PostgreSQL 连接流失

当客户端应用程序经常连接和断开连接，以至于 Aurora PostgreSQL 数据库集群响应时间变慢时，则称集群经历连接流失。与 Aurora PostgreSQL 数据库集群端点的每个新连接都会消耗资源，从而减少可用于处理实际工作负载的资源。我们建议您按照以下讨论的一些最佳实践来管理连接流失问题。

对于初学者来说，您可以缩短连接流失率较高的 Aurora PostgreSQL 数据库集群的响应时间。为此，您可以使用 RDS 代理之类的连接池程序。连接池程序 为客户端提供了一个随时可用连接的缓存。几乎 Aurora PostgreSQL 的所有版本均支持 RDS 代理。有关更多信息，请参阅 Amazon RDS Proxy with Aurora PostgreSQL。

如果特定 Aurora PostgreSQL 版本不支持 RDS 代理，则可以使用另一个 PostgreSQL 兼容的连接池程序，例如 PgBouncer。要了解更多信息，请参阅 PGBouncer 网站。

要查看您的 Aurora PostgreSQL 数据库集群能否从连接池中受益，您可以查看 `postgresql.log` 文件以了解连接与断开连接。您还可以使用性能详情来了解您的 Aurora PostgreSQL 数据库集群正在经历的连接流失量。接下来，您将了解有关这两个主题的信息。

记录连接和断开连接

PostgreSQL `log_connections` 和 `log_disconnections` 参数可以捕获与 Aurora PostgreSQL 数据库集群的写入器实例的连接和断开连接。原定设置情况下，这些参数处于关闭状态。要开启这些参数，请使用自定义参数组并通过将值更改为 1 来开启。有关自定义数据库参数组的更多信息，请参阅Amazon Aurora 数据库集群的数据库集群参数组。要检查设置，请使用 psql 和查询连接到 Aurora PostgreSQL 的数据库集群端点，如下所示。

```
labdb=> SELECT setting FROM pg_settings
  WHERE name = 'log_connections';
 setting
---------
on
(1 row)
labdb=> SELECT setting FROM pg_settings
  WHERE name = 'log_disconnections';
setting
---------
on
(1 row)
```

开启这两个参数后，日志会捕获所有新的连接和断开连接。您将看到每个新的授权连接的用户和数据库。在断开连接时，还记录会话持续时间，如以下示例中所示。

```
2022-03-07 21:44:53.978 UTC [16641] LOG: connection authorized: user=labtek
 database=labdb application_name=psql
2022-03-07 21:44:55.718 UTC [16641] LOG: disconnection: session time: 0:00:01.740
 user=labtek database=labdb host=[local]
```

要检查您的应用程序是否存在连接流失，请开启这些参数（如果尚未开启）。然后，通过在实际工作负载和时间段内运行应用程序，在 PostgreSQL 日志中收集数据以进行分析。您可以在 RDS 控制台中查看日志文件。选择您的 Aurora PostgreSQL 数据库集群的写入器实例，然后选择 Logs & events（日志和事件）选项卡。有关更多信息，请参阅 查看和列出数据库日志文件。

也可以从控制台下载日志文件并使用以下命令序列。此序列查找每分钟授权和断开的连接总数。

```
grep "connection authorized\|disconnection: session time:"
 postgresql.log.2022-03-21-16|\
awk {'print $1,$2}' |\
sort |\
uniq -c |\
sort -n -k1
```

在示例输出中，您可以看到授权连接出现峰值，随后从 16:12:10 开始出现断开连接。

```
.....
,.......
.........
5 2022-03-21 16:11:55 connection authorized:
9 2022-03-21 16:11:55 disconnection: session
5 2022-03-21 16:11:56 connection authorized:
5 2022-03-21 16:11:57 connection authorized:
5 2022-03-21 16:11:57 disconnection: session
32 2022-03-21 16:12:10 connection authorized:
30 2022-03-21 16:12:10 disconnection: session
31 2022-03-21 16:12:11 connection authorized:
27 2022-03-21 16:12:11 disconnection: session
27 2022-03-21 16:12:12 connection authorized:
27 2022-03-21 16:12:12 disconnection: session
41 2022-03-21 16:12:13 connection authorized:
47 2022-03-21 16:12:13 disconnection: session
46 2022-03-21 16:12:14 connection authorized:
41 2022-03-21 16:12:14 disconnection: session
24 2022-03-21 16:12:15 connection authorized:
29 2022-03-21 16:12:15 disconnection: session
28 2022-03-21 16:12:16 connection authorized:
24 2022-03-21 16:12:16 disconnection: session
40 2022-03-21 16:12:17 connection authorized:
42 2022-03-21 16:12:17 disconnection: session
40 2022-03-21 16:12:18 connection authorized:
40 2022-03-21 16:12:18 disconnection: session
.....
,.......
.........
1 2022-03-21 16:14:10 connection authorized:
1 2022-03-21 16:14:10 disconnection: session
1 2022-03-21 16:15:00 connection authorized:
```

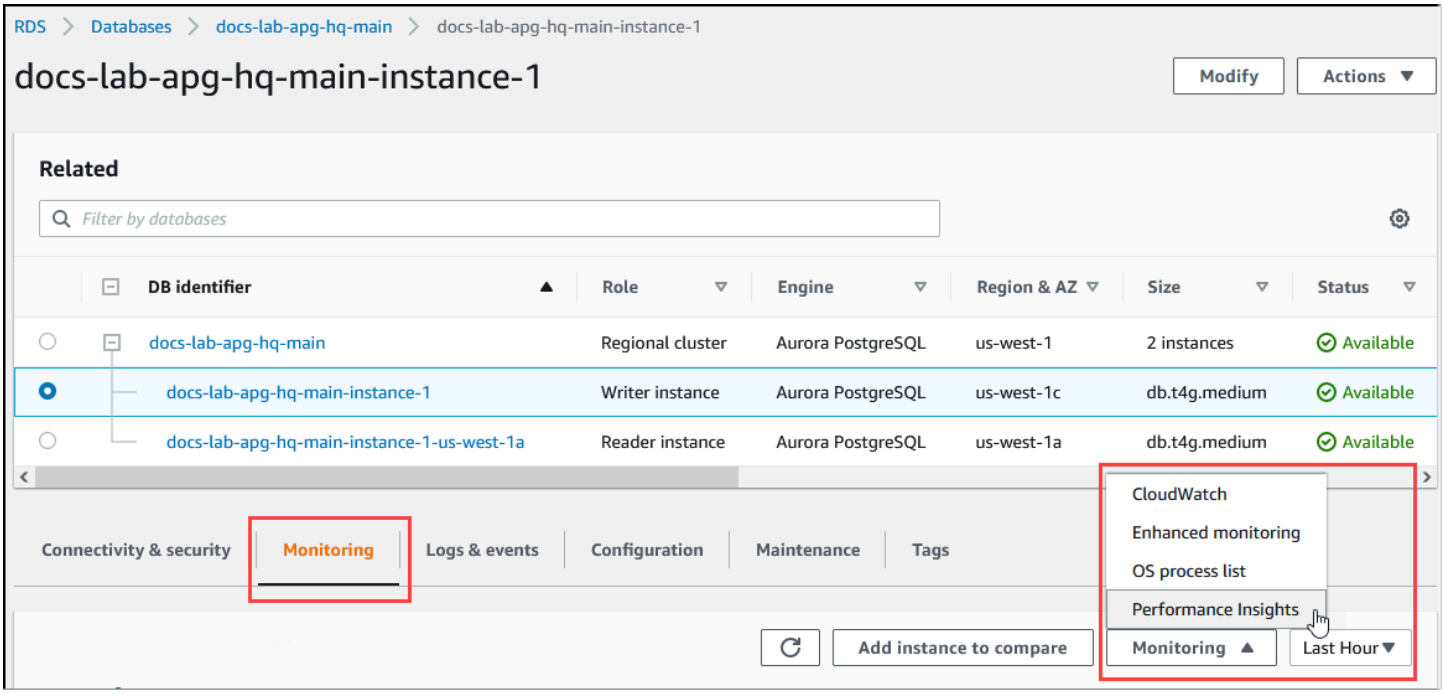
```
1 2022-03-21 16:16:00 connection authorized:
```

有了这些信息，您可以决定您的工作负载是否可以从连接池程序中受益。要进行更详细的分析，您可以使用性能详情。

使用性能详情检测连接流失

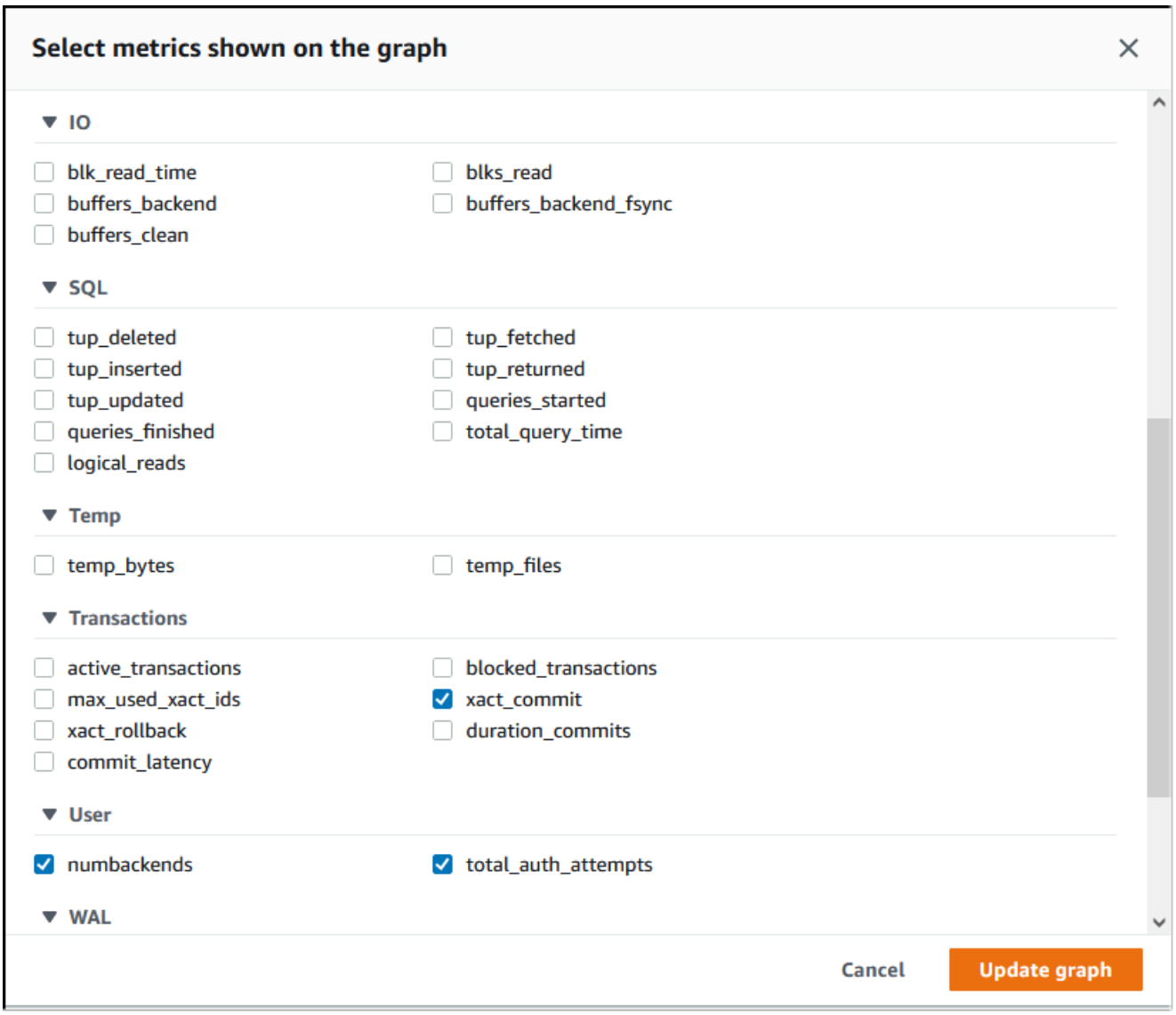
您可以使用性能详情评估 Aurora PostgreSQL 兼容版数据库集群 上的连接流失量。在创建 Aurora PostgreSQL 数据库集群时，性能详情的设置在原定设置情况下处于开启状态。如果您在创建数据库集群时清除了此选项，请修改集群以开启该功能。有关更多信息，请参阅 修改 Amazon Aurora 数据库集群。

在 Aurora PostgreSQL 数据库集群上运行性能详情后，您可以选择要监控的指标。您可以通过控制台中的导航窗格访问性能详情。还可以从 Aurora PostgreSQL 数据库集群的写入器实例的 Monitoring（监控）选项卡访问性能详情，如下图所示。



在性能详情控制台中，选择 Manage metrics（管理指标）。要分析您的 Aurora PostgreSQL 数据库集群的连接和断开连接活动，请选择以下指标。这些都是来自 PostgreSQL 的指标。

- `xact_commit` – 已提交事务的数量。
- `total_auth_attempts` – 每分钟尝试进行身份验证的用户连接的数量。
- `numbackends` – 当前连接到数据库的后端数量。



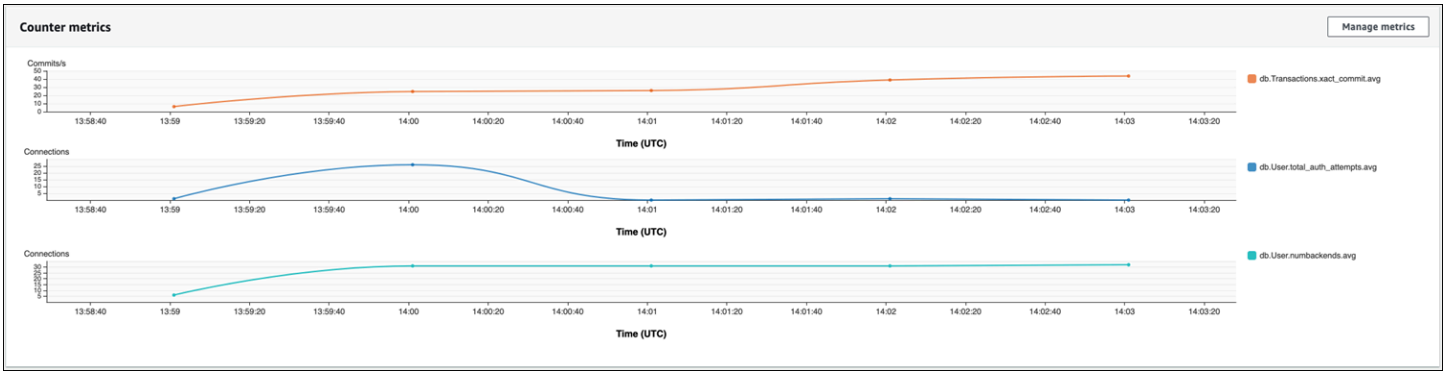
要保存设置并显示连接活动，请选择 Update graph（更新图表）。

在下图中，您可以看到对 100 个用户运行 pgbench 所产生的影响。显示连接的线条呈现一致的向上坡度。要了解有关 pgbench 和如何使用它的更多信息，请参阅 PostgreSQL 文档中的 pgbench。



该图显示，在没有连接池程序的情况下运行少至 100 个用户的工作负载，可能会导致 `total_auth_attempts` 数量在整个工作负载处理过程中显著增加。请注意，最好让 `total_auth_attempts` 保持尽可能接近零。

使用 RDS 代理连接池时，连接尝试次数会在工作负载开始时增加。设置连接池后，平均值下降。事务和后端使用的资源在整个工作负载处理过程中保持一致。



有关对您的 Aurora PostgreSQL 数据库集群使用性能详情的更多信息，请参阅[在 Amazon Aurora 上使用性能详情监控数据库负载](#)。要分析指标，请参阅[使用 Performance Insights 控制面板分析指标](#)。

演示连接池的优势

如前所述，如果您确定您的 Aurora PostgreSQL 数据库集群存在连接流失问题，则可以使用 RDS 代理来提高性能。接下来，您可以了解一个示例，该示例显示了使用和未使用连接池时处理工作负载的差异。该示例使用 pgbench 对事务工作负载进行建模。

与 psql 一样，pgbench 是一个 PostgreSQL 客户端应用程序，您可以从本地客户端计算机上安装和运行该应用程序。也可以从用于管理 Aurora PostgreSQL 数据库集群的 Amazon EC2 实例中安装和运行该应用程序。有关更多信息，请参阅 PostgreSQL 文档中的 pgbench。

要逐步完成此示例，您首先要在数据库中创建 pgbench 环境。以下命令是在指定的数据库中初始化 pgbench 表的基本模板。此示例使用原定设置的主用户账户 `postgres` 进行登录。根据需要，为您的 Aurora PostgreSQL 数据库集群更改此账户。您在集群的写入器实例上的数据库中创建 pgbench 环境。

> **Note**
>
> pgbench 初始化过程删除并重新创建名为 `pgbench_accounts`、`pgbench_branches`|`pgbench_history` 和 `pgbench_tellers` 的表。请确保在初始化 pgbench 时为 *`dbname`* 选择的数据库不使用这些名称。

```
pgbench -U postgres -h db-cluster-instance-1.111122223333.aws-region.rds.amazonaws.com
 -p 5432 -d -i -s 50 dbname
```

对于 pgbench，请指定以下参数。

-d

在 pgbench 运行时输出调试报告。

-h

指定 Aurora PostgreSQL 数据库集群的写入器实例的端点。

-i

初始化数据库中的 pgbench 环境以进行基准测试。

-p

标识用于数据库连接的端口。Aurora PostgreSQL 的原定设置值通常为 5432 或 5433。

-s

指定用于在表中填充行的缩放因子。原定设置的缩放因子为 1，它在 pgbench_branches 表中生成 1 行、在 pgbench_tellers 表中生成 10 行，而在 pgbench_accounts 表中生成 100000 行。

-U

为 Aurora PostgreSQL 数据库集群的写入器实例指定用户账户。

设置 pgbench 环境后，可以在有或没有连接池的情况下运行基准测试。原定设置测试由每个事务的一系列五个 SELECT、UPDATE 和 INSERT 命令组成，这些命令在指定时间内重复运行。您可以指定缩放因子、客户端数量和其他详细信息来对自己的使用案例建模。

例如，以下命令使用 20 个并发连接（-c 选项）运行基准测试 60 秒（-T 选项，表示时间）。-C 选项使测试每次都使用新连接运行，而不是每个客户端会话运行一次。此设置可以为您指示连接开销。

```
pgbench -h docs-lab-apg-133-test-instance-1.c3zr2auzukpa.us-west-1.rds.amazonaws.com -U
 postgres -p 5432 -T 60 -c 20 -C labdb
Password:**********
pgbench (14.3, server 13.3)
  starting vacuum...end.
  transaction type: <builtin: TPC-B (sort of)>
```

```
    scaling factor: 50
    query mode: simple
    number of clients: 20
    number of threads: 1
    duration: 60 s
    number of transactions actually processed: 495
    latency average = 2430.798 ms
    average connection time = 120.330 ms
    tps = 8.227750 (including reconnection times)
```

在不重复使用连接的情况下，在 Aurora PostgreSQL 数据库集群的写入器实例上运行 pgbench 表明每秒只处理大约 8 个事务。在 1 分钟的测试中，总共有 495 个事务。

如果您重复使用连接，Aurora PostgreSQL 数据库集群对用户数量的响应速度快了将近 20 倍。通过重复使用，总共处理了 9,042 个事务。相比之下，在相同的时间和相同数量的用户连接下处理了 495 个事务。区别在于，在后一种情况中重复使用每个连接。

```
pgbench -h docs-lab-apg-133-test-instance-1.c3zr2auzukpa.us-west-1.rds.amazonaws.com -U
 postgres -p 5432 -T 60 -c 20 labdb
Password:*********
pgbench (14.3, server 13.3)
      starting vacuum...end.
      transaction type: <builtin: TPC-B (sort of)>
      scaling factor: 50
      query mode: simple
      number of clients: 20
      number of threads: 1
      duration: 60 s
      number of transactions actually processed: 9042
      latency average = 127.880 ms
      initial connection time = 2311.188 ms
      tps = 156.396765 (without initial connection time)
```

此示例向您表明，池连接可以显著缩短响应时间。有关为您的 Aurora PostgreSQL 数据库集群设置 RDS 代理的信息，请参阅适用于 Aurora 的 Amazon RDS 代理。

PostgreSQL 中的失效连接处理

当客户端应用程序已放弃或异常终止，但数据库会话在服务器上仍保持活动状态时，就会出现失效连接。当客户端进程崩溃或意外终止，但没有正确地关闭其数据库连接或取消正在进行的请求时，通常会出现这种情况。

当服务器进程处于空闲状态或试图向客户端发送数据时，PostgreSQL 可以高效地识别和清理失效连接。但是，对于处于空闲状态、等待客户端输入或当前正在运行查询的会话，检测过程颇具挑战性。为了处理这些情况，PostgreSQL 提供了 `tcp_keepalives_*`、`tcp_user_timeout` 和 `client_connection_check_interval` 参数。

主题

- 了解 TCP keepalive
- Aurora PostgreSQL 中的关键 TCP keepalive 参数
- TCP keepalive 设置的使用案例
- 最佳实践

了解 TCP keepalive

TCP Keepalive 是一种协议级机制，有助于保持和验证连接完整性。每个 TCP 连接都保持内核级别的设置，这些设置控制着 keepalive 行为。当 keepalive 计时器到期时，系统会执行以下操作：

- 发送一个没有数据且设置了 ACK 标志的探测数据包。
- 根据 TCP/IP 规范，期待来自远程端点的响应。
- 根据响应或无响应来管理连接状态。

Aurora PostgreSQL 中的关键 TCP keepalive 参数

参数	描述	默认值
`tcp_keepalives_idle`	Specifies number of seconds of inactivity before sending keepalive message.	300
`tcp_keepalives_interval`	Specifies number of seconds between retransmissions of unacknowledged keepalive messages.	30
`tcp_keepalives_count`	Maximum lost keepalive messages before declaring connection dead	2

参数	描述	默认值
tcp_user_timeout	Specifies how long (in Milliseconds) unacknowledged data can remain before forcibly closing the connection.	0
client_connection_check_interval	Sets the interval (in Milliseconds) for checking client connection status during long-running queries. This ensures quicker detection of closed connections.	0

TCP keepalive 设置的使用案例

使空闲会话保持连接状态

为防止空闲连接因处于非活动状态而被防火墙或路由器终止：

- 配置 tcp_keepalives_idle 定期发送 keepalive 数据包。

检测失效连接

要迅速检测失效连接：

- 调整 tcp_keepalives_idle、tcp_keepalives_interval 和 tcp_keepalives_count。例如，使用 Aurora PostgreSQL 默认值，检测到失效连接大约需要一分钟（2 次探测 × 30 秒）。降低这些值可以加快检测速度。
- 使用 tcp_user_timeout 来指定等待确认的最长时间。

TCP keepalive 设置有助于内核检测失效连接，但在使用套接字之前，PostgreSQL 可能无法执行操作。如果会话正在运行长时间的查询，则可能只有在查询完成后才能检测到失效连接。在 PostgreSQL 14 及更高版本中，client_connection_check_interval 可以通过在查询执行期间定期轮询套接字来加快失效连接检测。

最佳实践

- 设置合理的 keepalive 间隔：调整 `tcp_user_timeout`、`tcp_keepalives_idle`、`tcp_keepalives_count` 和 `tcp_keepalives_interval`，以平衡检测速度与资源用量。
- 针对环境进行优化：使设置与网络行为、防火墙策略和会话需求相一致。
- 利用 PostgreSQL 功能：在 PostgreSQL 14 及更高版本中使用 `client_connection_check_interval` 来进行高效的连接检查。

调整 Aurora PostgreSQL 的内存参数

在 Amazon Aurora PostgreSQL 中，您可以使用多个参数来控制用于各种处理任务的内存量。如果任务占用的内存超过为给定参数设置的内存量，Aurora PostgreSQL 会使用其他资源进行处理，例如写入磁盘。这可能会导致您的 Aurora PostgreSQL 数据库集群变慢或可能停止，并出现内存不足错误。

每个内存参数的原定设置通常可以处理其预期的处理任务。但是，您也可以调整 Aurora PostgreSQL 数据库集群 的内存相关参数。进行此调整是为了确保分配足够的内存来处理您的特定工作负载。

在下文中，您可以了解有关控制内存管理的参数的信息。您还可以学习如何评估内存利用率。

检查和设置参数值

您可以设置的用于管理内存和评估 Aurora PostgreSQL 数据库集群的内存使用情况的参数包括以下各项：

- `work_mem` – 指定 Aurora PostgreSQL 数据库集群在写入临时磁盘文件之前用于内部排序操作和哈希表的内存量。
- `log_temp_files` – 记录临时文件创建、文件名和大小。启用此参数后，将为创建的每个临时文件存储一个日志条目。启用此参数可查看您的 Aurora PostgreSQL 数据库集群需要写入磁盘的频率。收集了有关 Aurora PostgreSQL 数据库集群的临时文件生成的信息后，再次将其关闭，以避免过多的日志记录。
- `logical_decoding_work_mem` – 指定每个内部重新排序缓冲区在溢出到磁盘之前要使用的内存量（以千字节为单位）。此内存用于逻辑解码，这是用于创建副本的过程。此过程是通过将数据从预写日志（WAL）文件转换为目标所需的逻辑流输出来完成的。

  此参数的值会创建单个缓冲区，其大小相当于为每个复制连接指定的大小。原定设置情况下，此值为 65536KB。填满该缓冲区后，多余的部分将作为文件写入到磁盘。为了最大限度地减少磁盘活动，可以将该参数的值设置为远高于 `work_mem` 的值。

这些都是动态参数，因此您可以针对当前会话更改它们。为此，请通过 psql 并使用 SET 语句连接到 Aurora PostgreSQL 数据库集群，如下所示。

```
SET parameter_name TO parameter_value;
```

会话设置只在会话持续时间内有效。当会话结束时，该参数将恢复为其在数据库集群参数组 中的设置。在更改任何参数之前，首先通过查询 pg_settings 表检查当前值，如下所示。

```
SELECT unit, setting, max_val
   FROM pg_settings WHERE name='parameter_name';
```

例如，要查找 work_mem 参数的值，请连接到 Aurora PostgreSQL 数据库集群的写入器实例，并运行以下查询。

```
SELECT unit, setting, max_val, pg_size_pretty(max_val::numeric)
  FROM pg_settings WHERE name='work_mem';
unit  | setting  | max_val    | pg_size_pretty
------+----------+------------+----------------
 kB   | 1024     | 2147483647| 2048 MB
(1 row)
```

要更改参数设置以使其保持不变，需要使用自定义数据库集群参数组 。使用 SET 语句通过这些参数的不同值实施 Aurora PostgreSQL 数据库集群后，您可以创建一个自定义参数组并应用于 Aurora PostgreSQL 数据库集群 。有关更多信息，请参阅 Amazon Aurora 的参数组。

了解工作内存参数

工作内存参数（work_mem）指定 Aurora PostgreSQL 可用于处理复杂查询的最大内存量。复杂查询包括那些涉及排序或分组操作的查询，换句话说，使用以下子句的查询：

- ORDER BY
- DISTINCT
- GROUP BY
- JOIN（MERGE 和 HASH）

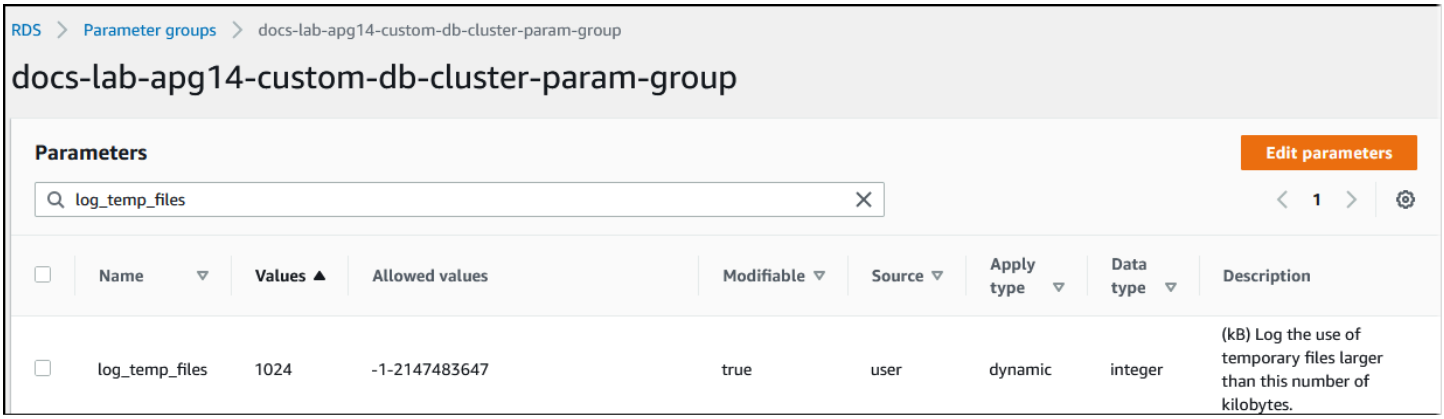
查询计划程序间接影响您的 Aurora PostgreSQL 数据库集群使用工作内存的方式。查询计划程序生成用于处理 SQL 语句的执行计划。给定的计划可能会将复杂的查询分解为多个可以并行运行的工作单

元。在可能的情况下，Aurora PostgreSQL 在为每个并行进程写入磁盘之前，使用在 work_mem 参数中为每个会话指定的内存量。

多个数据库用户同时运行多个操作并且并行生成多个工作单元，可能会耗尽 Aurora PostgreSQL 数据库集群的已分配工作内存。这可能导致过多的临时文件创建和磁盘 I/O，或者更糟糕的是，它可能导致内存不足错误。

识别临时文件使用情况

每当处理查询所需的内存超过在 work_mem 参数中指定的值时，工作数据将卸载到磁盘上的临时文件中。通过开启 log_temp_files 参数，可以查看这种情况发生的频率。原定设置情况下，此参数处于禁用状态（设置为 -1）。要捕获所有临时文件信息，请将此参数设置为 0。将 log_temp_files 设置为任何其他正整数，以捕获等于或大于该数据量（以 KB 为单位）的文件的临时文件信息。在下图中，您可以看到一个来自 AWS Management Console的示例。



配置临时文件日志记录后，您可以使用自己的工作负载进行测试，看看您的工作内存设置是否足够。您还可以使用 pgbench 来模拟工作负载，pgbench 是一款来自 PostgreSQL 社区的简单基准测试应用程序。

以下示例通过创建运行测试所需的表和行来初始化 (-i) pgbench。在本例中，缩放系数（-s 50）在 labdb 数据库中的 pgbench_branches 表中创建 50 行，在 pgbench_tellers 表中创建 500 行，在 pgbench_accounts 表中创建 5000000 行。

```
pgbench -U postgres -h your-cluster-instance-1.111122223333.aws-regionrds.amazonaws.com
 -p 5432 -i -s 50 labdb
Password:
dropping old tables...
NOTICE:  table "pgbench_accounts" does not exist, skipping
NOTICE:  table "pgbench_branches" does not exist, skipping
NOTICE:  table "pgbench_history" does not exist, skipping
```

```
NOTICE:  table "pgbench_tellers" does not exist, skipping
creating tables...
generating data (client-side)...
5000000 of 5000000 tuples (100%) done (elapsed 15.46 s, remaining 0.00 s)
vacuuming...
creating primary keys...
done in 61.13 s (drop tables 0.08 s, create tables 0.39 s, client-side generate 54.85
 s, vacuum 2.30 s, primary keys 3.51 s)
```

初始化环境后，可以针对特定时间（-T）和客户端数量（-c）运行基准测试。此示例还使用 -d 选项，以在 Aurora PostgreSQL 数据库集群处理事务时输出调试信息。

```
pgbench -h -U postgres your-cluster-instance-1.111122223333.aws-regionrds.amazonaws.com
 -p 5432 -d -T 60 -c 10 labdb
Password:*******
pgbench (14.3)
starting vacuum...end.
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 50
query mode: simple
number of clients: 10
number of threads: 1
duration: 60 s
number of transactions actually processed: 1408
latency average = 398.467 ms
initial connection time = 4280.846 ms
tps = 25.096201 (without initial connection time)
```

有关 pgbench 的更多信息，请参阅 PostgreSQL 文档中的 pgbench。

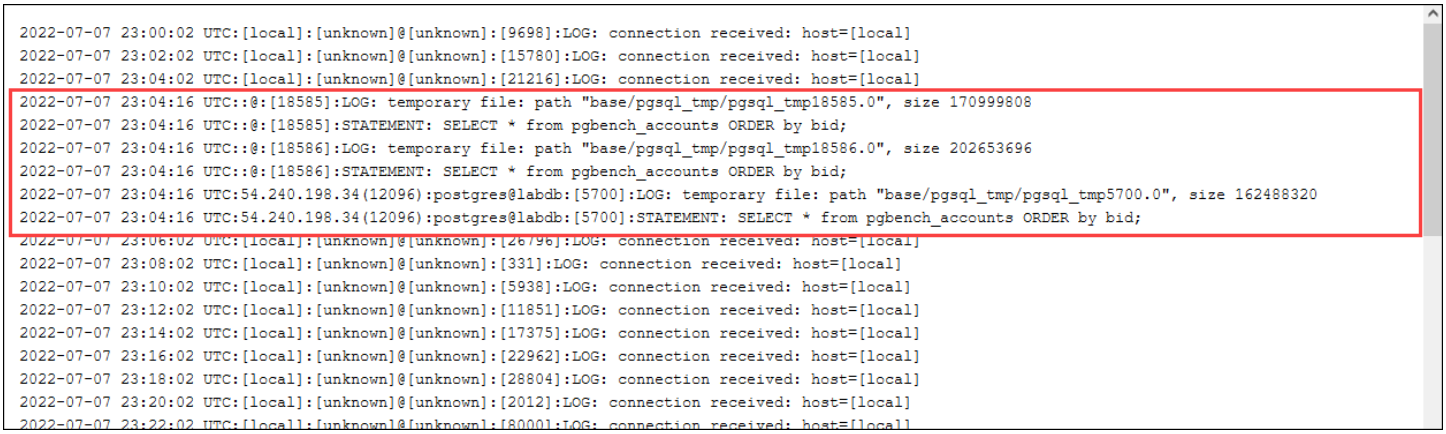
您可以使用 psql 元命令（\d）列出 pgbench 创建的表、视图和索引等关系。

```
labdb=>  \d pgbench_accounts
 Table "public.pgbench_accounts"
  Column  |     Type      | Collation | Nullable | Default
----------+---------------+-----------+----------+---------
 aid      | integer       |           | not null |
 bid      | integer       |           |          |
 abalance | integer       |           |          |
 filler   | character(84) |           |          |
Indexes:
    "pgbench_accounts_pkey" PRIMARY KEY, btree (aid)
```

如输出中所示，pgbench_accounts 表是按 aid 列编制索引的。为确保此下一个查询使用工作内存，请查询任何非索引列，如以下示例中所示的列。

```
postgres=> SELECT * FROM pgbench_accounts ORDER BY bid;
```

检查日志中是否有临时文件。为此，请打开 AWS Management Console，选择 Aurora PostgreSQL 数据库集群实例，然后选择 Logs & Events（日志和事件）选项卡。在控制台中查看这些日志，或下载以供进一步分析。如下图所示，处理查询所需的临时文件的大小表明，您应该考虑增加为 work_mem 参数指定的数量。



您可以根据您的运营需求为个人和团体配置不同的参数。例如，可以针对名为 dev_team 的角色将 work_mem 参数设置为 8 GB。

```
postgres=> ALTER ROLE dev_team SET work_mem='8GB';
```

对 work_mem 使用此设置，将向作为 dev_team 角色成员的任何角色分配最多 8 GB 的工作内存。

使用索引缩短响应时间

如果查询返回结果的时间过长，则可以验证索引是否按预期使用。首先，打开 psql 元命令 \timing，如下所示。

```
postgres=>  \timing on
```

开启计时后，使用简单的 SELECT 语句。

```
postgres=> SELECT COUNT(*) FROM
  (SELECT * FROM pgbench_accounts
  ORDER BY bid)
```

```
  AS accounts;
count
-------
5000000
(1 row)
Time: 3119.049 ms (00:03.119)
```

如输出所示，此查询仅用 3 秒就完成了。要缩短响应时间，请为 pgbench_accounts 创建索引，如下所示。

```
postgres=> CREATE INDEX ON pgbench_accounts(bid);
CREATE INDEX
```

重新运行查询，注意响应时间更快。在此示例中，查询的完成速度快了大约 5 倍，大约需要半秒钟。

```
postgres=>  SELECT COUNT(*) FROM (SELECT * FROM pgbench_accounts ORDER BY bid) AS
 accounts;
 count
-------
 5000000
(1 row)
Time: 567.095 ms
```

调整工作内存以进行逻辑解码

自从在 PostgreSQL 版本 10 中引入以来，逻辑复制已在所有版本的 Aurora PostgreSQL 中可用。配置逻辑复制时，还可以设置 logical_decoding_work_mem 参数，以指定逻辑解码过程可用于解码和流式传输过程的内存量。

在逻辑解码期间，预写日志（WAL）记录转换为 SQL 语句，然后这些语句将发送到另一个目标以用于逻辑复制或其他任务。当事务写入 WAL 并随后进行转换时，整个事务必须符合为 logical_decoding_work_mem 指定的值。原定设置情况下，该参数设置为 65536KB。任何溢出都将写入磁盘。这意味着，必须先从磁盘中重新读取溢出的数据，然后才能将其发送到目的地，从而减慢了整个过程。

您可以使用 aurora_stat_file 函数评估在特定时间点当前工作负载中的事务溢出量，如下例所示。

```
SELECT split_part (filename, '/', 2)
   AS slot_name, count(1) AS num_spill_files,
   sum(used_bytes) AS slot_total_bytes,
```

```
   pg_size_pretty(sum(used_bytes)) AS slot_total_size
   FROM aurora_stat_file()
   WHERE filename like '%spill%'
   GROUP BY 1;
  slot_name | num_spill_files | slot_total_bytes | slot_total_size
------------+-----------------+------------------+-----------------
 slot_name  |       590       |      411600000   | 393 MB
(1 row)
```

此查询返回调用该查询时 Aurora PostgreSQL 数据库集群上溢出文件的数量和大小。运行时间较长的工作负载在磁盘上可能没有任何溢出文件。要分析长时间运行的工作负载，我们建议您创建一个表，以便在工作负载运行时捕获溢出文件信息。您可以按如下所示创建表。

```
CREATE TABLE spill_file_tracking AS
    SELECT now() AS spill_time,*
    FROM aurora_stat_file()
    WHERE filename LIKE '%spill%';
```

要查看在逻辑复制期间如何使用溢出文件，请设置发布者和订阅者，然后开始简单复制。有关更多信息，请参阅 为 Aurora PostgreSQL 数据库集群设置逻辑复制。随着复制进行，您可以创建一个任务以从 `aurora_stat_file()` 溢出文件函数中捕获结果集，如下所示。

```
INSERT INTO spill_file_tracking
  SELECT now(),*
  FROM aurora_stat_file()
  WHERE filename LIKE '%spill%';
```

使用以下 psql 命令每秒运行一次任务。

```
\watch 0.5
```

在任务运行时，从另一个 psql 会话连接到写入器实例。使用以下一系列语句运行超出内存配置并导致 Aurora PostgreSQL 创建溢出文件的工作负载。

```
labdb=> CREATE TABLE my_table (a int PRIMARY KEY, b int);
CREATE TABLE
labdb=> INSERT INTO my_table SELECT x,x FROM generate_series(0,10000000) x;
INSERT 0 10000001
labdb=> UPDATE my_table SET b=b+1;
UPDATE 10000001
```

完成这些语句可能需要几分钟时间。完成后，同时按下 Ctrl 键和 C 键以停止监控功能。然后，使用以下命令创建一个表，以保存有关 Aurora PostgreSQL 数据库集群的溢出文件使用情况的信息。

```
SELECT spill_time, split_part (filename, '/', 2)
    AS slot_name, count(1)
    AS spills, sum(used_bytes)
    AS slot_total_bytes, pg_size_pretty(sum(used_bytes))
    AS slot_total_size FROM spill_file_tracking
  GROUP BY 1,2 ORDER BY 1;
                  spill_time | slot_name             | spills | slot_total_bytes |
 slot_total_size
-------------------------------+-----------------------+--------+-----------------
+-----------------
2022-04-15 13:42:52.528272+00 | replication_slot_name | 1      | 142352280        | 136
 MB
2022-04-15 14:11:33.962216+00 | replication_slot_name | 4      | 467637996        | 446
 MB
2022-04-15 14:12:00.997636+00 | replication_slot_name | 4      | 569409176        | 543
 MB
2022-04-15 14:12:03.030245+00 | replication_slot_name | 4      | 569409176        | 543
 MB
2022-04-15 14:12:05.059761+00 | replication_slot_name | 5      | 618410996        | 590
 MB
2022-04-15 14:12:07.22905+00  | replication_slot_name | 5      | 640585316        | 611
 MB
(6 rows)
```

输出显示，运行该示例创建了五个使用了 611MB 内存的溢出文件。为了避免写入磁盘，建议将 `logical_decoding_work_mem` 参数设置为下一个最大内存大小 1024。

使用 Amazon CloudWatch 指标分析 Aurora PostgreSQL 的资源使用情况

Aurora 以 1 分钟为间隔自动将指标数据发送到 CloudWatch。您可以使用 CloudWatch 指标分析 Aurora PostgreSQL 的资源使用情况。您可以使用指标评估网络吞吐量和网络使用情况。

使用 CloudWatch 评估网络吞吐量

当您的系统使用情况接近实例类型的资源限制时，处理速度可能会变慢。您可以使用 CloudWatch Logs Insights（日志洞察）监控您的存储资源使用情况，并确保有充足的可用资源。必要时，您可以将数据库实例修改为更大的实例类。

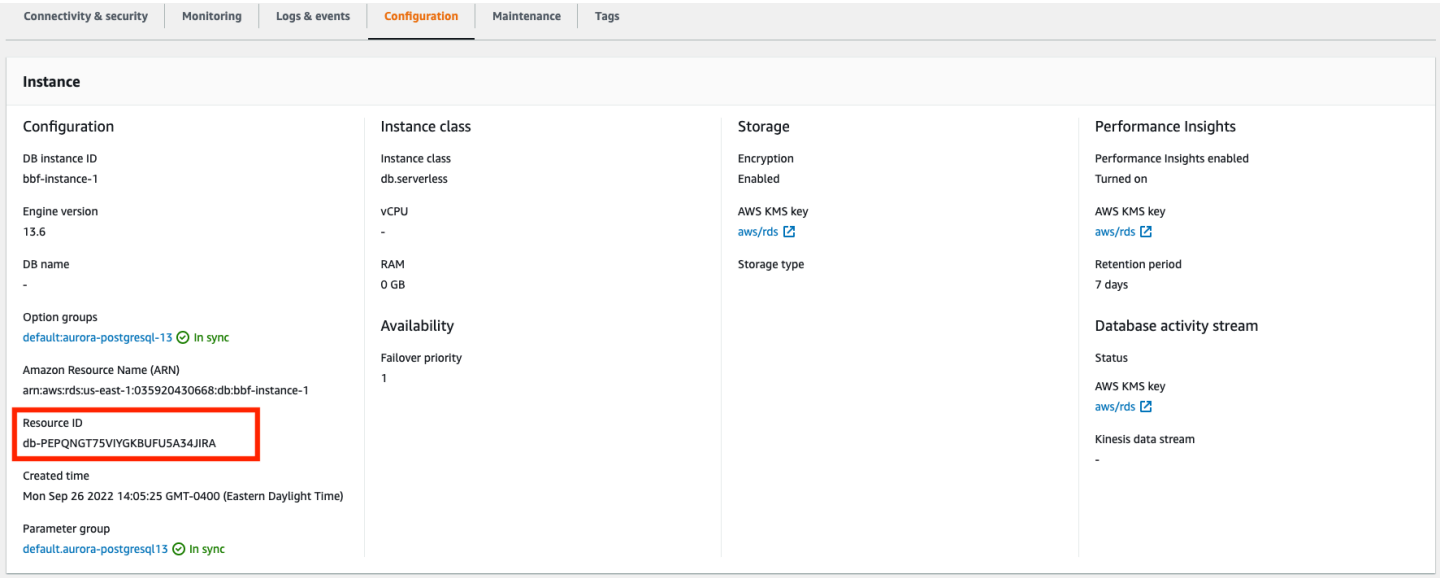
Aurora 存储处理可能很慢，原因是：

- 客户端与数据库实例之间的网络带宽不足。
- 存储子系统的网络带宽不足。
- 对于您的实例类型而言，工作负载很大。

您可以查询 CloudWatch Logs Insights（日志洞察）以生成 Aurora 存储资源使用情况的图表来监控资源。该图显示 CPU 利用率和指标，可帮助您决定是否纵向扩展到更大的实例大小。有关 CloudWatch Logs Insights（日志洞察）的查询语法的信息，请参阅 CloudWatch Logs Insights 查询语法。

要使用 CloudWatch，您需要将 Aurora PostgreSQL 日志文件导出到 CloudWatch。您还可以修改现有集群，以将日志导出到 CloudWatch。有关将日志导出到 CloudWatch 的信息，请参阅 开启将日志发布到 Amazon CloudWatch 的选项。

您需要数据库实例的 Resource ID（资源 ID）才能查询 CloudWatch Logs Insights（日志洞察）。Resource ID（资源 ID）可在控制台的 Configuration（配置）选项卡中找到：



要在日志文件中查询资源存储指标，请执行以下操作：

1. 通过 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。

   将显示 CloudWatch 概览主页。
2. 如果需要，更改 AWS 区域。从导航栏中，选择您的 AWS 资源所在的 AWS 区域。有关更多信息，请参阅区域和端点。
3. 在导航窗格中，选择 Logs（日志），然后选择 Logs Insights（日志洞察）。

   将出现 Logs Insights（日志洞察）页面。

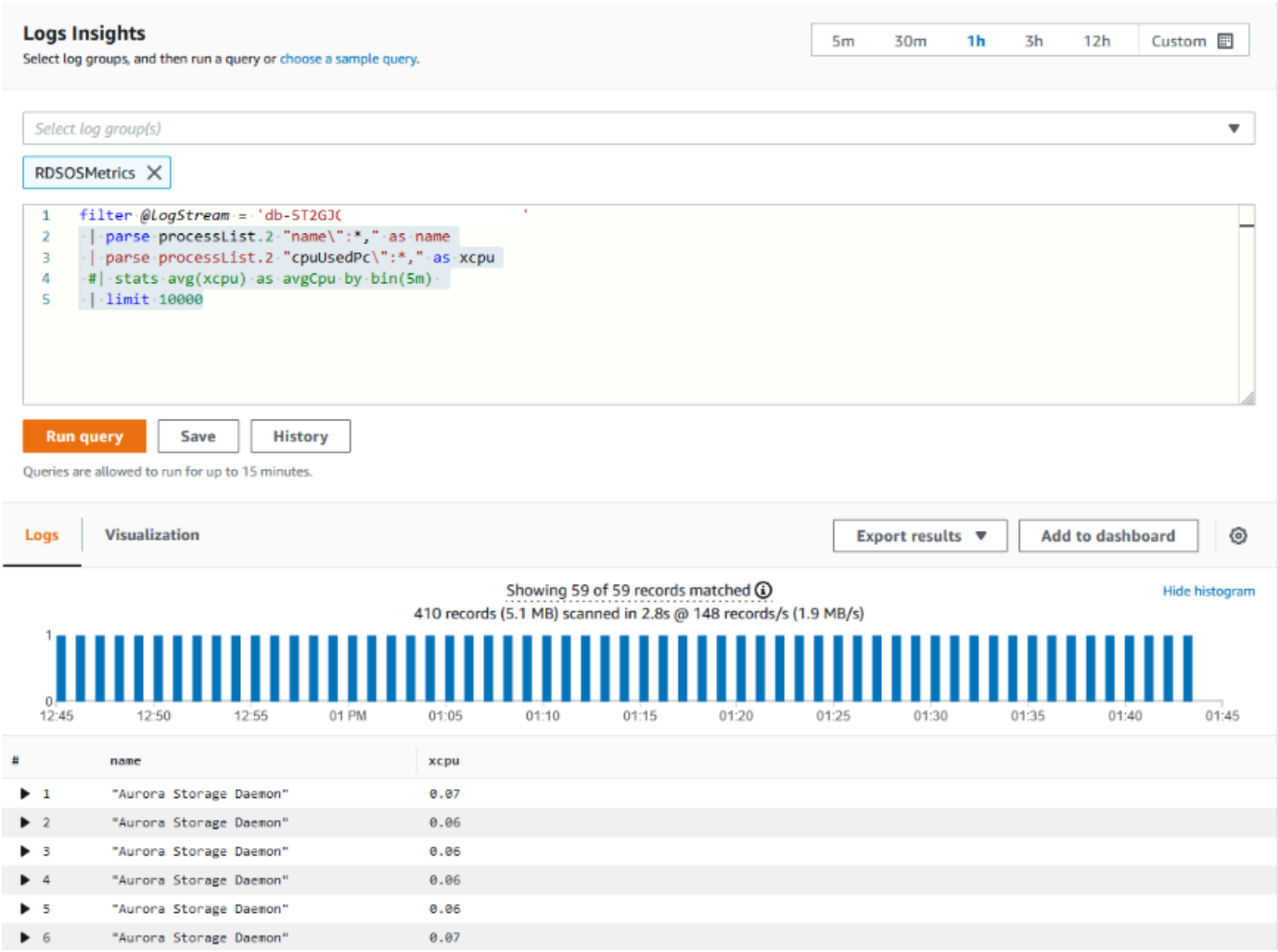
4. 从下拉列表中，选择日志文件进行分析。
5. 在字段中输入以下查询，将 <resource ID> 替换为数据库集群的资源 ID：

```
filter @logStream = <resource ID> | parse @message "\"Aurora Storage Daemon\"*memoryUsedPc\":*,\"cpuUsedPc\":*," as a,memoryUsedPc,cpuUsedPc | display memoryUsedPc,cpuUsedPc #| stats avg(xcpu) as avgCpu by bin(5m) | limit 10000
```

6. 单击 Run query（运行查询）。

   将显示存储利用率图表。

   下图提供了 Logs Insights（日志洞察）页面和图表显示。



使用 CloudWatch 指标评估 Aurora PostgreSQL 数据库实例使用情况

您可以使用 CloudWatch 指标来观察您的实例吞吐量，并发现您的实例类是否为您的应用程序提供了足够的资源。有关数据库实例类限制的信息，请访问 适用于 Aurora 的数据库实例类的硬件规格 并找到数据库实例类的规格以了解您的网络性能。

如果您的数据库实例使用情况接近实例类限制，则性能可能会开始变慢。CloudWatch 指标可以证实这种情况，因此您可以计划手动纵向扩展到更大的实例类。

结合使用以下 CloudWatch 指标值，看看您是否接近实例类限制：

- NetworkThroughput –Aurora 数据库集群中每个实例的客户端接收和发送的网络吞吐量。此吞吐量值不包括数据库集群中的实例与集群卷之间的网络流量。
- StorageNetworkThroughput – Aurora 数据库集群中每个实例从 Aurora 存储子系统接收以及发送到此子系统的网络吞吐量。

将 NetworkThroughput 与 StorageNetworkThroughput 相加，以找出 Aurora 数据库集群中每个实例从 Aurora 存储子系统接收和发送到此系统的网络吞吐量。您的实例的实例类限制应大于这两个组合指标的总和。

在发送和接收时，您可以使用以下指标来查看来自客户端应用程序的网络流量的更多详细信息：

- NetworkReceiveThroughput – Aurora PostgreSQL 数据库集群中每个实例从客户端接收的网络吞吐量。此吞吐量不包括数据库集群中的实例与集群卷之间的网络流量。
- NetworkTransmitThroughput – Aurora 数据库集群中每个实例发送到客户端的网络吞吐量。此吞吐量不包括数据库集群中的实例与集群卷之间的网络流量。
- StorageNetworkReceiveThroughput – 数据库集群中每个实例从 Aurora 存储子系统接收的网络吞吐量。
- StorageNetworkTransmitThroughput – Aurora MySQL 数据库集群中每个实例发送到 Aurora 存储子系统的网络吞吐量。

将所有这些指标相加，以评估您的网络使用量与实例类限制的对比情况。实例类限制应大于这些组合指标的总和。

网络限制和存储的 CPU 利用率是相互的。当网络吞吐量增加时，CPU 利用率也会增加。监控 CPU 和网络使用情况可提供有关资源耗尽的方式和原因的信息。

为了帮助显著减少网络使用量，您可以考虑：

- 使用更大的实例类。
- 使用 pg_partman 分区策略。
- 批量划分写入请求以减少总体事务量。
- 将只读工作负载定向到只读实例。
- 删除任何未使用的索引。
- 检查是否有臃肿的对象和 VACUUM。如果出现严重臃肿，请使用 PostgreSQL 扩展 pg_repack。有关 pg_repack 的详细信息，请参阅 Reorganize tables in PostgreSQL databases with minimal locks（使用最小锁定重新组织 PostgreSQL 数据库中的表）。

使用逻辑复制对 Aurora PostgreSQL 执行主要版本升级

使用逻辑复制和 Aurora 快速克隆，您可以执行使用当前版本 Aurora PostgreSQL 数据库的主要版本升级，同时将不断变化的数据逐渐迁移到新的主要版本数据库。这种停机时间短的升级过程称为蓝绿升级。数据库的当前版本称为“蓝色”环境，新的数据库版本称为“绿色”环境。

Aurora 快速克隆通过拍摄源数据库的快照来完全加载现有数据。快速克隆使用构建在 Aurora 存储层之上的写入时复制协议，这可让您在短时间内创建数据库的克隆。升级到大型数据库时，此方法非常有效。

PostgreSQL 中的逻辑复制会跟踪您的数据更改，并将其从初始实例传输到并行运行的新实例，直到您移至 PostgreSQL 的更新版本。逻辑复制使用发布和订阅模型。有关 Aurora PostgreSQL 逻辑复制的更多信息，请参阅使用 Amazon Aurora PostgreSQL 进行复制。

> **Tip**
>
> 您可以使用托管式 Amazon RDS 蓝绿部署功能，最大限度地减少主要版本升级所需的停机时间。有关更多信息，请参阅 使用 Amazon Aurora 蓝绿部署进行数据库更新。

主题

- 要求
- 限制
- 设置和检查参数值
- 将 Aurora PostgreSQL 升级到新的主要版本

- 执行升级后任务

要求

必须满足以下要求，才能执行这一停机时间短的升级过程：

- 您必须拥有 rds_superuser 权限。
- 您打算升级的 Aurora PostgreSQL 数据库集群必须运行支持的版本，该版本可以使用逻辑复制执行主要版本升级。确保将任何次要版本更新和补丁应用于数据库集群。以下版本的 Aurora PostgreSQL 支持此技术中使用的 `aurora_volume_logical_start_lsn` 函数：
  - 15.2 及更高的 15 版本
  - 14.3 及更高的 14 版本
  - 13.6 及更高的 13 版本
  - 12.10 及更高的 12 版本
  - 11.15 及更高的 11 版本
  - 10.20 及更高的 10 版本

  有关 `aurora_volume_logical_start_lsn` 函数的更多信息，请参阅aurora_volume_logical_start_lsn。
- 您的所有表都必须具有主键或包含 PostgreSQL 标识列。
- 为您的 VPC 配置安全组，以允许两个 Aurora PostgreSQL 数据库集群（包括新旧集群）之间的入站和出站访问。您可以授予对特定的无类别域间路由（CIDR）范围的访问权限，或对 VPC 或对等 VPC 中另一个安全组的访问权限。（对等 VPC 要求 VPC 对等连接。）

> **Note**
>
> 有关配置和管理正在运行的逻辑复制场景所需的权限的详细信息，请参阅 PostgreSQL 核心文档。

限制

当您在 Aurora PostgreSQL 数据库集群上执行停机时间短的升级以将其升级到新的主要版本时，您使用的是原生 PostgreSQL 逻辑复制功能。它具有与 PostgreSQL 逻辑复制相同的功能和限制。有关更多信息，请参阅 PostgreSQL 逻辑复制。

- 不复制数据定义语言（DDL）命令。
- 复制不支持在实时数据库中进行模式更改。在克隆过程中，将以其原始形式重新创建模式。如果您在克隆之后但在完成升级之前更改模式，则该模式不会反映在升级后的实例中。
- 无法复制大型对象，但您可以将数据存储在普通表中。
- 只有表（包括分区表）支持复制。不支持复制到其他类型的关系，例如视图、实例化视图或外部表。
- 未复制序列数据，需要在失效转移后手动更新。

> **Note**
>
> 此升级不支持自动脚本编写。您应该手动执行所有步骤。

设置和检查参数值

升级之前，将您的 Aurora PostgreSQL 数据库集群的写入器实例配置为发布服务器。实例应使用具有以下设置的自定义数据库集群参数组：

- `rds.logical_replication` – 将该参数设置为 1。`rds.logical_replication` 参数的用途与独立 PostgreSQL 服务器的 `wal_level` 参数以及其他控制预写日志文件管理的参数相同。
- `max_replication_slots` – 将此参数设置为您计划创建的订阅总数。如果您使用 AWS DMS，请将此参数设置为您计划用于从此数据库集群中捕获更改数据的 AWS DMS 任务数。
- `max_wal_senders` – 设置为并发连接数，加上几个额外连接，以供管理任务和新会话使用。如果您使用的是 AWS DMS，max_wal_senders 的数量应等于并发会话数加上在任何给定时间可能正在运行的 AWS DMS 任务数。
- `max_logical_replication_workers` – 设置为您预期的逻辑复制工作线程和表同步工作线程的数量。通常，将复制工作线程的数量设置为用于 max_wal_senders 的相同值是安全的。工作进程取自为服务器分配的后台进程池（max_worker_processes）。
- `max_worker_processes` – 设置为服务器的后台进程数。这个数字应该足够大，足以为复制、自动清理过程和其他可能同时进行的维护过程分配工作线程。

升级到 Aurora PostgreSQL 的更高版本时，您需要复制您在参数组的早期版本中修改的任何参数。这些参数应用于升级后的版本。您可以查询 `pg_settings` 表以获取参数设置列表，以便可以在新的 Aurora PostgreSQL 数据库集群上重新创建这些设置。

例如，要获取复制参数的设置，请运行以下查询：

```
SELECT name, setting FROM pg_settings WHERE name in
('rds.logical_replication', 'max_replication_slots',
'max_wal_senders', 'max_logical_replication_workers',
'max_worker_processes');
```

将 Aurora PostgreSQL 升级到新的主要版本

让发布者做好准备（蓝色）

1. 在下面的示例中，源写入器实例（蓝色）是运行 PostgreSQL 版本 11.15 的 Aurora PostgreSQL 数据库集群。这是我们复制场景中的发布节点。对于本演示，我们的源代码编写器实例托管了一个包含一系列值的示例表：

   ```
   CREATE TABLE my_table (a int PRIMARY KEY);
   INSERT INTO my_table VALUES (generate_series(1,100));
   ```

2. 要在源实例上创建发布，请使用 psql（PostgreSQL 的 CLI）或您选择的客户端连接到该实例的写入器节点。在每个数据库中输入以下命令：

   ```
   CREATE PUBLICATION publication_name FOR ALL TABLES;
   ```

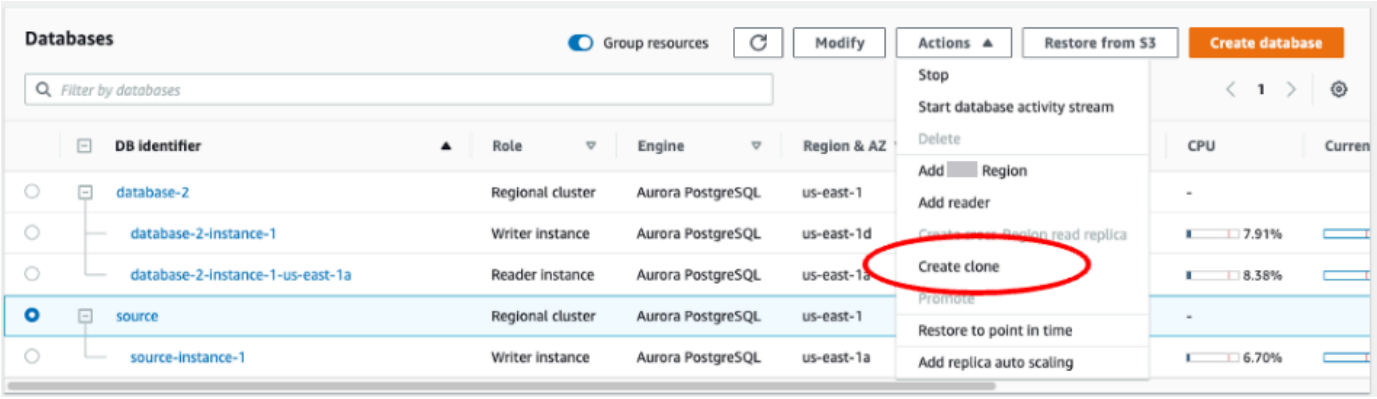
   publication_name 指定发布的名称。

3. 您还需要在实例上创建复制插槽。以下命令创建复制插槽并加载 pgoutput 逻辑解码插件。该插件将读取的内容从预写日志（WAL）更改为逻辑复制协议，并根据发布规范筛选数据。

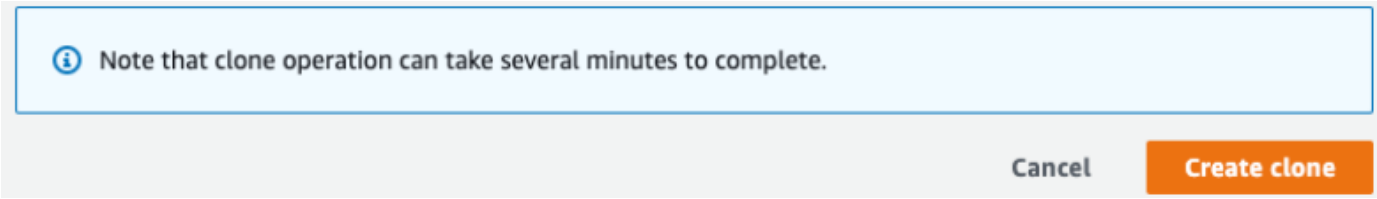
   ```
   SELECT pg_create_logical_replication_slot('replication_slot_name', 'pgoutput');
   ```

克隆发布者

1. 使用 Amazon RDS 控制台创建源实例的克隆。在 Amazon RDS 控制台中突出显示实例名称，然后在 Actions（操作）菜单中选择 Create clone（创建克隆）。



2. 为实例提供唯一名称。大多数设置是源实例中的缺省设置。当您对新实例进行了所需的更改后，选择 Create clone（创建克隆）。



3. 当目标实例启动时，写入器节点的 Status（状态）列在 Status（状态）列中显示 Creating（正在创建）。在实例准备就绪后，状态将变为 Available（可用）。

为升级准备克隆

1. 克隆是部署模型中的“绿色”实例。它是复制订阅节点的主机。当节点变为可用时，连接 psql 并查询新的写入器节点以获取日志序列号（LSN）。LSN 标识 WAL 流中记录的开头。

```
SELECT aurora_volume_logical_start_lsn();
```

2. 在查询的响应中，您可以找到 LSN 编号。您稍后在此过程中需要这个编号，所以请记下它。

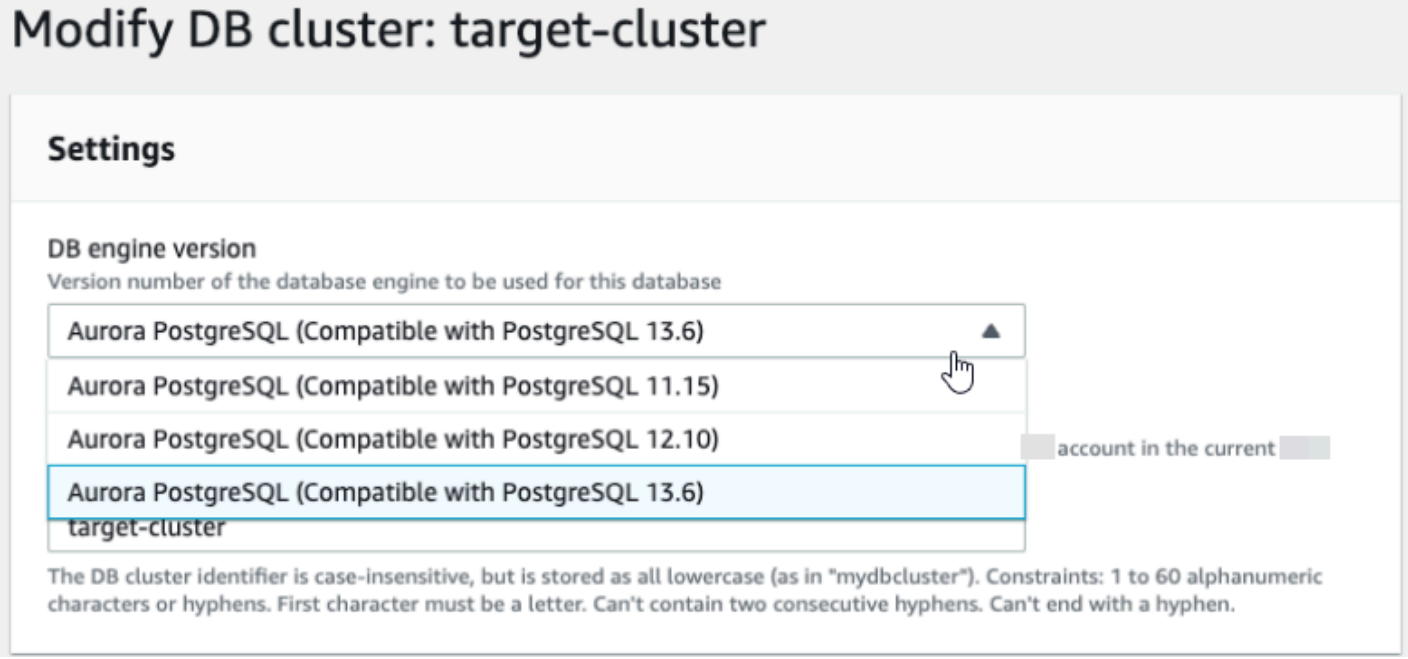
```
postgres=> SELECT aurora_volume_logical_start_lsn();
aurora_volume_logical_start_lsn
---------------
0/402E2F0
(1 row)
```

3. 在升级克隆之前，删除克隆的复制插槽。

```
SELECT pg_drop_replication_slot('replication_slot_name');
```

将集群升级到新的主要版本

- 在克隆提供商节点时，请使用 Amazon RDS 控制台在订阅节点上启动主要版本升级。在 RDS 控制台中突出显示实例名称，然后选择 Modify（修改）按钮。选择更新的版本和更新的参数组，然后立即应用设置来升级目标实例。



- 还可以使用 CLI 来执行升级：

```
aws rds modify-db-cluster –db-cluster-identifier $TARGET_Aurora_ID –engine-version
 13.6 –allow-major-version-upgrade –apply-immediately
```

让订阅者做好准备（绿色）

1. 当升级后克隆变为可用时，连接到 psql 并定义订阅。为此，您需要在 `CREATE SUBSCRIPTION` 命令中指定以下选项：

   - `subscription_name` – 订阅的名称。
   - `admin_user_name` – 拥有 rds_superuser 权限的管理用户的名称。
   - `admin_user_password` – 与管理用户关联的密码。
   - `source_instance_URL` – 发布服务器实例的 URL。
   - `database` – 订阅服务器将与之连接的数据库。
   - `publication_name` – 发布服务器的名称。

- replication_slot_name – 复制插槽的名称。

```
CREATE SUBSCRIPTION subscription_name
CONNECTION 'postgres://admin_user_name:admin_user_password@source_instance_URL/
database' PUBLICATION publication_name
WITH (copy_data = false, create_slot = false, enabled = false, connect = true,
 slot_name = 'replication_slot_name');
```

2. 创建订阅后，查询 pg_replication_origin 视图以检索 roname 值，该值是复制源的标识符。每个实例都有一个 roname：

```
SELECT * FROM pg_replication_origin;
```

例如：

```
postgres=>
SELECT * FROM pg_replication_origin;

roident | roname
---------+----------
1 | pg_24586
```

3. 在命令中提供您在发布节点的先前查询中保存的 LSN 以及从订阅节点 [实例]返回的 roname。此命令使用 pg_replication_origin_advance 函数指定要复制的日志序列的起点。

```
SELECT pg_replication_origin_advance('roname', 'log_sequence_number');
```

roname 是 pg_replication_origin 视图返回的标识符。

log_sequence_number 是 aurora_volume_logical_start_lsn 函数的先前查询返回的值。

4. 然后，使用 ALTER SUBSCRIPTION... ENABLE 子句开启逻辑复制。

```
ALTER SUBSCRIPTION subscription_name ENABLE;
```

5. 此时，您可以确认复制正在运行。向发布实例添加一个值，然后确认该值已复制到订阅节点。

然后，使用以下命令监控发布节点上的复制滞后：

```
SELECT now() AS CURRENT_TIME, slot_name, active, active_pid,
 pg_size_pretty(pg_wal_lsn_diff(pg_current_wal_lsn(),
confirmed_flush_lsn)) AS diff_size, pg_wal_lsn_diff(pg_current_wal_lsn(),
confirmed_flush_lsn) AS diff_bytes FROM pg_replication_slots WHERE slot_type =
 'logical';
```

例如：

```
postgres=> SELECT now() AS CURRENT_TIME, slot_name, active, active_pid,
 pg_size_pretty(pg_wal_lsn_diff(pg_current_wal_lsn(),
confirmed_flush_lsn)) AS diff_size, pg_wal_lsn_diff(pg_current_wal_lsn(),
 confirmed_flush_lsn) AS diff_bytes FROM pg_replication_slots WHERE slot_type =
 'logical';

current_time                   | slot_name             | active | active_pid |
 diff_size | diff_bytes
-------------------------------+-----------------------+--------+------------
+-----------+------------
2022-04-13 15:11:00.243401+00  | replication_slot_name | t      | 21854      | 136
 bytes | 136
(1 row)
```

您可以使用 `diff_size` 和 `diff_bytes` 值监控复制滞后。当这些值达到 0 时，即表示副本已赶上源数据库实例进度。

执行升级后任务

升级完成后，实例状态在控制台控制面板的 Status（状态）列中显示为 Available（可用）。在新实例上，建议您执行以下操作：

- 重定向应用程序以指向写入器节点。
- 添加读取器节点以管理案例量，并在写入器节点出现问题时提供高可用性。
- Aurora PostgreSQL 数据库集群偶尔需要操作系统更新。这些更新可能包含较新版本的 glibc 库。在此类更新期间，我们建议您遵循 Aurora PostgreSQL 中支持的排序规则中所述的指南。
- 更新用户对新实例的权限以确保访问权限。

在新实例上测试应用程序和数据后，我们建议您在删除初始实例之前对其进行最终备份。有关在 Amazon 主机上使用逻辑复制的更多信息，请参阅 为 Aurora PostgreSQL 数据库集群设置逻辑复制。

排查 Aurora PostgreSQL 中的存储问题

如果排序或索引创建操作所需的工作内存量超过 work_mem 参数分配的量，则 Aurora PostgreSQL 将多余的数据写入到临时磁盘文件。写入数据时，Aurora PostgreSQL 使用它用于存储错误和消息日志的相同存储空间，也即本地存储。您的 Aurora PostgreSQL 数据库集群中的每个实例都有一定量的可用本地存储。存储量基于其数据库实例类。要增加本地存储量，您需要修改实例以使用更大的数据库实例类。有关数据库实例类规格，请参阅 适用于 Aurora 的数据库实例类的硬件规格。

您可以通过观看 Amazon CloudWatch 的 FreeLocalStorage 指标来监控 Aurora PostgreSQL 数据库集群的本地存储空间。此指标报告 Aurora 数据库集群中的每个数据库实例可用于临时表和日志的存储量。有关更多信息，请参阅 使用 Amazon CloudWatch 监控 Amazon Aurora 指标。

排序、索引和分组操作在工作内存中开始，但通常必须分载到本地存储。如果您的 Aurora PostgreSQL 数据库集群由于这些类型的操作而耗尽本地存储，则可以通过采取以下操作之一来解决问题。

- 增加工作内存量。这减少了使用本地存储的需求。缺省情况下，PostgreSQL 为每个排序、分组和索引操作分配 4MB。要检查 Aurora PostgreSQL 数据库集群的写入器实例的当前工作内存值，请使用 psql 连接到该实例，并运行以下命令。

  ```
  postgres=> SHOW work_mem;
  work_mem
  ----------
   4MB
  (1 row)
  ```

  可以在排序、分组和其他操作之前增加会话级别的工作内存，如下所示。

  ```
  SET work_mem TO '1 GB';
  ```

  有关共享内存的更多信息，请参阅 PostgreSQL 文档中的资源消耗量。
- 更改日志保留期，以便将日志存储更短的时间范围。要了解如何操作，请参阅Aurora PostgreSQL 数据库日志文件。

对于大于 40TB 的 Aurora PostgreSQL 集数据库群，请勿使用 db.t2、db.t3 或 db.t4g 实例类。建议仅将 T 数据库实例类用于开发和测试服务器，或其他非生产服务器。有关更多信息，请参阅 数据库实例类类型。

使用 Amazon Aurora PostgreSQL 进行复制

接下来，可以找到有关使用 Amazon Aurora PostgreSQL 进行复制的信息，包括如何监控和使用逻辑复制。

主题

- 使用 Aurora 副本
- 提高 Aurora 副本的读取可用性
- 监控 Aurora PostgreSQL 复制
- 使用 Aurora 进行 PostgreSQL 逻辑复制的概述
- 为 Aurora PostgreSQL 数据库集群设置逻辑复制
- 关闭逻辑复制
- 监控 Aurora PostgreSQL 逻辑复制的直写缓存和逻辑插槽
- 示例：将逻辑复制与 Aurora PostgreSQL 数据库集群结合使用
- 示例：使用 Aurora PostgreSQL 和 AWS Database Migration Service 的逻辑复制

使用 Aurora 副本

Aurora 副本是 Aurora 数据库集群中的独立端点，最适合用于扩展读取操作以及提高可用性。Aurora 数据集群可以包含位于 Aurora 数据库集群AWS区域的整个可用区中的最多 15 Aurora 个副本。

数据库集群卷由该数据库集群的多个数据副本组成。不过，集群卷中的数据，对于主要写入器数据库实例和数据库集群中的 Aurora 副本表示为单个逻辑卷。有关 Aurora 副本的更多信息，请参阅 Aurora 副本。

Aurora 副本十分适用于读取扩展，因为它们完全专用于集群卷上的读取操作。写入器数据库实例管理写入操作。集群卷在 Aurora PostgreSQL 数据库集群中的所有实例之间共享。因此，无需额外地复制每个 Aurora 副本的数据副本。

对于 Aurora PostgreSQL，在删除 Aurora 副本时，系统将立即删除其实例端点，并将 Aurora 副本从读取器端点中删除。如果在正待删除的 Aurora 副本上运行语句，则有 3 分钟宽限期。现有语句可在此宽限期内正常完成。当此宽限期结束后，将关闭并删除 Aurora 副本。

Aurora PostgreSQL 数据库集群支持不同 AWS 区域中使用 Aurora Global Database 的 Aurora 副本。有关更多信息，请参阅 使用 Amazon Aurora Global Database。

> **Note**
>
> 有了读取可用性功能后，如果您想在数据库集群中重启 Aurora 副本，必须手动执行此操作。对于在此功能之前创建的数据库集群，重启写入器数据库实例会自动重启 Aurora 副本。自动重启将重新建立可以保证整个数据库集群读取/写入一致性的入口点。

提高 Aurora 副本的读取可用性

Aurora PostgreSQL 通过在写入器数据库实例重新启动或 Aurora 副本无法跟上写入流量时持续提供读取请求，来提高数据库集群的读取可用性。

默认情况下，读取可用性功能在以下版本的 Aurora PostgreSQL 上可用：

- 16.1 及所有更高版本
- 15.2 及更高的 15 版本
- 14.7 及更高的 14 版本
- 13.10 及更高的 13 版本
- 12.14 及更高的 12 版本

Aurora Global Database 的以下版本支持读取可用性功能：

- 16.1 及所有更高版本
- 15.4 及更高的 15 版本
- 14.9 及更高的 14 版本
- 13.12 及更高的 13 版本
- 12.16 及更高的 12 版本

对于在推出读取可用性功能之前在其中一个版本上创建的数据库集群，要使用该功能，请重新启动数据库集群的写入器实例。

修改 Aurora PostgreSQL 数据库集群的静态参数时，必须重新启动编写器实例，以便参数更改生效。例如，当您设置 `shared_buffers` 的值时，必须重新启动写入器实例。利用 Aurora 副本的读取可用性功能，数据库集群可保持改进的可用性，减少重启写入器实例所带来的影响。读取器实例不会重新启动和继续响应读取请求。要应用静态参数更改，请重启每个单独的读取器实例。

Aurora PostgreSQL 数据库集群的 Aurora 副本可以通过在与写入器重新连接后快速恢复到内存数据库状态，从写入器重新启动、失效转移、缓慢复制和网络问题等复制错误中恢复。这种方法允许 Aurora 副本实例在客户端数据库仍然可用时与最新的存储更新保持一致。

与复制恢复冲突的正在进行的事务可能会收到错误，但客户端可以在读取器赶上写入器后重试这些事务。

监控 Aurora 副本

从写入器断开连接中恢复时，您可以监控 Aurora 副本。使用以下指标检查有关读取器实例的最新信息，并跟踪进行中的只读事务。

- `aurora_replica_status` 函数会更新，以便在读取器实例仍处于连接状态时返回该实例的最新信息。对于与用于执行查询的数据库实例对应的行，`aurora_replica_status` 中的最后一次更新时间戳始终为空。这表明读取器实例具有最新的数据。
- 当 Aurora 副本与写入器实例断开连接并重新连接时，会发出以下数据库事件：

  ```
  Read replica has been disconnected from the writer instance and
  reconnected.
  ```
- 当由于恢复冲突而取消了只读查询时，您可能会在数据库错误日志中看到以下一条或多条错误消息：

  ```
  Canceling statement due to conflict with recovery.

  User query may not have access to page data to replica disconnect.

  User query might have tried to access a file that no longer exists.

  When the replica reconnects, you will be able to repeat your command.
  ```

限制

以下限制适用于具有读取可用性功能的 Aurora 副本：

- 如果在复制恢复期间无法从写入器实例流式传输数据，则可以重新启动辅助数据库集群的 Aurora 副本。
- 如果一个 Aurora 副本已在进行中并将重新启动，则 Aurora 副本不支持在线复制恢复。
- 当您的数据库实例接近事务 ID 重叠时，Aurora 副本将重新启动。有关事务 ID 重叠的更多信息，请参阅防止事务 ID 重叠故障。
- 在某些情况下，当复制过程受阻时，Aurora 副本可能重新启动。

监控 Aurora PostgreSQL 复制

读取扩展和高可用性依赖于尽可能短的滞后时间。您可以通过监控 Amazon CloudWatch `ReplicaLag` 指标来监控 Aurora 副本滞后于 Aurora PostgreSQL 数据库集群写入器数据库实例的时间。由于 Aurora 副本从写入器数据库实例所在的同一个集群卷读取数据，因此 `ReplicaLag` 指标对于 Aurora PostgreSQL 数据库集群有不同的含义。Aurora 副本的 `ReplicaLag` 指标表示 Aurora 副本的页面缓存相较写入器数据库实例页面缓存的滞后时间。

有关监控 RDS 实例和 CloudWatch 指标的更多信息，请参阅监控 Amazon Aurora 集群中的指标。

使用 Aurora 进行 PostgreSQL 逻辑复制的概述

通过将 PostgreSQL 的逻辑复制功能与 Aurora PostgreSQL 数据库集群结合使用，您可以复制和同步各个表而不是整个数据库实例。逻辑复制使用发布和订阅模型将更改从源复制到一个或多个接收者。它的工作原理是使用 PostgreSQL 预写日志（WAL）中的更改记录。源或发布者将指定表的 WAL 数据发送给一个或多个接收者（订阅者），从而复制更改并使订阅者的表与发布者的表保持同步。来自发布者的一组更改使用发布来识别。订阅者通过创建订阅（用于定义与发布者数据库及其发布内容的连接）来获取更改。复制插槽是该方案中用于跟踪订阅进度的机制。

对于 Aurora PostgreSQL 数据库集群，WAL 记录保存在 Aurora 存储中。在逻辑复制场景中充当发布者的 Aurora PostgreSQL 数据库集群从 Aurora 存储中读取 WAL 数据，对其进行解码，然后将其发送给订阅者，以便可以将更改应用于该实例上的表。发布者使用逻辑解码器对数据进行解码以供订阅者使用。原定设置情况下，Aurora PostgreSQL 数据库集群在发送数据时使用原生 PostgreSQL `pgoutput` 插件。提供了其他逻辑解码器。例如，Aurora PostgreSQL 还支持将 WAL 数据转换为 JSON 的 `wal2json` 插件。

从 Aurora PostgreSQL 版本 14.5、13.8、12.12 和 11.17 起，Aurora PostgreSQL 使用直写缓存增强了 PostgreSQL 逻辑复制过程以提高性能。WAL 事务日志本地缓存在缓冲区中，以减少磁盘输入/输出量，即在逻辑解码期间从 Aurora 存储中读取。每当您对 Aurora PostgreSQL 数据库集群使用逻辑复制时，原定设置情况下使用直写缓存。Aurora 提供了多种可用于管理缓存的函数。有关更多信息，请参阅 监控 Aurora PostgreSQL 逻辑复制直写缓存。

所有当前可用的 Aurora PostgreSQL 版本都支持逻辑复制。有关更多信息，请参阅《Aurora PostgreSQL 版本注释》中的 Amazon Aurora PostgreSQL 更新。

适用于 Aurora PostgreSQL 的 Babelfish 支持从以下版本进行逻辑复制：

- 15.7 及更高版本
- 16.3 及更高版本

> **Note**
>
> 除了 PostgreSQL 10 中引入的原生 PostgreSQL 逻辑复制功能外，Aurora PostgreSQL 还支持 `pglogical` 扩展。有关更多信息，请参阅 使用 pglogical 跨实例同步数据。

有关 PostgreSQL 逻辑复制的更多信息，请参阅 PostgreSQL 文档中的逻辑复制和逻辑解码概念。

为 Aurora PostgreSQL 数据库集群设置逻辑复制

设置逻辑复制需要 `rds_superuser` 权限。Aurora PostgreSQL 数据库集群必须配置为使用自定义数据库集群参数组，以便您可以设置必要的参数，详见以下过程。有关更多信息，请参阅 Amazon Aurora 数据库集群的数据库集群参数组。

为 Aurora PostgreSQL 数据库集群设置 PostgreSQL 逻辑复制

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择您的 Aurora PostgreSQL 数据库集群。
3. 打开 Configuration（配置）选项卡。在实例详细信息中，找到包含类型为数据库集群参数组的参数组链接。
4. 选择此链接以打开与 Aurora PostgreSQL 数据库集群关联的自定义参数。
5. 在 Parameters（参数）搜索字段中，键入 `rds` 以查找 `rds.logical_replication` 参数。此参数的原定设置值为 `0`，这意味着它在原定设置情况下处于关闭状态。
6. 选择 Edit parameters（编辑参数）以访问属性值，然后从选择器中选择 `1` 以开启该功能。根据您的预期使用情况，您可能还需要更改以下参数的设置。但是，在许多情况下，原定设置值就足够了。

   - `max_replication_slots` – 将此参数设置为至少等于您计划的逻辑复制发布和订阅总数的值。如果您正在使用 AWS DMS，则此参数应至少等于集群中计划的更改数据捕获任务数加上逻辑复制发布和订阅数。
   - `max_wal_senders` 和 `max_logical_replication_workers` – 将这些参数设置为至少与您打算激活的逻辑复制插槽数量或更改数据捕获的活动 AWS DMS 任务数量相等的值。使逻辑复制插槽处于非活动状态可防止 vacuum 从表中删除过时的元组，因此，我们建议您监视复制插槽，并根据需要删除非活动的插槽。
   - `max_worker_processes` – 将此参数设置为至少等于 `max_logical_replication_workers`、`autovacuum_max_workers` 和

max_parallel_workers 值的总和的值。在小型数据库实例类上，后台工件进程可能会影响应用程序工作负载，因此，如果将 max_worker_processes 设置为高于原定设置值，请监控数据库的性能。（原定设置值是 GREATEST(${DBInstanceVCPU*2},8} 的结果，这意味着，在原定设置情况下，这是 8 或数据库实例类的 CPU 等效量的 2 倍，以较大者为准）。

> **Note**
>
> 您可以修改客户创建的数据库参数组中的参数值，但不能更改原定设置数据库参数组中的参数值。

7. 选择 Save changes（保存更改）。
8. 重启 Aurora PostgreSQL 数据库集群的写入器实例，以使更改生效。在 Amazon RDS 控制台中，选择集群的主数据库实例，然后从 Actions（操作）菜单中选择 Reboot（重启）。
9. 当实例可用时，您可以验证逻辑复制是否已开启，如下所示。

   a. 使用 psql 连接到 Aurora PostgreSQL 数据库集群的写入器实例。

   ```
   psql --host=your-db-cluster-instance-1.aws-region.rds.amazonaws.com --port=5432
    --username=postgres --password --dbname=labdb
   ```

   b. 使用以下命令验证逻辑复制是否已启用。

   ```
   labdb=> SHOW rds.logical_replication;
    rds.logical_replication
   -------------------------
    on
   (1 row)
   ```

   c. 验证 wal_level 设置为 logical。

   ```
   labdb=> SHOW wal_level;
     wal_level
   -----------
    logical
   (1 row)
   ```

有关使用逻辑复制使数据库表与来自源 Aurora PostgreSQL 数据库集群的更改保持同步的示例，请参阅 示例：将逻辑复制与 Aurora PostgreSQL 数据库集群结合使用。

关闭逻辑复制

完成复制任务后，应停止复制过程，删除复制插槽并关闭逻辑复制。在删除插槽之前，请确保不再需要它们。无法删除活动的复制插槽。

关闭逻辑复制

1. 删除所有复制插槽。

   要删除所有复制插槽，请连接到发布者并运行以下 SQL 命令。

   ```
   SELECT pg_drop_replication_slot(slot_name)
     FROM pg_replication_slots
    WHERE slot_name IN (SELECT slot_name FROM pg_replication_slots);
   ```

   运行此命令时，复制插槽不能处于活动状态。
2. 修改与发布者关联的自定义数据库集群参数组（如为 Aurora PostgreSQL 数据库集群设置逻辑复制详述），但将 `rds.logical_replication` 参数设置为 0。

   有关自定义数据库参数组的更多信息，请参阅在 Amazon Aurora 中修改数据库集群参数组中的参数。
3. 重启发布者 Aurora PostgreSQL 数据库集群，以使对 `rds.logical_replication` 参数的更改生效。

监控 Aurora PostgreSQL 逻辑复制的直写缓存和逻辑插槽

监控逻辑复制直写缓存并管理逻辑插槽，以提高 Aurora PostgreSQL 数据库集群的性能。接下来，查找有关直写缓存和逻辑插槽的更多信息。

主题

- 监控 Aurora PostgreSQL 逻辑复制直写缓存
- 管理 Aurora PostgreSQL 的逻辑插槽

监控 Aurora PostgreSQL 逻辑复制直写缓存

原定设置情况下，Aurora PostgreSQL 版本 14.5、13.8、12.12 和 11.17 及更高版本使用直写缓存来提高逻辑复制的性能。如果没有直写缓存，Aurora PostgreSQL 在实现原生 PostgreSQL 逻辑复制过程时使用 Aurora 存储层。为此，它将 WAL 数据写入存储，然后从存储中读回数据以对其进行解码并发送

（复制）到其目标（订阅者）。这可能会导致在 Aurora PostgreSQL 数据库集群的逻辑复制过程中出现瓶颈。

直写缓存尽可能减少了对 Aurora 存储层的依赖。Aurora PostgreSQL 并不始终在该层进行写入和读取，而是使用缓冲区来缓存逻辑 WAL 流以便在复制过程中使用，从而减少了访问磁盘的需求。该缓冲区是逻辑复制中使用的原生 PostgreSQL 缓存，在 Aurora PostgreSQL 数据库集群参数中标识为 `rds.logical_wal_cache`。

当您在 Aurora PostgreSQL 数据库集群（对于支持直写缓存的版本）中使用逻辑复制时，您可以监控缓存命中率以查看其对您的使用案例的效果有多好。为此，请使用 `psql` 连接到 Aurora PostgreSQL 数据库集群的写入实例，然后使用 Aurora 函数 `aurora_stat_logical_wal_cache`，如以下示例所示。

```
SELECT * FROM aurora_stat_logical_wal_cache();
```

该函数返回如下输出。

```
name       | active_pid | cache_hit | cache_miss | blks_read | hit_rate |
 last_reset_timestamp
-----------+------------+-----------+------------+-----------+----------+--------------
test_slot1 | 79183      | 24        | 0          | 24        | 100.00%  | 2022-08-05
 17:39...
test_slot2 |            | 1         | 0          |  1        | 100.00%  | 2022-08-05
 17:34...
(2 rows)
```

为了便于阅读，已缩短了 `last_reset_timestamp` 值。有关此函数的更多信息，请参阅 aurora_stat_logical_wal_cache。

Aurora PostgreSQL 提供了以下两个用于监控直写缓存的函数。

- `aurora_stat_logical_wal_cache` 函数 – 有关参考文档，请参阅 aurora_stat_logical_wal_cache。
- `aurora_stat_reset_wal_cache` 函数 – 有关参考文档，请参阅 aurora_stat_reset_wal_cache。

如果您发现自动调整的 WAL 缓存大小不足以满足您的工作负载，则可以手动更改 `rds.logical_wal_cache` 的值。请考虑以下事项：

- 当 `rds.logical_replication` 参数禁用时，`rds.logical_wal_cache` 设置为零（0）。

- 当 rds.logical_replication 参数启用时，rds.logical_wal_cache 默认值为 16 MB。
- rds.logical_wal_cache 参数是静态的，并需要重新启动数据库实例以使更改生效。根据 8 Kb 数据块定义此参数。请注意，任何小于 32 Kb 的正值均视为 32 Kb。有关 wal_buffers 的更多信息，请参阅 PostgreSQL 文档中的预写日志。

管理 Aurora PostgreSQL 的逻辑插槽

流式传输活动在 pg_replication_origin_status 视图中捕获。要查看此视图的内容，您可以使用 pg_show_replication_origin_status() 函数，如下所示：

```
SELECT * FROM pg_show_replication_origin_status();
```

您可以使用以下 SQL 查询获取逻辑插槽的列表。

```
SELECT * FROM pg_replication_slots;
```

要删除逻辑插槽，请使用 pg_drop_replication_slot 以及插槽的名称，如以下命令所示。

```
SELECT pg_drop_replication_slot('test_slot');
```

示例：将逻辑复制与 Aurora PostgreSQL 数据库集群结合使用

以下过程说明了如何在两个 Aurora PostgreSQL 数据库集群之间启动逻辑复制。发布者和订阅者都必须配置为进行逻辑复制，如为 Aurora PostgreSQL 数据库集群设置逻辑复制中详述。

作为指定发布者的 Aurora PostgreSQL 数据库集群也必须允许访问复制插槽。为此，请基于 Amazon VPC 服务修改与 Aurora PostgreSQL 数据库集群的虚拟公有云（VPC）关联的安全组。通过将与订阅者的 VPC 关联的安全组添加到发布者的安全组来允许入站访问。有关安全组的更多信息，请参阅《Amazon VPC 用户指南》中的使用安全组控制到资源的流量。

完成这些初步步骤后，您可以对发布者使用 PostgreSQL 命令 CREATE PUBLICATION，并对订阅者使用 PostgreSQL 命令 CREATE SUBSCRIPTION，详见以下过程。

在两个 Aurora PostgreSQL 数据库集群之间启动逻辑复制过程

这些步骤假设您的 Aurora PostgreSQL 数据库集群有一个写入器实例，其中包含一个用于创建示例表的数据库。

1. 在发布者 Aurora PostgreSQL 数据库集群上

   a. 使用以下 SQL 语句创建一个表。

   ```
   CREATE TABLE LogicalReplicationTest (a int PRIMARY KEY);
   ```

   b. 使用以下 SQL 语句将数据插入到发布者数据库。

   ```
   INSERT INTO LogicalReplicationTest VALUES (generate_series(1,10000));
   ```

   c. 使用以下 SQL 语句验证表中是否存在数据。

   ```
   SELECT count(*) FROM LogicalReplicationTest;
   ```

   d. 使用 CREATE PUBLICATION 语句为此表创建发布，如下所示。

   ```
   CREATE PUBLICATION testpub FOR TABLE LogicalReplicationTest;
   ```

2. 在订阅者 Aurora PostgreSQL 数据库集群上

   a. 在订阅者上创建与在发布者上创建的同一个 LogicalReplicationTest 表，如下所示。

   ```
   CREATE TABLE LogicalReplicationTest (a int PRIMARY KEY);
   ```

   b. 验证此表为空。

   ```
   SELECT count(*) FROM LogicalReplicationTest;
   ```

   c. 创建订阅以获取来自发布者的更改。您需要使用以下有关发布者 Aurora PostgreSQL 数据库集群的详细信息。

   - host – 发布者 Aurora PostgreSQL 数据库集群的写入器数据库实例。
   - port – 写入器数据库实例正在侦听的端口。PostgreSQL 的默认值为 5432。
   - dbname – 数据库的名称。

   ```
   CREATE SUBSCRIPTION testsub CONNECTION
      'host=publisher-cluster-writer-endpoint port=5432 dbname=db-name user=user
    password=password'
      PUBLICATION testpub;
   ```

> **Note**
>
> 作为安全最佳实践，请指定除此处所示提示以外的密码。

创建订阅后，将在发布者上创建逻辑复制槽。

d. 要验证此示例中的初始数据是否在订阅者上复制，请在订阅者数据库中使用以下 SQL 语句。

```
SELECT count(*) FROM LogicalReplicationTest;
```

在发布者上进行的任何其他更改都会被复制到订阅者。

逻辑复制会影响性能。我们建议您在复制任务完成后关闭逻辑复制。

示例：使用 Aurora PostgreSQL 和 AWS Database Migration Service 的逻辑复制

您可以使用 AWS Database Migration Service (AWS DMS) 复制数据库或数据库的一部分。使用 AWS DMS 将数据从 Aurora PostgreSQL 数据库迁移到另一个开源或商用数据库。有关 AWS DMS 的更多信息，请参阅 AWS Database Migration Service 用户指南。

以下示例演示如何从作为发布者的 Aurora PostgreSQL 数据库设置逻辑复制，然后使用 AWS DMS 进行迁移。此示例使用在 示例：将逻辑复制与 Aurora PostgreSQL 数据库集群结合使用 中创建的相同发布者和订阅者。

要设置 AWS DMS 的逻辑复制，需要有您在 Amazon RDS 中的发布者和订阅者的详细信息。特别是，您需要有发布者的写入器数据库实例和订阅者的数据库实例的详细信息。

请获取发布者的写入器数据库实例的以下信息：

- Virtual Private Cloud (VPC) 标识符
- 子网组
- 可用区 (AZ)
- VPC 安全组
- 数据库实例 ID

请获取发布者的数据库实例的以下信息：

- 数据库实例 ID
- 源引擎

使用 AWS DMS 进行 Aurora PostgreSQL 的逻辑复制

1. 准备发布者数据库来与 AWS DMS 一起工作。

   要实现此目的，PostgreSQL 10.x 及更高版本的数据库需要您对发布者数据库应用 AWS DMS 封装函数。有关此步骤以及后续步骤的详细信息，请参阅《AWS Database Migration Service 用户指南》的将 PostgreSQL 版本 10.x 及更高版本用作 AWS DMS 的源中的说明。
2. 通过以下网址登录 AWS Management Console并打开 AWS DMS 控制台：https://console.aws.amazon.com/dms/v2。在右上方，选择发布者和订阅者所位于的相同 AWS 区域。
3. 创建 AWS DMS 复制实例。

   选择与发布者的写入器数据库实例的值相同的值。包括以下设置：

   - 对于 VPC，选择与写入器数据库实例相同的 VPC。
   - 对于 Replication Subnet Group（复制子网组），选择与写入器数据库实例具有相同值的子网组。如有必要，请创建一个新的子网组。
   - 对于 Availability zone (可用区)，选择与写入器数据库实例相同的区域。
   - 对于 VPC Security Group (VPC 安全组)，选择与写入器数据库实例相同的组。
4. 为源创建 AWS DMS 终端节点。

   使用以下设置将发布者指定为源终端节点：

   - 对于 Endpoint type (终端节点类型)，选择 Source endpoint (源终端节点)。
   - 选择 Select RDS DB Instance (选择 RDS 数据库实例)。
   - 对于 RDS Instance (RDS 实例)，选择发布者的写入器数据库实例的数据库标识符。
   - 对于 Source engine (源引擎)，选择 postgres。
5. 为目标创建 AWS DMS 终端节点。

   使用以下设置将订阅者指定为目标终端节点：

   - 对于 Endpoint type (终端节点类型)，选择 Target endpoint (目标终端节点)。
   - 选择 Select RDS DB Instance (选择 RDS 数据库实例)。

- 对于 RDS Instance (RDS 实例)，选择订阅者数据库实例的数据库标识符。
- 为 Source engine (源引擎) 选择值。例如，如果订阅者是 RDS PostgreSQL 数据库，则选择 postgres。如果订阅者是 Aurora PostgreSQL 数据库，请选择 aurora-postgresql。

6. 创建 AWS DMS 数据库迁移任务。

   您使用数据库迁移任务指定要迁移的数据库表，使用目标架构映射数据并在目标数据库上创建新表。至少为 Task configuration (任务配置) 使用以下设置：

   - 对于 Replication instance (复制实例)，选择您在之前的步骤中创建的复制实例。
   - 对于 Source database endpoint (源数据库终端节点)，选择您在之前的步骤中创建的发布者源。
   - 对于 Target database endpoint (目标数据库终端节点)，选择您在之前的步骤中创建的订阅者目标。

   其余任务详细信息取决于您的迁移项目。有关为 DMS 任务指定所有详细信息的更多信息，请参阅 AWS Database Migration Service 用户指南中的使用 AWS DMS 任务。

在 AWS DMS 创建任务后，它开始将数据从发布者迁移到订阅者。

Aurora PostgreSQL 中的本地写入转发

本地（集群内）写入转发支持您的应用程序直接在 Aurora 副本上发出读/写事务。然后，写入命令转发到写入器数据库实例进行提交。对于偶尔写入且要求先写后读一致性（即能够读取事务中的最新写入内容）的应用程序，您可以使用本地写入转发。

如果没有写入转发，您的应用程序必须完全拆分所有读取和写入流量，保持两组数据库连接才能将流量发送到相应的端点。只读副本从写入器实例异步接收更新。此外，由于不同只读副本的复制滞后可能不同，因此很难在所有副本之间实现全局读取一致性。必须在写入器数据库实例上处理任何要求先写后读一致性的读取。或者，您需要开发复杂的自定义应用程序逻辑，以利用多个只读副本在确保一致性的同时实现可扩展性。

借助写入转发，您无需拆分这些事务或将它们专门发送到写入器实例。您也不必开发复杂的应用程序逻辑来实现先写后读一致性。

本地写入转发在提供 Aurora PostgreSQL 的每个区域均可用。以下 Aurora PostgreSQL 版本支持该特征：

- 16.4 及更高的 16 版本

- 15.8 及更高的 15 版本
- 14.13 及更高的 14 版本

本地写入转发用于转发来自区域内副本的写入。要转发来自全局副本的写入，请参阅 在 Amazon Aurora Global Database 中使用写入转发。

主题

- Aurora PostgreSQL 中的本地写入转发的限制和注意事项
- 配置 Aurora PostgreSQL 以进行本地写入转发
- 在 Aurora PostgreSQL 中使用本地写入转发
- 监控 Aurora PostgreSQL 中的本地写入转发

Aurora PostgreSQL 中的本地写入转发的限制和注意事项

目前，Aurora PostgreSQL 中的本地写入转发有以下限制：

- RDS 代理不支持本地写入转发。
- 在具有写入转发功能的 Aurora PostgreSQL 中不允许使用某些语句，否则可能产生过时的结果。此外，不支持用户定义的函数和用户定义的过程。因此，默认情况下，数据库集群的 `EnableLocalWriteForwarding` 设置处于关闭状态。在启用它之前，请检查以确保您的应用程序代码不受任何这些限制的影响。
- 写入转发不支持以下类型的 SQL 语句：

> **Note**
>
> 这些语句可以由您在应用程序中隐式使用，也可以由 PostgreSQL 协议推断出来。例如，PL/SQL 异常处理可能会导致使用 SAVEPOINT，但该语句不受支持。

  - `ANALYZE`
  - `CLUSTER`
  - `COPY`
  - 游标 - 不支持游标，因此在使用本地写入转发之前，请务必将其关闭。
  - 数据定义语言（DDL）语句

- GRANT|REVOKE|REASSIGN OWNED|SECURITY LABEL
- LISTEN / NOTIFY
- LOCK
- SAVEPOINT
- SELECT INTO
- SET CONSTRAINTS
- 序列更新：nextval()、setval()
- TRUNCATE
- 两阶段提交命令：PREPARE TRANSACTION、COMMIT PREPARED、ROLLBACK PREPARED
- 用户定义的函数和用户定义的过程。
- VACUUM

您可以考虑将以下 SQL 语句与写入转发一起使用：

- DML 语句可能由多个部分组成，如 INSERT ... SELECT 语句或 DELETE ... WHERE 语句。在这种情况下，整个语句将转发到写入器数据库实例并在此处运行。
- 数据操作语言 (DML) 语句，如 INSERT、DELETE 和 UPDATE。
- EXPLAIN 语句和此列表中的语句。
- PREPARE 和 EXECUTE 语句。
- SELECT FOR { UPDATE | NO KEY UPDATE | SHARE | KEY SHARE } 语句。

配置 Aurora PostgreSQL 以进行本地写入转发

使用以下部分，您可以为 Amazon Aurora PostgreSQL 数据库集群启用本地写入转发，配置一致性级别以及使用写入转发来管理事务。

启用本地写入转发

默认情况下，Aurora PostgreSQL 数据库集群不启用本地写入转发。您在集群级别而不是在实例级别启用本地写入转发。

控制台

使用 AWS Management Console，在创建或修改数据库集群时，选中只读副本写入转发下的开启本地写入转发复选框。

AWS CLI

要使用 AWS CLI 启用本地写入转发，请使用 `--enable-local-write-forwarding` 选项。当您使用 `create-db-cluster` 命令创建新的数据库集群时，此选项将起作用。当您使用 `modify-db-cluster` 命令修改现有数据库集群时，它也起作用。您可以结合这些相同的 CLI 命令使用 `--no-enable-local-write-forwarding` 选项来禁用本地写入转发。

以下示例创建了一个启用本地写入转发的 Aurora PostgreSQL 数据库集群。

```
aws rds create-db-cluster \
--db-cluster-identifier write-forwarding-test-cluster \
--enable-local-write-forwarding \
--engine aurora-postgresql \
--engine-version 16.4 \
--master-username myuser \
--master-user-password mypassword \
--backup-retention 1
```

然后，您可以创建写入器和读取器数据库实例，以便您可以使用写入转发。有关更多信息，请参阅 创建 Amazon Aurora 数据库集群。

RDS API

要使用 Amazon RDS API 启用本地写入转发，请将 `EnableLocalWriteForwarding` 参数设置为 `true`。当您使用 `CreateDBCluster` 操作创建新的数据库集群时，此参数起作用。当您使用 `ModifyDBCluster` 操作修改现有数据库集群时，它也起作用。您可以通过将 `EnableLocalWriteForwarding` 参数设置为 `false` 来禁用本地写入转发。

为数据库会话启用本地写入转发

`apg_write_forward.consistency_mode` 参数是启用写入转发的数据库参数和数据库集群参数。您可以为读取一致性级别指定 `SESSION`、`EVENTUAL`、`GLOBAL` 或 `OFF`。要了解有关一致性级别的更多信息，请参阅Aurora PostgreSQL 中本地写入转发的一致性和隔离。

以下规则适用于此参数：

- 默认值为 `SESSION`。
- 仅当您将 `apg_write_forward.consistency_mode` 设置为 `EVENTUAL`、`SESSION` 或 `GLOBAL` 时，本地写入转发才可用。此参数仅适用于启用了本地写入转发的数据库集群的读取器实例。

- 将该值设置为 OFF 将在会话中禁用本地写入转发。

Aurora PostgreSQL 中本地写入转发的一致性和隔离

您可以控制只读副本上的读取一致性程度。您可以调整读取一致性级别，以确保会话中的所有转发的写入操作在只读副本中均可见，然后再进行任何后续查询。您还可以使用此设置来确保只读副本上的查询始终看到来自写入器数据库实例的最新更新。即使是由其他会话或其他集群提交的更新。要为应用程序指定此类行为，请为会话级别参数 apg_write_forward.consistency_mode 选择一个合适的值。apg_write_forward.consistency_mode 参数仅对启用了本地写入转发的只读副本产生影响。

> **Note**
>
> 对于 apg_write_forward.consistency_mode 参数，您可以指定值 SESSION、EVENTUAL、GLOBAL 和 OFF。默认情况下， 值设为 SESSION。将该值设置为 OFF 将禁用写入转发。

随着您提高一致性级别，您的应用程序会花更多时间等待更改传播到只读副本。您可以选择在较低延迟与确保运行查询之前在其他位置进行的更改完全可用之间取得平衡。

对于每种可用的一致性模式设置，效果如下所示：

- SESSION – 只读副本上使用本地写入转发的会话会看到在该会话中所做的所有更改的结果。无论事务是否已提交，这些更改都是可见的。如有必要，查询将等待转发的写入操作的结果复制到当前读取器数据库实例。它不会等待在当前数据库集群内的其他会话中执行的写入操作的更新结果。
- EVENTUAL – 只读副本上使用本地写入转发的查询可能会看到由于复制滞后而稍微过时的数据。在对写入器数据库实例执行写入操作并将其复制到只读副本之前，看不到同一会话中写入操作的结果。查询不会等待更新的结果可用。因此，它可能会检索较旧的数据或更新的数据，具体取决于语句的时间和复制滞后量。
- GLOBAL – 只读副本上的会话会看到该会话所做的更改。它还可以查看来自写入器数据库实例和其他只读副本的所有已提交的更改。每个查询可能会等待一段时间，该时间取决于会话滞后量。从查询开始时，如果只读副本处于最新状态（具有来自写入器数据库实例的所有已提交数据），查询将继续进行。

> **Note**
>
> 全局一致性模式会影响会话中执行的查询的延迟。即使会话未发送任何写入查询，它也会执行等待。

- `OFF` – 本地写入转发已禁用。

在使用写入转发的会话中，您可以使用 `REPEATABLE READ` 和 `READ COMMITTED` 隔离级别。但是，不支持 `SERIALIZABLE` 隔离级别。

有关写入转发涉及的所有参数的更多信息，请参阅 写入转发的默认参数设置。

具有写入转发的事务访问模式

如果事务访问模式设置为只读，则不使用本地写入转发。在连接到启用了本地写入转发的数据库集群和会话时，可以将访问模式设置为只读写。

有关事务访问模式的更多信息，请参阅 SET TRANSACTION。

在 Aurora PostgreSQL 中使用本地写入转发

使用以下部分，您可以检查数据库集群是否启用了本地写入转发，查看兼容性注意事项，并查看可配置参数和身份验证设置。这些信息详细介绍了如何有效利用 Aurora PostgreSQL 中的本地写入转发功能。

> **Note**
>
> 当使用本地写入转发的集群中的写入器实例重启时，使用本地写入转发的读取器实例上任何活跃的转发事务和查询都将自动关闭。在写入器实例再次可用后，您可以重试这些事务。

检查数据库集群是否启用了本地写入转发

要确定是否可以在数据库集群中使用本地写入转发，请确认该集群的属性 `LocalWriteForwardingStatus` 是否设置为 `enabled`。

在 AWS Management Console中，在集群详细信息页面的配置选项卡上，您可以看到本地只读副本写入转发的状态为已启用。

要查看所有集群的本地写入转发设置的状态，请运行以下 AWS CLI 命令。

Example

```
aws rds describe-db-clusters \
--query '*[].
{DBClusterIdentifier:DBClusterIdentifier,LocalWriteForwardingStatus:LocalWriteForwardingStatus}

[
{
"LocalWriteForwardingStatus": "enabled",
"DBClusterIdentifier": "write-forwarding-test-cluster-1"
},
{
"LocalWriteForwardingStatus": "disabled",
"DBClusterIdentifier": "write-forwarding-test-cluster-2"
},
{
"LocalWriteForwardingStatus": "requested",
"DBClusterIdentifier": "test-global-cluster-2"
},
{
"LocalWriteForwardingStatus": "null",
"DBClusterIdentifier": "aurora-postgresql-v2-cluster"
}
]
```

对于 LocalWriteForwardingStatus，数据库集群可以具有以下值：

- disabled – 本地写入转发已禁用。
- disabling – 正在禁用本地写入转发。
- enabled – 本地写入转发已启用。
- enabling – 正在启用本地写入转发。
- null – 本地写入转发不适用于此数据库集群。
- requested – 已请求本地写入转发，但尚未激活。

写入转发的默认参数设置

Aurora 集群参数组包含本地写入转发功能的设置。由于这些是集群参数，因此每个集群中的所有数据库实例对于这些变量具有相同的值。下表汇总了有关这些参数的详细信息，表后附有使用说明。

参数	范围	类型	默认值	有效值
apg_write_forward.connect_timeout	会话	秒	30	0–2147483647
apg_write_forward.consistency_mode	Session	enum	会话	SESSION、EVENTU和 OFF
apg_write_forward.idle_in_transaction_session_timeout	会话	毫秒	86400000	0–2147483647
apg_write_forward.idle_session_timeout	会话	毫秒	300000	0–2147483647
apg_write_forward.max_forwarding_connections_percent	全局	整数	25	1–100

apg_write_forward.max_forwarding_connections_percent 参数是可以用于处理从读取器转发的查询的数据库连接插槽上限。它表示为写入器数据库实例的 max_connections 设置的百分比。例如，如果 max_connections 是 800，而 apg_write_forward.max_forwarding_connections_percent 是 10，则写入器最多允许 80 个同时转发会话。这些连接来自由 max_connections 设置管理的同一连接池。此设置仅在集群启用了本地写入转发时适用于写入器数据库实例。

使用以下设置来控制本地写入转发请求：

- apg_write_forward.consistency_mode – 会话级参数，用于控制只读副本上的读取一致性程度。有效值为 SESSION、EVENTUAL、GLOBAL 或 OFF。默认情况下，值设为 SESSION。将该值设置为 OFF 将在会话中禁用本地写入转发。要了解有关一致性级别的更多信息，请参阅Aurora PostgreSQL 中本地写入转发的一致性和隔离。此参数仅适用于启用了本地写入转发的读取器实例。
- apg_write_forward.connect_timeout – 只读副本在与写入器数据库实例建立连接时在放弃之前等待的最大秒数。值 0 表示无限期等待。
- apg_write_forward.idle_in_transaction_session_timeout – 写入器数据库实例在关闭从具有未处理事务的只读副本转发的连接之前等待活动的毫秒数。如果在此段时间之后，会话在事务中仍处于空闲状态，则 Aurora 会终止会话。0 值禁用超时。

- `apg_write_forward.idle_session_timeout` – 写入器数据库实例在关闭从只读副本转发的连接之前等待活动的毫秒数。如果会话在这段时间之后仍处于空闲状态，则 Aurora 会终止会话。值 `0` 禁用超时。

rdswriteforwarduser

`rdswriteforwarduser` 是我们将用于在只读副本和写入器数据库实例之间建立连接的用户。

> **Note**
>
> `rdswriteforwarduser` 通过 PUBLIC 角色继承其对客户数据库的 CONNECT 权限。如果撤销 PUBLIC 角色的权限，则需要为将写入转发到的数据库授予 CONNECT 权限。

监控 Aurora PostgreSQL 中的本地写入转发

使用以下部分，您可以监控 Aurora PostgreSQL 集群中的本地写入转发（包括相关的 CloudWatch 指标和等待事件），以跟踪性能并识别潜在问题。

用于写入转发的 Amazon CloudWatch 指标和 Aurora PostgreSQL 状态变量

在一个或多个只读副本上使用写入转发时，以下 Amazon CloudWatch 指标适用于写入器数据库实例。

CloudWatch 指标	单位和描述
`AuroraLocal ForwardingWriterDML Throughput`	计数（每秒）。此写入器数据库实例每秒处理的转发 DML 语句数。
`AuroraLocalForwardingWriter OpenSessions`	计数。此写入器数据库实例上处理转发查询的打开会话数。
`AuroraLocal ForwardingWriterTo talSessions`	计数。此写入器数据库实例上的转发会话总数。

以下 CloudWatch 指标适用于每个只读副本。这些指标在启用了本地写入转发的数据库集群中的每个读取器数据库实例上进行测量。

CloudWatch 指标	单位和描述
AuroraForwardingReplicaCommitThroughput	计数（每秒）。此副本每秒转发的会话中的提交数。
AuroraForwardingReplicaDMLLatency	毫秒。副本上转发 DML 的平均响应时间（以毫秒为单位）。
AuroraForwardingReplicaDMLThroughput	计数（每秒）。此副本每秒处理的转发 DML 语句数。
AuroraForwardingReplicaErrorSessionsLimit	计数。由于达到最大连接数或最大写入转发连接数限制而被写入器数据库实例拒绝的会话数。
AuroraForwardingReplicaOpenSessions	计数。在副本实例上使用本地写入转发的会话数。
AuroraForwardingReplicaReadWaitLatency	毫秒。副本等待与写入器数据库实例 LSN 一致的平均等待时间（以毫秒为单位）。读取器数据库实例等待的程度取决于 apg_write_forward.consistency_mode 设置。有关该设置的信息，请参阅 the section called “Aurora PostgreSQL 中写入转发的配置参数”。

在 Aurora PostgreSQL 中使用本地写入转发的等待事件

当您在 Aurora PostgreSQL 中使用写入转发时，Amazon Aurora 会生成以下等待事件。

主题

- IPC:AuroraWriteForwardConnect
- IPC:AuroraWriteForwardConsistencyPoint
- IPC:AuroraWriteForwardExecute
- IPC:AuroraWriteForwardGetGlobalConsistencyPoint
- IPC:AuroraWriteForwardXactAbort
- IPC:AuroraWriteForwardXactCommit
- IPC:AuroraWriteForwardXactStart

IPC:AuroraWriteForwardConnect

当只读副本上的后端进程在等待开启与写入器数据库实例的连接时，就会发生该 `IPC:AuroraWriteForwardConnect` 事件。

等待次数增加的可能原因

随着尝试从只读副本连接到写入器节点的次数不断增加，此事件也会增加。

操作

减少从只读副本到写入器节点的同时连接数量。

IPC:AuroraWriteForwardConsistencyPoint

该 `IPC:AuroraWriteForwardConsistencyPoint` 事件描述在将转发写入操作的结果复制到当前区域之前，来自只读副本上节点的查询将等待的时间。仅当会话级别的参数 `apg_write_forward.consistency_mode` 设置为以下项之一时，才会生成此事件：

- `SESSION` – 只读副本上的查询等待在该会话中所做的所有更改的结果。
- `GLOBAL` – 只读副本上的查询等待该会话所做更改的结果，以及写入器数据库实例和只读副本中已提交的所有更改的结果。

有关 `apg_write_forward.consistency_mode` 参数设置的更多信息，请参阅 the section called “Aurora PostgreSQL 中写入转发的配置参数”。

等待次数增加的可能原因

等待时间较长的常见原因包括以下几点：

- 副本滞后延长，如 Amazon CloudWatch `ReplicaLag` 指标所衡量。有关该指标的更多信息，请参阅监控 Aurora PostgreSQL 复制。
- 写入器数据库实例或只读副本的负载增加。

操作

根据应用程序的要求更改一致性模式。

IPC:AuroraWriteForwardExecute

当只读副本上的后端进程在等待转发查询完成并从数据库集群写入器节点获取结果时，就会发生该 `IPC:AuroraWriteForwardExecute` 事件。

等待次数增加的可能原因

等待次数增加的常见原因包括以下几点：

- 从写入器节点获取大量行。
- 写入器节点和只读副本之间的网络延迟增加会增加只读副本从写入器节点获得数据所花费的时间。
- 只读副本的负载增加可能会延迟从只读副本向写入器节点传输查询请求的时间。
- 写入器节点负载的增加可能会延迟从写入器节点向只读副本传输数据的时间。

操作

根据等待事件的原因，我们建议采取不同的操作。

- 优化查询以仅检索必要的数据。
- 优化数据操纵语言（DML）操作，使其仅修改必要数据。
- 如果只读副本或写入器节点受到 CPU 或网络带宽的限制，请考虑将其更改为具有更大 CPU 容量或更多网络带宽的实例类型。

IPC:AuroraWriteForwardGetGlobalConsistencyPoint

当使用 GLOBAL 一致性模式的只读副本上的后端进程在执行查询之前等待从写入器节点获取全局一致性点时，就会发生该 `IPC:AuroraWriteForwardGetGlobalConsistencyPoint` 事件。

等待次数增加的可能原因

等待次数增加的常见原因包括以下几点：

- 只读副本和写入器节点之间的网络延迟增加会增加只读副本从写入器节点获得数据所花费的时间。
- 只读副本的负载增加可能会延迟从只读副本向写入器节点传输查询请求的时间。
- 写入器节点负载的增加可能会延迟从写入器节点向只读副本传输数据的时间。

操作

根据等待事件的原因，我们建议采取不同的操作。

- 根据应用程序的要求更改一致性模式。
- 如果只读副本或写入器节点受到 CPU 或网络带宽的限制，请考虑将其更改为具有更大 CPU 容量或更多网络带宽的实例类型。

IPC:AuroraWriteForwardXactAbort

当只读副本上的后端进程在等待远程清理查询的结果时，就会发生该 `IPC:AuroraWriteForwardXactAbort` 事件。在写入转发的事务中止后，会发出清理查询，以将进程恢复到适当的状态。Amazon Aurora 之所以执行这些操作，要么是因为发现了错误，要么是因为用户发出了明确的 `ABORT` 命令或取消了正在运行的查询。

等待次数增加的可能原因

等待次数增加的常见原因包括以下几点：

- 只读副本和写入器节点之间的网络延迟增加会增加只读副本从写入器节点获得数据所花费的时间。
- 只读副本的负载增加可能会延迟从只读副本向写入器节点传输清理查询请求的时间。
- 写入器节点负载的增加可能会延迟从写入器节点向只读副本传输数据的时间。

操作

根据等待事件的原因，我们建议采取不同的操作。

- 调查事务中止的原因。
- 如果只读副本或写入器数据库实例受到 CPU 或网络带宽的限制，请考虑将其更改为具有更大 CPU 容量或更多网络带宽的实例类型。

IPC:AuroraWriteForwardXactCommit

当只读副本上的后端进程在等待转发提交事务命令的结果时，就会发生该 `IPC:AuroraWriteForwardXactCommit` 事件。

等待次数增加的可能原因

等待次数增加的常见原因包括以下几点：

- 只读副本和写入器节点之间的网络延迟增加会增加只读副本从写入器节点获得数据所花费的时间。
- 只读副本的负载增加可能会延迟从只读副本向写入器节点传输查询请求的时间。
- 写入器节点负载的增加可能会延迟从写入器节点向只读副本传输数据的时间。

操作

如果只读副本或写入器节点受到 CPU 或网络带宽的限制，请考虑将其更改为具有更大 CPU 容量或更多网络带宽的实例类型。

IPC:AuroraWriteForwardXactStart

当只读副本上的后端进程在等待转发开始事务命令的结果时，就会发生该 `IPC:AuroraWriteForwardXactStart` 事件。

等待次数增加的可能原因

等待次数增加的常见原因包括以下几点：

- 只读副本和写入器节点之间的网络延迟增加会增加只读副本从写入器节点获得数据所花费的时间。
- 只读副本的负载增加可能会延迟从只读副本向写入器节点传输查询请求的时间。
- 写入器节点负载的增加可能会延迟从写入器节点向只读副本传输数据的时间。

操作

如果只读副本或写入器节点受到 CPU 或网络带宽的限制，请考虑将其更改为具有更大 CPU 容量或更多网络带宽的实例类型。

使用 Aurora PostgreSQL 作为 Amazon Bedrock 的知识库

可以将 Aurora PostgreSQL 数据库集群作为 Amazon Bedrock 知识库。有关更多信息，请参阅在 Amazon Aurora 中创建向量存储。知识库会自动获取存储在 Amazon S3 存储桶中的非结构化文本数据，将其转换为文本块和向量，然后存储在 PostgreSQL 数据库中。借助生成式人工智能应用程序，您可以使用 Amazon Bedrock 的代理来查询存储在知识库中的数据，并使用这些查询的结果来增强基础模型提供的答案。此工作流程称为检索增强生成（RAG）。有关 RAG 的更多信息，请参阅检索增强生成（RAG）。

有关使用 Aurora PostgreSQL 通过 RAG 构建生成式人工智能应用程序的详细信息，请参阅此博客文章。

主题

- 先决条件
- 准备将 Aurora PostgreSQL 用作 Amazon Bedrock 的知识库
- 在 Bedrock 控制台中创建知识库

- 为 Amazon Bedrock 快速创建 Aurora PostgreSQL 知识库

先决条件

要使用 Aurora PostgreSQL 集群作为 Amazon Bedrock 的知识库，请自行熟悉以下先决条件。简而言之，您需要配置以下服务来与 Bedrock 结合使用：

- 使用以下任何版本创建的 Amazon Aurora PostgreSQL 数据库集群：
  - 16.1 及所有更高版本
  - 15.4 及更高版本
  - 14.9 及更高版本
  - 13.12 及更高版本
  - 12.16 及更高版本

  > **Note**
  >
  > 您必须在目标数据库中启用 `pgvector` 扩展并使用版本 0.5.0 或更高版本。有关更多信息，请参阅支持 HNSW 索引的 pgvector 0.5.0 版本。

- RDS 数据 API
- 在 AWS Secrets Manager 中管理的用户。有关更多信息，请参阅 使用 Amazon Aurora 和 AWS Secrets Manager 管理密码。

准备将 Aurora PostgreSQL 用作 Amazon Bedrock 的知识库

按照以下各节中介绍的步骤，准备将 Aurora PostgreSQL 用作 Amazon Bedrock 知识库。

创建和配置 Aurora PostgreSQL

要使用 Aurora PostgreSQL 数据库集群配置 Amazon Bedrock，必须先创建 Aurora PostgreSQL 数据库集群，并记下使用 Amazon Bedrock 对其进行配置的重要字段。有关创建 Aurora PostgreSQL 数据库集群的更多信息，请参阅创建 Aurora PostgreSQL 数据库集群并连接到该集群。

- 创建 Aurora PostgreSQL 数据库集群时启用数据 API。有关支持的版本的更多信息，请参阅使用 Amazon RDS 数据 API。
- 确保记下 Aurora PostgreSQL 数据库集群的 Amazon 资源名称（ARN）。您需要使用它来配置数据库集群，以便与 Amazon Bedrock 结合使用。有关更多信息，请参阅 Amazon 资源名称（ARN）。

连接到数据库并安装 pgvector

可以使用任何连接实用程序连接到 Aurora PostgreSQL。有关这些实用程序的更多详细信息，请参阅连接到 Amazon Aurora PostgreSQL 数据库集群。或者，可以使用 RDS 控制台查询编辑器来运行查询。您需要一个启用了 RDS 数据 API 的 Aurora 数据库集群，才能使用查询编辑器。

1. 以主用户身份登录数据库并设置 pgvector。如果未安装该扩展，请使用以下命令：

   ```
   CREATE EXTENSION IF NOT EXISTS vector;
   ```

   使用支持 HNSW 索引的 pgvector 0.5.0 及更高版本。有关更多信息，请参阅支持 HNSW 索引的 pgvector 0.5.0 版本。
2. 使用以下命令检查已安装的 pg_vector 版本：

   ```
   SELECT extversion FROM pg_extension WHERE extname='vector';
   ```

设置数据库对象和权限

1. 创建 Bedrock 可以用来查询数据的特定架构。使用以下命令创建架构：

   ```
   CREATE SCHEMA bedrock_integration;
   ```
2. 创建 Bedrock 可用来查询数据库的新角色。使用以下命令创建新角色：

   ```
   CREATE ROLE bedrock_user WITH PASSWORD 'password' LOGIN;
   ```

   > **Note**
   >
   > 记下此密码，因为您以后需要这一密码来创建 Secrets Manager 密码。

   如果您使用的是 psql 客户端，则使用以下命令来创建新角色：

   ```
   CREATE ROLE bedrock_user LOGIN;
   \PASSWORD password;
   ```
3. 向 bedrock_user 授予管理 bedrock_integration 架构的权限。这将提供在架构中创建表或索引的功能。

```
GRANT ALL ON SCHEMA bedrock_integration to bedrock_user;
```

4. 以 bedrock_user 身份登录并在 bedrock_integration schema 中创建表。

```
CREATE TABLE bedrock_integration.bedrock_kb (id uuid PRIMARY KEY, embedding
 vector(n), chunks text, metadata json, custom_metadata jsonb);
```

此命令将在 bedrock_integration 架构中创建包含 Titan 嵌入的 bedrock_kb 表。

将 vector(*n*) 数据类型中的 n 替换为对应您正在使用的嵌入模型的合适大小。使用以下建议来帮助选择大小：

- 对于 Titan v2 模型，使用 vector(1024)、vector(512) 或 vector (256)。要了解更多信息，请参阅 Amazon Titan Embeddings Text。
- 对于 Titan v1.2 模型，使用 vector(1536)。要了解更多信息，请参阅 Amazon Titan Multimodal Embeddings G1。
- 对于 Cohere 嵌入模型，使用 vector(1024)。要了解更多信息，请参阅 Cohere Embed models。
- 对于 Cohere Embed Multilingual v3，使用 vector(1024)。

前四列是必需的。为了处理元数据，Bedrock 会将元数据文件中的数据写入 custom_metadata 列。如果您计划使用元数据和筛选，我们建议您创建此列。如果您没有创建 custom_metadata 列，请在开始摄取之前，为表中的每个元数据属性添加单独的列。有关更多信息，请参阅 Configure and customize queries and response generation。

5. 按照以下步骤创建 Bedrock 用于查询数据的必需索引：

- 使用余弦运算符创建索引，Bedrock 可以使用该运算符来查询数据。

```
CREATE INDEX ON bedrock_integration.bedrock_kb USING hnsw (embedding
 vector_cosine_ops);
```

- 对于使用并行索引构建的 pgvector 0.6.0 及更高版本，建议您将 ef_construction 的值设置为 256。

```
CREATE INDEX ON bedrock_integration.bedrock_kb USING hnsw (embedding
 vector_cosine_ops) WITH (ef_construction=256);
```

- 创建 Bedrock 可用来查询文本数据的索引。

```
CREATE INDEX ON bedrock_integration.bedrock_kb USING gin (to_tsvector('simple',
 chunks));
```

- 如果您为自定义元数据创建了一列，请创建一个索引，Bedrock 可以使用该索引来查询元数据。

```
CREATE INDEX ON bedrock_integration.bedrock_kb USING gin (custom_metadata);
```

在 Secrets Manager 中创建密钥

Secrets Manager 可让您存储 Aurora 凭证，这样它们就可以安全地传输到应用程序。如果您在创建 Aurora PostgreSQL 数据库集群时未选择 AWS Secrets Manager 选项，则可以立即创建密钥。有关创建 AWS Secrets Manager 数据库密钥的更多信息，请参阅 AWS Secrets Manager database secret。

在 Bedrock 控制台中创建知识库

在准备将 Aurora PostgreSQL 用作知识库的向量存储时，必须收集您需要提供给 Amazon Bedrock 控制台的以下详细信息。

- Amazon Aurora 数据库集群 ARN – 数据库集群的 ARN。
- 密钥 ARN – 数据库集群的 AWS Secrets Manager 密钥的 ARN。
- 数据库名称 – 数据库的名称。例如，可以使用默认数据库 *postgres*。
- 表名称 - 建议您在创建表时，使用类似于以下内容的命令提供架构限定名称：

```
CREATE TABLE bedrock_integration.bedrock_kb;
```

  此命令将在 bedrock_integration 架构中创建 bedrock_kb 表。
- 创建表时，请务必使用指定的列和数据类型对表进行配置。可以使用您首选的列名称，而不使用表中列出的列名称。请记得记下您在知识库设置过程中选择的以供参考的名称。

列名称	数据类型	描述
id	UUID（主键）	包含每条记录的唯一标识符。
chunks	文本	包含来自数据来源的原始文本块。

列名称	数据类型	描述
嵌入	Vector	包含数据来源的向量嵌入。
metadata	JSON	包含执行来源归因以及启用数据摄取和查询所需的元数据。
custom_metadata	JSONB	（可选）定义 Amazon Bedrock 从数据来源写入元数据详细信息的目标列。

利用这些详细信息，您现在可以在 Bedrock 控制台中创建一个知识库。有关设置向量索引和创建知识库信息的更多详细信息，请参阅 Create a vector store in Amazon Aurora 和 Create a vector store in Amazon Aurora。

将 Aurora 添加为知识库后，您现在可以提取数据来源来进行搜索和查询。有关更多信息，请参阅 Ingest your data sources into the Knowledge Base。

为 Amazon Bedrock 快速创建 Aurora PostgreSQL 知识库

Amazon Bedrock 的检索增强生成（RAG）工作流程依赖存储在 Aurora PostgreSQL 数据库中的向量数据来支持内容检索。过去，要想将 Aurora PostgreSQL 设置为 Bedrock 知识库的向量数据存储，则需要执行多个步骤，而且涉及跨不同用户界面的大量手动操作。这对于数据科学家和开发人员在其 Bedrock 项目中利用 Aurora 构成了挑战。

为改善用户体验，AWS 创建了一个新的基于 CloudFormation 的快速创建选项，可简化设置过程。利用 Aurora 快速创建功能，您现在只需单击一下，即可将预配置的 Aurora PostgreSQL 数据库集群预调配为 Amazon Bedrock 知识库的向量存储。

主题

- 支持的区域和 Aurora PostgreSQL 版本
- 了解快速创建过程
- 使用 Aurora 快速创建的好处
- Aurora 快速创建过程的限制

支持的区域和 Aurora PostgreSQL 版本

Aurora 快速创建选项在所有支持 Amazon Bedrock 知识库的 AWS 区域均可用。默认情况下，它创建版本为 15.7 的 Aurora PostgreSQL 数据库集群。有关支持区域的更多信息，请参阅 Supported models and regions for Amazon Bedrock Knowledge Bases。

了解快速创建过程

快速创建过程可自动预调配以下资源，以将 Amazon Aurora PostgreSQL 数据库设置为 Bedrock 知识库的向量数据存储：

您的账户中的 Aurora PostgreSQL 数据库集群，使用默认设置进行配置。

- ACU（Aurora 容量单位）的设置范围为 0 到 16。这使您的向量存储在不使用时可以缩减到零，从而节省计算成本。可以稍后在 Amazon RDS 控制台中调整 ACU。
- （分层导航小世界）HNSW 索引，使用欧几里得距离作为存储在 Aurora 中的 Bedrock 向量嵌入的相似度量。
- 数据库实例是无服务器 v2 实例。
- 该集群与默认 VPC 和子网关联，并已启用 RDS 数据 API。
- 集群管理员凭证由 AWS Secrets Manager 管理。

除默认设置外，还将为您设置以下设置。当您进行该过程时，会看到解释工作流程的屏幕。

- 使用必要的数据库对象播种 Aurora 集群：
  - 创建 Bedrock 知识库所需的 pgvector 扩展、架构、角色和表。
  - 注册一个有限权限的数据库用户，以便 Bedrock 与集群进行交互。
- 在整个资源预调配过程中将显示进度横幅，便于您跟踪以下子事件的状态：
  - 创建 Aurora 集群
  - 播种 Aurora 集群
  - 创建知识库

  即使您离开页面并返回，在知识库完全创建之前，横幅一直可见。
- 您可以单击进度横幅上的 `View details` 来查看每个步骤的状态。有关知识库创建期间的事件的更多信息，请在查看详细信息屏幕中选择 CloudFormation 链接。该过程完成后，您的新 Bedrock 知识库即可供使用。
- 如果您需要引用快速创建资源的堆栈 ID，在 Bedrock 知识库的标签中都能找到。

通过配置新近预调配的 Aurora 集群作为向量存储，一个 Bedrock 知识库就这样创建好了。

使用 Aurora 快速创建的好处

- 基于 CloudFormation 的快速创建过程大大减少了使用 Aurora 作为向量存储所需的时间和复杂性。
- Aurora 提供出色的性能、向量可扩展性和成本优势，并且能够缩减至零计算费用（不使用计算时）。
- 快速创建过程简化了端到端体验，使您可以使用 Aurora 轻松创建和配置 Bedrock 知识库。
- 客户可以基于 CloudFormation 模板进行构建，使用自己的配置来自定义预调配。

Aurora 快速创建过程的限制

- 使用 Aurora 快速创建选项时，将用默认配置来预调配数据库集群。但是，这些默认设置可能并不符合您的特定需求或预期用例。快速创建不提供在预调配过程中修改配置的选项。配置是自动设置的，以简化部署体验。如果您需要自定义 Aurora 数据库集群配置，可以在初始部署后通过在 Amazon RDS 控制台中快速创建来实现。
- 虽然快速创建过程简化了设置过程，但创建 Aurora 数据库集群的时间仍约为 10 分钟，与手动部署相当。这是因为预调配 Aurora 基础结构需要时间。
- 快速创建选项适合实验和快速设置用途。通过快速创建所创建的资源可能不适合生产用途，您不能直接将其迁移到 VPC 中的生产环境。

将 Amazon Aurora PostgreSQL 与其他AWS服务集成

Amazon Aurora 与其他AWS服务集成在一起，因此，您可以扩展 Aurora PostgreSQL 数据库集群以在AWS云中使用其他功能。您的 Aurora PostgreSQL 数据库集群可以使用AWS服务来执行以下操作：

- 使用 Amazon RDS 性能详情快速收集、查看和评估 Aurora PostgreSQL 数据库实例的性能。性能详情在现有 Amazon RDS 监控功能的基础上进行了扩展，以便通过示意图展示您的数据库的性能并帮助您分析影响性能的任何问题。利用性能详情控制面板，您可以可视化数据库负载并按等待状态、SQL 语句、主机或用户来筛选负载。有关性能详情的更多信息，请参阅 在 Amazon Aurora 上使用性能详情监控数据库负载。
- 配置 Aurora PostgreSQL 数据库集群以将日志数据发布到 Amazon CloudWatch Logs。CloudWatch Logs 可以为您的日志记录提供高持久性存储。利用 CloudWatch Logs，可以对日志数据进行实时分析并使用 CloudWatch 创建警报和查看指标。有关更多信息，请参阅 将 Aurora PostgreSQL 日志发布到 Amazon CloudWatch Logs。

- 将数据从 Amazon S3 存储桶导入 Aurora PostgreSQL 数据库集群中，或将数据从 Aurora PostgreSQL 数据库集群导出至 Amazon S3 存储桶。有关更多信息，请参阅 将 Amazon S3 中的数据导入到 Aurora PostgreSQL 数据库集群 和 将数据从 Aurora PostgreSQL 数据库集群导出到 Amazon S3。
- 使用 SQL 语言将基于机器学习的预测结果添加至数据库应用程序中。Aurora 机器学习使用 Aurora 数据库和 AWS 机器学习（ML）服务 SageMaker AI 以及 Amazon Comprehend 之间的高度优化集成。有关更多信息，请参阅 将 Amazon Aurora 机器学习与 Aurora PostgreSQL 结合使用。
- 从 Aurora PostgreSQL 数据库集群中调用 AWS Lambda 函数。为此，请使用 Aurora PostgreSQL 随附的 `aws_lambda` PostgreSQL 扩展。有关更多信息，请参阅 从 Aurora PostgreSQL 数据库集群中调用 AWS Lambda 函数。
- 集成来自 Amazon Redshift 和 Aurora PostgreSQL 的查询。有关更多信息，请参阅 Amazon Redshift 数据库开发人员指南中的开始对 PostgreSQL 使用联合查询。

将 Amazon S3 中的数据导入到 Aurora PostgreSQL 数据库集群

您可以将已使用 Amazon Simple Storage Service 存储的数据导入到 Aurora PostgreSQL 数据库集群上的表中。为此，您首先要安装 Aurora PostgreSQL `aws_s3` 扩展。该扩展提供用于从 AmazonS3 桶导入数据的函数。桶 是用于存储对象和文件的 Amazon S3 容器。数据可以位于逗号分隔值（CSV）文件、文本文件或压缩（gzip）文件中。接下来，您可以了解如何安装扩展以及如何将数据从 Amazon S3 导入到表中。

要从 Amazon S3 导入到 ，您的数据库必须运行 PostgreSQL 版本 10.7 或更高版本。Aurora PostgreSQL。

如果您没有将数据存储在 Amazon S3 上，则需要先创建桶并存储数据。有关更多信息，请参阅《Amazon Simple Storage Service 用户指南》中的以下主题。

- 创建存储桶
- 向存储桶添加对象

支持从 Amazon S3 跨账户导入。有关更多信息，请参阅《Amazon Simple Storage Service 用户指南》中的授予跨账户权限。

从 S3 导入数据时，您可以使用客户托管式密钥进行加密。有关更多信息，请参阅《Amazon Simple Storage Service 用户指南》中的 AWS KMS 中存储的 KMS 密钥。

> **Note**
>
> 对于 Aurora Serverless v1，不支持从 Amazon S3 导入数据。Aurora Serverless v2 支持此功能。

主题

- 安装 aws_s3 扩展名
- 从 Amazon S3 数据导入数据概述
- 设置 Amazon S3 存储桶的访问权限
- 将数据从 Amazon S3 导入到 Aurora PostgreSQL 数据库集群
- 函数参考

安装 aws_s3 扩展名

在将 Amazon S3 用于 Aurora PostgreSQL 数据库集群之前，您需要安装 aws_s3 扩展。此扩展提供从 Amazon S3 导入数据的函数。它还提供将数据从 Aurora PostgreSQL 数据库集群的实例中导出到 Amazon S3 桶的函数。有关更多信息，请参阅 将数据从 Aurora PostgreSQL 数据库集群导出到 Amazon S3。aws_s3 扩展依赖于 aws_commons 扩展（需要时自动安装）中的一些帮助程序函数。

安装 **aws_s3** 扩展

1. 使用 psql（或 pgAdmin）以具有 rds_superuser 权限的用户身份，连接到 Aurora PostgreSQL 数据库集群的写入器实例。如果您在设置过程中保留原定设置名称，则以 postgres 进行连接。

   ```
   psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --
   username=postgres --password
   ```

2. 要安装扩展，请运行以下命令。

   ```
   postgres=> CREATE EXTENSION aws_s3 CASCADE;
   NOTICE: installing required extension "aws_commons"
   CREATE EXTENSION
   ```

3. 要验证扩展是否已安装，可以使用 psql \dx 元命令。

   ```
   postgres=> \dx
          List of installed extensions
   ```

```
    Name     | Version |   Schema   |                Description
-------------+---------+------------+-------------------------------------------
 aws_commons | 1.2     | public     | Common data types across AWS services
 aws_s3      | 1.1     | public     | AWS S3 extension for importing data from S3
 plpgsql     | 1.0     | pg_catalog | PL/pgSQL procedural language
(3 rows)
```

从 Amazon S3 导入数据和将数据导出到 Amazon S3 的函数现在可供使用。

从 Amazon S3 数据导入数据概述

将 S3 数据导入到 Aurora PostgreSQL

首先，收集您需要为该函数提供的详细信息。其中包括 Aurora PostgreSQL 数据库集群的实例上的表名称、桶名称、文件路径、文件类型以及存储 Amazon S3 数据的 AWS 区域。有关更多信息，请参阅《Amazon Simple Storage Service 用户指南》中的查看对象。

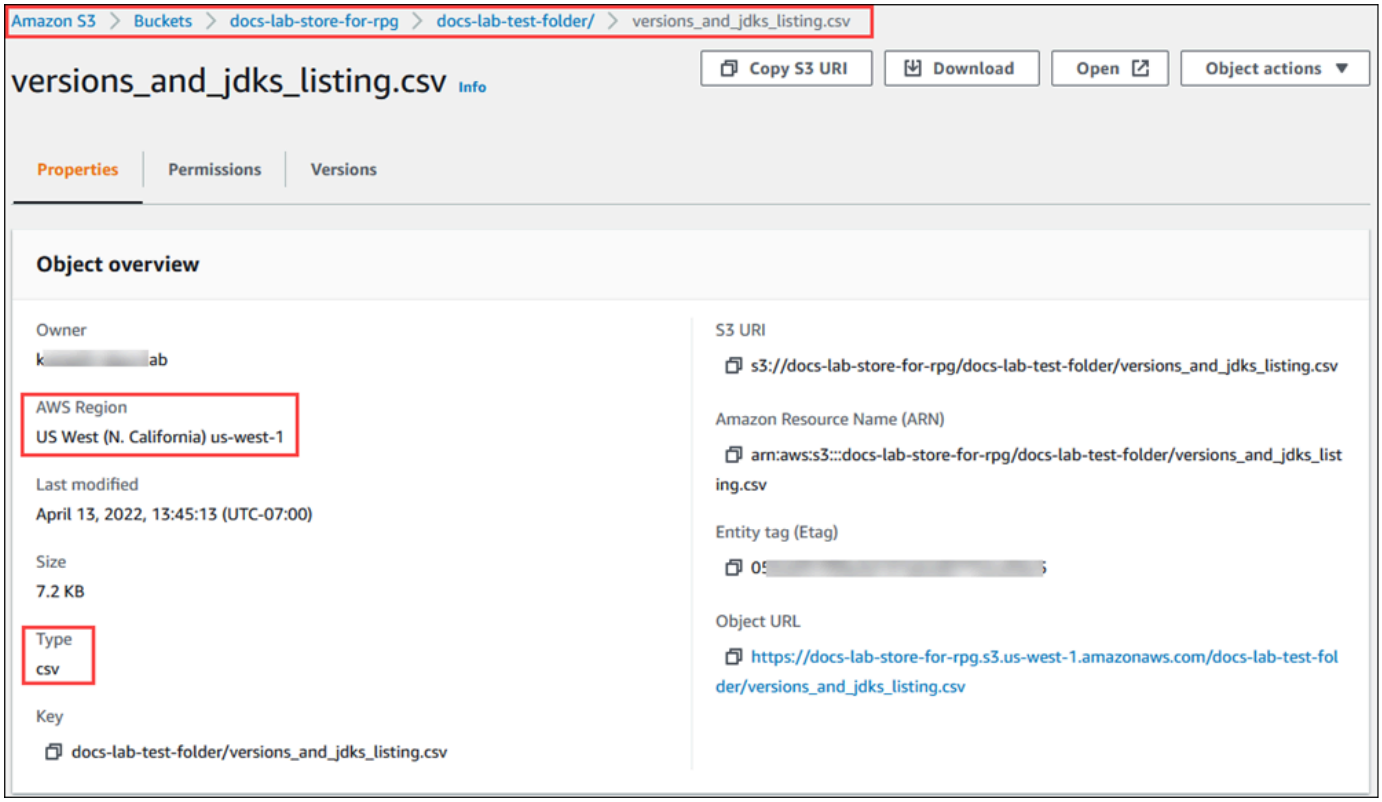
> **Note**
>
> 目前不支持从 Amazon S3 导入分段数据。

1. 获取 `aws_s3.table_import_from_s3` 函数要向其中导入数据的表的名称。例如，以下命令创建表 `t1`，供在后面的步骤中使用。

   ```
   postgres=> CREATE TABLE t1
       (col1 varchar(80),
       col2 varchar(80),
       col3 varchar(80));
   ```

2. 获取有关 Amazon S3 桶和要导入的数据的详细信息。为此，请通过以下网址打开 Amazon S3 控制台：https://console.aws.amazon.com/s3/，然后选择 Buckets（桶）。在列表中找到包含您的数据的桶。选择桶，打开其 Object overview（对象概述）页面，然后选择 Properties（属性）。

   记下桶名称、路径、AWS 区域和文件类型。您稍后需要提供 Amazon 资源名称（ARN），以便通过 IAM 角色设置对 Amazon S3 的访问权限。有关更多信息，请参阅设置 Amazon S3 存储桶的访问权限。下图显示了一个示例。



3. 您可以使用 AWS CLI 命令 `aws s3 cp` 验证 AmazonS3 桶上数据的路径。如果该信息正确无误，该命令将下载 Amazon S3 文件的副本。

```
aws s3 cp s3://amzn-s3-demo-bucket/sample_file_path ./
```

4. 在 Aurora PostgreSQL 数据库集群上设置权限，以允许访问 Amazon S3 桶中的文件。为此，您可以使用 AWS Identity and Access Management（IAM）角色或安全凭证。有关更多信息，请参阅[设置 Amazon S3 存储桶的访问权限]。

5. 将收集的路径和其他 Amazon S3 对象详细信息（请参阅步骤 2）提供给用于构造 Amazon S3 URI 对象的 `create_s3_uri` 函数。要了解有关此函数的更多信息，请参阅[aws_commons.create_s3_uri]。以下是在 psql 会话期间构造此对象的示例。

```
postgres=> SELECT aws_commons.create_s3_uri(
   'docs-lab-store-for-rpg',
   'versions_and_jdks_listing.csv',
   'us-west-1'
) AS s3_uri \gset
```

在下一步中，您将此对象（`aws_commons._s3_uri_1`）传递到 `aws_s3.table_import_from_s3` 函数，以便将数据导入表中。

6. 调用 `aws_s3.table_import_from_s3` 函数，以将数据从 Amazon S3 导入到您的表中。有关参考信息，请参阅aws_s3.table_import_from_s3。有关示例，请参阅 将数据从 Amazon S3 导入到 Aurora PostgreSQL 数据库集群。

设置 Amazon S3 存储桶的访问权限

要从 Amazon S3 文件中导入数据，请为 Aurora PostgreSQL 数据库集群 提供权限以访问包含该文件的 Amazon S3 存储桶。您可以通过两种方式提供 Amazon S3 存储桶的访问权限，如以下主题中所述。

主题

- 使用 IAM 角色访问 Amazon S3 存储桶
- 使用安全凭证访问 Amazon S3 存储桶
- Amazon S3 访问故障排除

使用 IAM 角色访问 Amazon S3 存储桶

从 Amazon S3 文件中加载数据之前，请为 Aurora PostgreSQL 数据库集群提供权限以访问该文件所在的 Amazon S3 存储桶。这样，您无需管理其他凭证信息或在 aws_s3.table_import_from_s3 函数调用中提供该信息。

为此，请创建一个 IAM 策略以提供 Amazon S3 存储桶的访问权限。创建一个 IAM 角色并将策略附加到该角色。然后，将该 IAM 角色分配给数据库集群。

> **Note**
>
> 您无法将 IAM 角色与 Aurora Serverless v1 数据库集群关联，因此以下步骤不适用。

通过 IAM 角色授予 Aurora PostgreSQL 数据库集群访问 Amazon S3 的权限

1. 创建一个 IAM 策略。

   该策略提供存储桶和对象权限，以允许 Aurora PostgreSQL 数据库集群访问 Amazon S3。

   在策略中包含以下必需操作，以允许将文件从 Amazon S3 存储桶传输到 Aurora PostgreSQL：

   - `s3:GetObject`

- `s3:ListBucket`

在策略中包含以下资源以标识 Amazon S3 存储桶以及存储桶中的对象。这会显示用于访问 Amazon S3 的 Amazon Resource Name (ARN) 格式。

- arn:aws:s3:::*amzn-s3-demo-bucket*
- arn:aws:s3:::*amzn-s3-demo-bucket*/*

有关为 Aurora PostgreSQL 创建 IAM policy 的更多信息，请参阅创建和使用适用于 IAM 数据库访问的 IAM 策略。另请参阅 IAM 用户指南中的教程：创建和附加您的第一个客户托管策略。

以下 AWS CLI 命令使用这些选项创建一个名为 `rds-s3-import-policy` 的 IAM 策略。该策略授予对名为 *amzn-s3-demo-bucket* 的存储桶的访问权限。

> **Note**
>
> 记下此命令返回的策略的 Amazon 资源名称（ARN）。在后续步骤中将策略附加到 IAM 角色时，您需要此 ARN。

Example

对于 Linux、macOS 或 Unix：

```
aws iam create-policy \
   --policy-name rds-s3-import-policy \
   --policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Sid": "s3import",
         "Action": [
           "s3:GetObject",
           "s3:ListBucket"
         ],
         "Effect": "Allow",
         "Resource": [
           "arn:aws:s3:::amzn-s3-demo-bucket",
           "arn:aws:s3:::amzn-s3-demo-bucket/*"
```

```
        ]
      }
    ]
  }'
```

对于 Windows：

```
aws iam create-policy ^
   --policy-name rds-s3-import-policy ^
   --policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Sid": "s3import",
         "Action": [
           "s3:GetObject",
           "s3:ListBucket"
         ],
         "Effect": "Allow",
         "Resource": [
           "arn:aws:s3:::amzn-s3-demo-bucket",
           "arn:aws:s3:::amzn-s3-demo-bucket/*"
         ]
       }
     ]
   }'
```

2. 创建一个 IAM 角色。

   这样，Aurora PostgreSQL 就可以担任该 IAM 角色以访问您的 Amazon S3 存储桶。有关更多信息，请参阅《IAM 用户指南》中的创建向 IAM 用户委派权限的角色。

   我们建议在基于资源的策略中使用 aws:SourceArn 和 aws:SourceAccount 全局条件上下文键，以限制对特定资源的服务权限。这是防范混淆代理人问题最有效的方法。

   如果同时使用全局条件上下文键和包含账户 ID 的 aws:SourceArn 值，则 aws:SourceAccount 值和 aws:SourceArn 值中的账户在同一策略语句中使用时，必须使用相同的账户 ID。

   - 如果您想对单个资源进行跨服务访问，请使用 aws:SourceArn。
   - 如果您想允许该账户中的任何资源与跨服务使用操作相关联，请使用 aws:SourceAccount。

在策略中，确保使用具有资源的完整 ARN 的 `aws:SourceArn` 全局条件上下文键。以下示例说明了如何使用 AWS CLI 命令创建一个名为 `rds-s3-import-role` 的角色来实现该目的。

Example

对于 Linux、macOS 或 Unix：

```
aws iam create-role \
   --role-name rds-s3-import-role \
   --assume-role-policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Principal": {
            "Service": "rds.amazonaws.com"
          },
         "Action": "sts:AssumeRole",
         "Condition": {
             "StringEquals": {
                "aws:SourceAccount": "111122223333",
                "aws:SourceArn": "arn:aws:rds:us-
east-1:111122223333:cluster:clustername"
                }
             }
       }
     ]
   }'
```

对于 Windows：

```
aws iam create-role ^
   --role-name rds-s3-import-role ^
   --assume-role-policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Principal": {
            "Service": "rds.amazonaws.com"
          },
```

```
          "Action": "sts:AssumeRole",
          "Condition": {
              "StringEquals": {
                 "aws:SourceAccount": "111122223333",
                 "aws:SourceArn": "arn:aws:rds:us-
east-1:111122223333:cluster:clustername"
                 }
              }
       }
     ]
   }'
```

3. 将您创建的 IAM 策略附加到您创建的 IAM 角色。

   以下 AWS CLI 命令将上一步中创建的策略附加到一个名为 `rds-s3-import-role` 的角色。请将 *`your-policy-arn`* 替换为您在前面的步骤中记下的策略 ARN。

   Example

   对于 Linux、macOS 或 Unix：

```
aws iam attach-role-policy \
   --policy-arn your-policy-arn \
   --role-name rds-s3-import-role
```

   对于 Windows：

```
aws iam attach-role-policy ^
   --policy-arn your-policy-arn ^
   --role-name rds-s3-import-role
```

4. 将该 IAM 角色添加到数据库集群中。

   您可以使用 AWS Management Console 或 AWS CLI 执行该操作，如下所述。

控制台

使用控制台为 PostgreSQL 数据库集群添加 IAM 角色

1. 登录AWS Management Console并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。

2. 选择 PostgreSQL 数据库集群名称以显示其详细信息。
3. 在连接性和安全性选项卡上的管理 IAM 角色部分中，在向此集群添加 IAM 角色下选择要添加的角色。
4. 在 Feature (功能) 下，选择 s3Import。
5. 选择 Add role (添加角色)。

AWS CLI

使用 CLI 为 PostgreSQL 数据库实例添加 IAM 角色

- 使用以下命令将角色添加到名为 `my-db-cluster` 的 PostgreSQL 数据库集群中。将 *`your-role-arn`* 替换为您在上一步中记下的角色 ARN。使用 `s3Import` 作为 `--feature-name` 选项的值。

  Example

  对于 Linux、macOS 或 Unix：

  ```
  aws rds add-role-to-db-cluster \
     --db-cluster-identifier my-db-cluster \
     --feature-name s3Import \
     --role-arn your-role-arn   \
     --region your-region
  ```

  对于 Windows：

  ```
  aws rds add-role-to-db-cluster ^
     --db-cluster-identifier my-db-cluster ^
     --feature-name s3Import ^
     --role-arn your-role-arn ^
     --region your-region
  ```

RDS API

要使用 Amazon RDS API 为 PostgreSQL 数据库集群添加 IAM 角色，请调用 AddRoleToDBCluster 操作。

使用安全凭证访问 Amazon S3 存储桶

如果愿意，您可以使用安全凭证提供 Amazon S3 存储桶的访问权限，而不是使用 IAM 角色提供访问权限。这可以通过在 aws_s3.table_import_from_s3 函数调用中指定 credentials 参数来实现。

credentials 参数是 aws_commons._aws_credentials_1 类型的结构，其中包含 AWS 凭证。使用 aws_commons.create_aws_credentials 函数设置 aws_commons._aws_credentials_1 结构中的访问密钥和私有密钥，如下所示。

```
postgres=> SELECT aws_commons.create_aws_credentials(
   'sample_access_key', 'sample_secret_key', '')
AS creds \gset
```

在创建 aws_commons._aws_credentials_1 结构后，请将 aws_s3.table_import_from_s3 函数与 credentials 参数结合使用来导入数据，如下所示。

```
postgres=> SELECT aws_s3.table_import_from_s3(
   't', '', '(format csv)',
   :'s3_uri',
   :'creds'
);
```

也可以在 aws_commons.create_aws_credentials 函数调用中以内联方式包括 aws_s3.table_import_from_s3 函数调用。

```
postgres=> SELECT aws_s3.table_import_from_s3(
   't', '', '(format csv)',
   :'s3_uri',
   aws_commons.create_aws_credentials('sample_access_key', 'sample_secret_key', '')
);
```

Amazon S3 访问故障排除

如果在尝试从 Amazon S3 导入数据时遇到连接问题，请参阅以下内容以了解相应的建议：

- 对 Amazon Aurora 身份和访问权限问题进行故障排除
- Amazon Simple Storage Service 用户指南中的排查 Amazon S3 的问题
- IAM 用户指南中的排查 Amazon S3 和 IAM 的问题

将数据从 Amazon S3 导入到 Aurora PostgreSQL 数据库集群

您可以使用 aws_S3 扩展的 `table_import_from_s3` 函数从 Amazon S3 桶导入数据。有关参考信息，请参阅aws_s3.table_import_from_s3。

> **Note**
>
> 以下示例使用 IAM 角色方法以允许访问 Amazon S3 桶。因此，`aws_s3.table_import_from_s3` 函数调用不包括凭据参数。

下面显示典型示例。

```
postgres=> SELECT aws_s3.table_import_from_s3(
   't1',
   '',
   '(format csv)',
   :'s3_uri'
);
```

下面是参数：

- `t1` – 将数据复制到的 PostgreSQL 数据库集群中的表的名称。
- `''` – 数据库表中的列的可选列表。您可以使用此参数来指示哪些 S3 数据列进入哪些表列中。如果未指定任何列，则会将所有列复制到表中。有关使用列列表的示例，请参阅 导入使用自定义分隔符的 Amazon S3 文件。
- `(format csv)` – PostgreSQL COPY 参数。复制过程使用 PostgreSQL COPY 命令的参数和格式以导入数据。格式选择包括本例中所示的逗号分隔值（CSV），以及文本和二进制。原定设置为文本。
- `s3_uri` – 包含标识 Amazon S3 文件的信息的结构。有关使用 aws_commons.create_s3_uri 函数创建 `s3_uri` 结构的示例，请参阅 从 Amazon S3 数据导入数据概述。

有关此函数的更多信息，请参阅 aws_s3.table_import_from_s3。

`aws_s3.table_import_from_s3` 函数返回文本。要指定要从 Amazon S3 存储桶中导入的其他类型的文件，请参阅以下示例之一。

> **Note**
>
> 导入 0 字节文件将导致错误。

主题

- 导入使用自定义分隔符的 Amazon S3 文件
- 导入 Amazon S3 压缩 (gzip) 文件
- 导入编码的 Amazon S3 文件

导入使用自定义分隔符的 Amazon S3 文件

以下示例说明了如何导入使用自定义分隔符的文件。它还说明如何使用 `column_list` 函数的 aws_s3.table_import_from_s3 参数来控制将数据放置在数据库表中的哪个位置。

在此示例中，假定将以下信息组织到 Amazon S3 文件中的竖线分隔列中。

```
1|foo1|bar1|elephant1
2|foo2|bar2|elephant2
3|foo3|bar3|elephant3
4|foo4|bar4|elephant4
...
```

导入使用自定义分隔符的文件

1. 在数据库中为导入的数据创建一个表。

   ```
   postgres=> CREATE TABLE test (a text, b text, c text, d text, e text);
   ```

2. 使用以下形式的 aws_s3.table_import_from_s3 函数从 Amazon S3 文件导入数据。

   您可以在 aws_commons.create_s3_uri 函数调用中以内联方式包括 `aws_s3.table_import_from_s3` 函数调用来指定文件。

   ```
   postgres=> SELECT aws_s3.table_import_from_s3(
      'test',
      'a,b,d,e',
      'DELIMITER ''|''',
      aws_commons.create_s3_uri('amzn-s3-demo-bucket', 'pipeDelimitedSampleFile', 'us-
   east-2')
   ```

```
);
```

数据现在位于表的以下列中。

```
postgres=> SELECT * FROM test;
a | b | c | d | e
---+------+---+---+------+-----------
1 | foo1 | | bar1 | elephant1
2 | foo2 | | bar2 | elephant2
3 | foo3 | | bar3 | elephant3
4 | foo4 | | bar4 | elephant4
```

导入 Amazon S3 压缩 (gzip) 文件

以下示例说明如何从 Amazon S3 导入使用 gzip 压缩的文件。导入的文件需要具有以下 Amazon S3 元数据：

- 键：`Content-Encoding`
- 值：`gzip`

如果使用 AWS Management Console 上传文件，则元数据通常由系统应用。有关使用 AWS Management Console、AWS CLI 或 API 将文件上传到 Amazon S3 的信息，请参阅《Amazon Simple Storage Service 用户指南》中的上传对象。

有关 Amazon S3 元数据的更多信息以及有关系统提供的元数据的详细信息，请参阅《Amazon Simple Storage Service 用户指南》中的在 Amazon S3 控制台中编辑对象元数据。

将 gzip 文件导入到 Aurora PostgreSQL 数据库集群，如下所示。

```
postgres=> CREATE TABLE test_gzip(id int, a text, b text, c text, d text);
postgres=> SELECT aws_s3.table_import_from_s3(
 'test_gzip', '', '(format csv)',
 'amzn-s3-demo-bucket', 'test-data.gz', 'us-east-2'
);
```

导入编码的 Amazon S3 文件

以下示例说明如何从 Amazon S3 导入具有 Windows-1252 编码的文件。

```
postgres=> SELECT aws_s3.table_import_from_s3(
```

```
 'test_table', '', 'encoding ''WIN1252''',
 aws_commons.create_s3_uri('amzn-s3-demo-bucket', 'SampleFile', 'us-east-2')
);
```

函数参考

函数

- aws_s3.table_import_from_s3
- aws_commons.create_s3_uri
- aws_commons.create_aws_credentials

aws_s3.table_import_from_s3

将 Amazon S3 数据导入到 Aurora PostgreSQL 表中。aws_s3 扩展提供 aws_s3.table_import_from_s3 函数。返回值为文本。

语法

必需的参数为 table_name、column_list 和 options。这些标识数据库表并指定如何将数据复制到表中。

您还可以使用以下参数：

- s3_info 参数指定要导入的 Amazon S3 文件。在您使用此参数时，IAM 角色为 PostgreSQL 数据库集群提供访问 Amazon S3 的权限。

```
aws_s3.table_import_from_s3 (
   table_name text,
   column_list text,
   options text,
   s3_info aws_commons._s3_uri_1
)
```

- credentials 参数指定凭证以访问 Amazon S3。在您使用此参数时，不使用 IAM 角色。

```
aws_s3.table_import_from_s3 (
   table_name text,
   column_list text,
   options text,
   s3_info aws_commons._s3_uri_1,
```

```
   credentials aws_commons._aws_credentials_1
)
```

参数

table_name

包含要将数据导入到的 PostgreSQL 数据库表的名称的必需文本字符串。

column_list

包含要将数据复制到的 PostgreSQL 数据库表列的可选列表的必需文本字符串。如果此字符串为空，将使用表的所有列。有关示例，请参阅导入使用自定义分隔符的 Amazon S3 文件。

options

包含 PostgreSQL COPY 命令的参数的必需文本字符串。这些参数指定如何将数据复制到 PostgreSQL 表中。有关更多详细信息，请参阅 PostgreSQL COPY 文档。

s3_info

包含有关 S3 对象的以下信息的 aws_commons._s3_uri_1 复合类型：

- bucket – 包含文件的 Amazon S3 存储桶的名称。
- file_path – 包含文件路径的 Amazon S3 文件名。
- region – 文件所在的AWS区域。有关 AWS 区域名称和关联值的列表，请参阅区域及可用区。

凭证

包含以下用于导入操作的凭证的 aws_commons._aws_credentials_1 复合类型：

- 访问密钥
- 私有密钥
- 会话令牌

有关创建 aws_commons._aws_credentials_1 复合结构的信息，请参阅 aws_commons.create_aws_credentials。

替代语法

为帮助进行测试，您可以使用一组扩展的参数而非 s3_info 和 credentials 参数。下面是 aws_s3.table_import_from_s3 函数的其他语法变化。

- 不使用 `s3_info` 参数来标识 Amazon S3 文件，而使用 `bucket`、`file_path` 和 `region` 参数的组合。使用此形式的函数，IAM 角色在 PostgreSQL 数据库实例上提供访问 Amazon S3 的权限。

```
aws_s3.table_import_from_s3 (
   table_name text,
   column_list text,
   options text,
   bucket text,
   file_path text,
   region text
)
```

- 不使用 `credentials` 参数来指定 Amazon S3 访问权限，而使用 `access_key`、`session_key` 和 `session_token` 参数的组合。

```
aws_s3.table_import_from_s3 (
   table_name text,
   column_list text,
   options text,
   bucket text,
   file_path text,
   region text,
   access_key text,
   secret_key text,
   session_token text
)
```

替代参数

bucket

包含 Amazon S3 存储桶（其中包含文件）的名称的文本字符串。

file_path

包含 Amazon S3 文件名（包含文件路径）的文本字符串。

region

标识文件的 AWS 区域位置的文本字符串。有关 AWS 区域名称和关联值的列表，请参阅区域及可用区。

access_key

包含用于导入操作的访问密钥的文本字符串。默认值为 NULL。

secret_key

包含用于导入操作的私有密钥的文本字符串。默认值为 NULL。

session_token

（可选）包含用于导入操作的会话密钥的文本字符串。默认值为 NULL。

aws_commons.create_s3_uri

创建 aws_commons._s3_uri_1 结构来保存 Amazon S3 文件信息。在 aws_commons.create_s3_uri 函数的 s3_info 参数中使用 aws_s3.table_import_from_s3 函数的结果。

语法

```
aws_commons.create_s3_uri(
   bucket text,
   file_path text,
   region text
)
```

参数

bucket

包含文件的 Amazon S3 存储桶名称的必需文本字符串。

file_path

包含 Amazon S3 文件名（包含文件路径）的必填文本字符串。

region

一个包含文件所在的 AWS 区域的必需文本字符串。有关 AWS 区域名称和关联值的列表，请参阅区域及可用区。

aws_commons.create_aws_credentials

在 aws_commons._aws_credentials_1 结构中设置访问密钥和私有密钥。在 aws_commons.create_aws_credentials 函数的 credentials 参数中使用 aws_s3.table_import_from_s3 函数的结果。

语法

```
aws_commons.create_aws_credentials(
   access_key text,
   secret_key text,
   session_token text
)
```

参数

access_key

包含用于导入 Amazon S3 文件的访问密钥的必需文本字符串。默认值为 NULL。

secret_key

包含用于导入 Amazon S3 文件的私有密钥的必需文本字符串。默认值为 NULL。

session_token

包含用于导入 Amazon S3 文件的会话令牌的可选文本字符串。默认值为 NULL。如果您提供了可选的 session_token，则可以使用临时凭证。

将数据从 Aurora PostgreSQL 数据库集群导出到 Amazon S3

您可以从 Aurora PostgreSQL 数据库集群中查询数据，并将数据直接导出到存储在 Amazon S3 存储桶中的文件中。为此，您首先要安装 Aurora PostgreSQL aws_s3 扩展。该扩展为您提供了用于将查询结果导出到 Amazon S3 的函数。接下来，您可以了解如何安装扩展以及如何将数据导出到 Amazon S3。

您可以从预调配或 Aurora Serverless v2 数据库实例中导出。Aurora Serverless v1 不支持这些步骤。

> **Note**
>
> 不支持跨账户导出到 Amazon S3。

所有当前可用的 Aurora PostgreSQL 版本都支持将数据导出到 Amazon Simple Storage Service。有关详细版本信息，请参阅《Aurora PostgreSQL 版本注释》中的 Amazon Aurora PostgreSQL 更新。

如果您没有为导出设置桶，请参阅《Amazon Simple Storage Service 用户指南》中的以下主题。

- 设置 Amazon S3
- 创建存储桶

默认情况下，从 Aurora PostgreSQL 导出到 Amazon S3 的数据使用具有 AWS 托管式密钥的服务器端加密。您也可以使用您已创建的客户自主管理型密钥。如果您使用存储桶加密，则 Amazon S3 存储桶必须使用 AWS Key Management Service（AWS KMS）密钥（SSE-KMS）进行加密。目前，不支持使用 Amazon S3 托管式密钥（SSE-S3）加密的存储桶。

> **Note**
>
> 您可以使用AWS Management Console、AWS CLI 或 Amazon RDS API 将数据库和数据库集群快照数据保存到 Amazon S3。有关更多信息，请参阅 将数据库集群快照数据导出到 Amazon S3。

主题

- 安装 aws_s3 扩展名
- 将数据导出到 Amazon S3 概述
- 指定要导出到的 Amazon S3 文件路径
- 设置 Amazon S3 存储桶的访问权限
- 使用 aws_s3.query_export_to_s3 函数导出查询数据
- 函数参考
- Amazon S3 访问故障排除

安装 aws_s3 扩展名

在将 Amazon Simple Storage Service 用于 Aurora PostgreSQL 数据库集群之前，您需要安装 aws_s3 扩展。此扩展提供将数据从 Aurora PostgreSQL 数据库集群的写入器实例中导出到 Amazon S3 桶的函数。它还提供从 Amazon S3 导入数据的函数。有关更多信息，请参阅 将 Amazon S3 中的数据导入到 Aurora PostgreSQL 数据库集群。aws_s3 扩展依赖于 aws_commons 扩展（需要时自动安装）中的一些帮助程序函数。

安装 **aws_s3** 扩展

1. 使用 psql（或 pgAdmin）以具有 rds_superuser 权限的用户身份，连接到 Aurora PostgreSQL 数据库集群的写入器实例。如果您在设置过程中保留原定设置名称，则以 postgres 进行连接。

```
psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password
```

2. 要安装扩展，请运行以下命令。

```
postgres=> CREATE EXTENSION aws_s3 CASCADE;
NOTICE: installing required extension "aws_commons"
CREATE EXTENSION
```

3. 要验证扩展是否已安装，可以使用 psql \dx 元命令。

```
postgres=> \dx
       List of installed extensions
    Name     | Version |   Schema   |                Description
-------------+---------+------------+-------------------------------------------
 aws_commons | 1.2     | public     | Common data types across AWS services
 aws_s3      | 1.1     | public     | AWS S3 extension for importing data from S3
 plpgsql     | 1.0     | pg_catalog | PL/pgSQL procedural language
(3 rows)
```

从 Amazon S3 导入数据和将数据导出到 Amazon S3 的函数现在可供使用。

验证您的 Aurora PostgreSQL 版本支持导出到 Amazon S3

您可以使用 describe-db-engine-versions 命令验证 Aurora PostgreSQL 版本是否支持导出到 Amazon S3。以下示例检查版本 10.14 是否可以导出到 Amazon S3。

```
aws rds describe-db-engine-versions --region us-east-1 \
--engine aurora-postgresql --engine-version 10.14 | grep s3Export
```

如果输出包含字符串 "s3Export"，则该引擎支持 Amazon S3 导出。否则，该引擎不支持它们。

将数据导出到 Amazon S3 概述

要将存储在 Aurora PostgreSQL 数据库中的数据导出到 Amazon S3 存储桶，请使用以下过程。

将 Aurora PostgreSQL 数据导出到 S3

1. 标识要用于导出数据的 Amazon S3 文件路径。有关此过程的详细信息，请参阅指定要导出到的 Amazon S3 文件路径。
2. 提供权限以访问 Amazon S3 存储桶。

   要将数据导出到 Amazon S3 文件，请为 Aurora PostgreSQL 数据库集群提供权限以访问导出将用于存储的 Amazon S3 存储桶。此操作包括以下步骤：

   1. 创建一个 IAM 策略，该策略提供对要导出到的 Amazon S3 存储桶的访问权限。
   2. 创建一个 IAM 角色。
   3. 将您创建的策略附在您创建的角色上。
   4. 将此 IAM 角色添加到数据库集群。

   有关此过程的详细信息，请参阅设置 Amazon S3 存储桶的访问权限。
3. 标识数据库查询以获取数据。通过调用 `aws_s3.query_export_to_s3` 函数来导出查询数据。

   在完成上述准备任务后，请使用 aws_s3.query_export_to_s3 函数将查询结果导出到 Amazon S3。有关此过程的详细信息，请参阅使用 aws_s3.query_export_to_s3 函数导出查询数据。

指定要导出到的 Amazon S3 文件路径

指定以下信息以标识要将数据导出到的 Amazon S3 中的位置：

- 存储桶名称 – 存储桶是 Amazon S3 对象或文件的容器。

  有关使用 Amazon S3 存储数据的更多信息，请参阅《Amazon Simple Storage Service 用户指南》中的创建存储桶和使用对象。
- 文件路径 – 文件路径标识 Amazon S3 存储桶中存储导出的位置。文件路径包含以下内容：
  - 用于标识虚拟文件夹路径的可选路径前缀。
  - 用于标识要存储的一个或多个文件的文件前缀。较大的导出将存储在多个文件中，每个文件的最大大小约为 6 GB。其他文件名具有相同的文件前缀，但追加了 `_part`*XX*。*XX* 表示 2，然后表示 3，依此类推。

  例如，具有 `exports` 文件夹和 `query-1-export` 文件前缀的文件路径为 `/exports/query-1-export`。

- AWS区域（可选）– Amazon S3 存储桶所在的AWS区域。如果您未指定AWS区域值，则 Aurora 会将文件保存到导出的数据库集群所在的同一AWS区域中的 Amazon S3 中。

  > **Note**
  >
  > 目前，AWS 区域必须与导出的数据库集群的区域相同。

  有关 AWS 区域名称和关联值的列表，请参阅区域及可用区。

要保存有关导出的存储位置的 Amazon S3 文件信息，您可以使用 aws_commons.create_s3_uri 函数创建一个 `aws_commons._s3_uri_1` 复合结构，如下所示。

```
psql=> SELECT aws_commons.create_s3_uri(
   'amzn-s3-demo-bucket',
   'sample-filepath',
   'us-west-2'
) AS s3_uri_1 \gset
```

稍后，您可以在对 `s3_uri_1` 函数的调用中将该 aws_s3.query_export_to_s3 值作为参数提供。有关示例，请参阅 使用 aws_s3.query_export_to_s3 函数导出查询数据。

设置 Amazon S3 存储桶的访问权限

要将数据导出到 Amazon S3，请为 PostgreSQL 数据库集群提供权限以访问文件将保存到的 Amazon S3 存储桶。

为此，请使用以下过程。

通过 IAM 角色向 PostgreSQL 数据库集群授予访问 Amazon S3 的权限

1. 创建一个 IAM 策略。

   该策略提供存储桶和对象权限，以允许 PostgreSQL 数据库集群访问 Amazon S3。

   在创建此策略的过程中，请执行以下步骤：

   a. 在策略中包含以下必需操作，以允许将文件从 PostgreSQL 数据库集群传输到 Amazon S3 存储桶：

      - `s3:PutObject`

- `s3:AbortMultipartUpload`

b. 包含用于标识 Amazon S3 存储桶以及其中的对象的 Amazon Resource Name (ARN)。用于访问 Amazon S3 的 ARN 格式为：`arn:aws:s3:::amzn-s3-demo-bucket/*`

有关为 Aurora PostgreSQL 创建 IAM 策略的更多信息，请参阅 创建和使用适用于 IAM 数据库访问的 IAM 策略。另请参阅 IAM 用户指南中的教程：创建和附加您的第一个客户托管式策略。

以下 AWS CLI 命令使用这些选项创建一个名为 `rds-s3-export-policy` 的 IAM 策略。该策略授予对名为 *amzn-s3-demo-bucket* 的存储桶的访问权限。

> **Warning**
>
> 我们建议您在一个私有 VPC 中设置数据库，该 VPC 配置了用于访问特定存储桶的端点策略。有关更多信息，请参阅 Amazon VPC 用户指南 中的对 Amazon S3 使用端点策略。强烈建议您不要创建具有所有资源访问权限的策略。此访问权限可能会对数据安全造成威胁。如果您使用 `S3:PutObject` 创建一个向 `"Resource":"*"` 授予对所有资源的访问权限的策略，则具有导出特权的用户可以将数据导出到您账户中的所有存储桶。此外，用户可以将数据导出到 AWS 区域内的任何可公开写入的存储桶。

在您创建策略之后，请记下策略的 Amazon Resource Name (ARN)。在将策略附加到 IAM 角色时，您在后面的步骤中需要使用 ARN。

```
aws iam create-policy  --policy-name rds-s3-export-policy  --policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Sid": "s3export",
         "Action": [
           "s3:PutObject*",
           "s3:ListBucket",
           "s3:GetObject*",
           "s3:DeleteObject*",
           "s3:GetBucketLocation",
           "s3:AbortMultipartUpload"
         ],
         "Effect": "Allow",
         "Resource": [
           "arn:aws:s3:::amzn-s3-demo-bucket/*"
```

```
        ]
      }
    ]
  }'
```

2. 创建一个 IAM 角色。

   这样，Aurora PostgreSQL 就可以担任该 IAM 角色以代表您访问 Amazon S3 存储桶。有关更多信息，请参阅 IAM 用户指南中的创建向 IAM 用户委派权限的角色。

   我们建议在基于资源的策略中使用 aws:SourceArn 和 aws:SourceAccount 全局条件上下文键，以限制对特定资源的服务权限。这是防范混淆代理人问题最有效的方法。

   如果同时使用全局条件上下文键和包含账户 ID 的 aws:SourceArn 值，则 aws:SourceAccount 值和 aws:SourceArn 值中的账户在同一策略语句中使用时，必须使用相同的账户 ID。

   - 如果您想对单个资源进行跨服务访问，请使用 aws:SourceArn。
   - 如果您想允许该账户中的任何资源与跨服务使用操作相关联，请使用 aws:SourceAccount。

   在策略中，确保使用具有资源的完整 ARN 的 aws:SourceArn 全局条件上下文键。以下示例说明了如何使用 AWS CLI 命令创建一个名为 rds-s3-export-role 的角色来实现该目的。

   Example

   对于 Linux、macOS 或 Unix：

```
aws iam create-role  \
    --role-name rds-s3-export-role  \
    --assume-role-policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Principal": {
            "Service": "rds.amazonaws.com"
          },
         "Action": "sts:AssumeRole",
         "Condition": {
             "StringEquals": {
                "aws:SourceAccount": "111122223333",
```

```
                "aws:SourceArn": "arn:aws:rds:us-
east-1:111122223333:cluster:dbname"
                }
            }
        }
      ]
   }'
```

对于 Windows：

```
aws iam create-role  ^
    --role-name rds-s3-export-role  ^
    --assume-role-policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Principal": {
            "Service": "rds.amazonaws.com"
          },
         "Action": "sts:AssumeRole",
         "Condition": {
             "StringEquals": {
                "aws:SourceAccount": "111122223333",
                "aws:SourceArn": "arn:aws:rds:us-
east-1:111122223333:cluster:dbname"
                }
            }
        }
      ]
   }'
```

3. 将您创建的 IAM 策略附加到您创建的 IAM 角色。

   以下 AWS CLI 命令将以前创建的策略附加到一个名为 `rds-s3-export-role.` 的角色。请将 *your-policy-arn* 替换为您在前面的步骤中记下的策略 ARN。

   ```
   aws iam attach-role-policy  --policy-arn your-policy-arn  --role-name rds-s3-
   export-role
   ```

4. 将该 IAM 角色添加到数据库集群中。您可以使用 AWS Management Console 或 AWS CLI 执行该操作，如下所述。

控制台

使用控制台为 PostgreSQL 数据库集群添加 IAM 角色

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 选择 PostgreSQL 数据库集群名称以显示其详细信息。
3. 在 Connectivity & security (连接性和安全性) 选项卡上的 Manage IAM roles (管理 IAM 角色) 部分中，在 Add IAM roles to this instance (向此实例添加 IAM 角色) 下选择要添加的角色。
4. 在 Feature (功能) 下，选择 s3Export。
5. 选择 Add role (添加角色)。

AWS CLI

使用 CLI 为 PostgreSQL 数据库实例添加 IAM 角色

- 使用以下命令将角色添加到名为 `my-db-cluster` 的 PostgreSQL 数据库集群中。将 *`your-role-arn`* 替换为您在上一步中记下的角色 ARN。使用 `s3Export` 作为 `--feature-name` 选项的值。

  Example

  对于 Linux、macOS 或 Unix：

  ```
  aws rds add-role-to-db-cluster \
     --db-cluster-identifier my-db-cluster \
     --feature-name s3Export \
     --role-arn your-role-arn   \
     --region your-region
  ```

  对于 Windows：

  ```
  aws rds add-role-to-db-cluster ^
     --db-cluster-identifier my-db-cluster ^
     --feature-name s3Export ^
     --role-arn your-role-arn ^
     --region your-region
  ```

使用 aws_s3.query_export_to_s3 函数导出查询数据

通过调用 aws_s3.query_export_to_s3 函数将 PostgreSQL 数据导出到 Amazon S3。

主题

- 先决条件
- 调用 aws_s3.query_export_to_s3
- 导出到使用自定义分隔符的 CSV 文件
- 导出到具有编码的二进制文件

先决条件

在使用 aws_s3.query_export_to_s3 函数之前，请确保完成以下先决条件：

- 安装所需的 PostgreSQL 扩展，如将数据导出到 Amazon S3 概述中所述。
- 确定要将数据导出到 Amazon S3 的位置，如指定要导出到的 Amazon S3 文件路径中所述。
- 确保数据库集群具有 设置 Amazon S3 存储桶的访问权限 中所述的对 Amazon S3 的访问权限。

以下示例使用一个称作 sample_table 的数据库表。这些示例将数据导出到称为 *amzn-s3-demo-bucket* 的存储桶中。使用以下 SQL 语句在 psql 中创建示例表和数据。

```
psql=> CREATE TABLE sample_table (bid bigint PRIMARY KEY, name varchar(80));
psql=> INSERT INTO sample_table (bid,name) VALUES (1, 'Monday'), (2,'Tuesday'), (3,
 'Wednesday');
```

调用 aws_s3.query_export_to_s3

下面说明了调用 aws_s3.query_export_to_s3 函数的基本方法。

这些示例使用变量 s3_uri_1 确定包含用于标识 Amazon S3 文件的信息的结构。使用 aws_commons.create_s3_uri 函数可创建结构。

```
psql=> SELECT aws_commons.create_s3_uri(
   'amzn-s3-demo-bucket',
   'sample-filepath',
   'us-west-2'
) AS s3_uri_1 \gset
```

尽管以下两个 aws_s3.query_export_to_s3 函数调用的参数不同，但这些示例的结果是相同的。sample_table 表的所有行都将导出到称为 *amzn-s3-demo-bucket* 的存储桶中。

```
psql=> SELECT * FROM aws_s3.query_export_to_s3('SELECT * FROM
 sample_table', :'s3_uri_1');

psql=> SELECT * FROM aws_s3.query_export_to_s3('SELECT * FROM
 sample_table', :'s3_uri_1', options :='format text');
```

参数如下所述：

- 'SELECT * FROM sample_table' – 第一个参数是包含 SQL 查询的必需文本字符串。PostgreSQL 引擎将运行此查询。查询的结果将复制到其他参数中标识的 S3 存储桶。
- :'s3_uri_1' – 此参数是标识 Amazon S3 文件的结构。此示例使用变量来标识先前创建的结构。相反，您可通过内联方式在 aws_commons.create_s3_uri 函数调用中包含 aws_s3.query_export_to_s3 函数调用来创建结构，如下所示。

  ```
  SELECT * from aws_s3.query_export_to_s3('select * from sample_table',
     aws_commons.create_s3_uri('amzn-s3-demo-bucket', 'sample-filepath', 'us-west-2')
  );
  ```

- options :='format text' – options 参数是包含 PostgreSQL COPY 参数的可选文本字符串。复制过程使用 PostgreSQL COPY 命令的参数和格式。

如果指定的文件在 Amazon S3 存储桶中不存在，则会创建该文件。如果该文件已存在，则将覆盖该文件。以下是用于访问 Amazon S3 中的导出数据的语法。

```
s3-region://bucket-name[/path-prefix]/file-prefix
```

较大的导出将存储在多个文件中，每个文件的最大大小约为 6 GB。其他文件名具有相同的文件前缀，但追加了 _part*XX*。*XX* 表示 2，然后表示 3，依此类推。例如，假设您指定用于存储数据文件的路径，如下所示。

```
s3-us-west-2://amzn-s3-demo-bucket/my-prefix
```

如果导出必须创建三个数据文件，则 Amazon S3 存储桶将包含以下数据文件。

```
s3-us-west-2://amzn-s3-demo-bucket/my-prefix
```

```
s3-us-west-2://amzn-s3-demo-bucket/my-prefix_part2
s3-us-west-2://amzn-s3-demo-bucket/my-prefix_part3
```

有关此函数的完整参考以及其他调用方法，请参阅aws_s3.query_export_to_s3。有关访问 Amazon S3 中的文件的更多信息，请参阅 Amazon Simple Storage Service 用户指南中的查看对象。

导出到使用自定义分隔符的 CSV 文件

以下示例说明如何调用 aws_s3.query_export_to_s3 函数以将数据导出到使用自定义分隔符的文件。此示例使用 PostgreSQL COPY 命令的参数来指定逗号分隔值 (CSV) 格式和冒号 (:) 分隔符。

```
SELECT * from aws_s3.query_export_to_s3('select * from basic_test', :'s3_uri_1',
 options :='format csv, delimiter $$:$$');
```

导出到具有编码的二进制文件

以下示例说明如何调用 aws_s3.query_export_to_s3 函数以将数据导出到具有 Windows-1253 编码的二进制文件。

```
SELECT * from aws_s3.query_export_to_s3('select * from basic_test', :'s3_uri_1',
 options :='format binary, encoding WIN1253');
```

函数参考

函数

- aws_s3.query_export_to_s3
- aws_commons.create_s3_uri

aws_s3.query_export_to_s3

将 PostgreSQL 查询结果导出到 Amazon S3 存储桶。aws_s3 扩展提供 aws_s3.query_export_to_s3 函数。

两个必需的参数为 query 和 s3_info。这两个参数定义了要导出的查询，并标识了要导出到的 Amazon S3 存储桶。一个称作 options 的可选参数，用于定义各种导出参数。有关使用 aws_s3.query_export_to_s3 函数的示例，请参阅使用 aws_s3.query_export_to_s3 函数导出查询数据。

语法

```
aws_s3.query_export_to_s3(
    query text,
    s3_info aws_commons._s3_uri_1,
    options text,
    kms_key text
)
```

输入参数

query

一个必需的文本字符串，其中包含 PostgreSQL 引擎运行的 SQL 查询。此查询的结果将复制到 `s3_info` 参数中标识的 S3 存储桶。

s3_info

包含有关 S3 对象的以下信息的 `aws_commons._s3_uri_1` 复合类型：

- `bucket` – 要包含文件的 Amazon S3 存储桶的名称。
- `file_path` – Amazon S3 文件名和路径。
- `region` – 存储桶所在的AWS区域。有关 AWS 区域名称和关联值的列表，请参阅区域及可用区。

  目前，此值必须是导出的数据库集群的同一个 AWS 区域。默认值为导出的数据库集群的 AWS 区域。

要创建 `aws_commons._s3_uri_1` 复合结构，请参阅 aws_commons.create_s3_uri 函数。

options

一个包含 PostgreSQL `COPY` 命令的参数的可选文本字符串。这些参数指定了在导出时复制数据的方式。有关更多详细信息，请参阅 PostgreSQL COPY 文档。

kms_key 文本

一个可选文本字符串，其中包含要将数据导出到的 S3 存储桶的客户托管 KMS 密钥。

备用输入参数

为帮助进行测试，您可以使用一组扩展的参数而非 `s3_info` 参数。下面是 `aws_s3.query_export_to_s3` 函数的其他语法变化。

不使用 s3_info 参数来标识 Amazon S3 文件，而使用 bucket、file_path 和 region 参数的组合。

```
aws_s3.query_export_to_s3(
    query text,
    bucket text,
    file_path text,
    region text,
    options text,
    kms_key text
)
```

query

一个必需的文本字符串，其中包含 PostgreSQL 引擎运行的 SQL 查询。此查询的结果将复制到 s3_info 参数中标识的 S3 存储桶。

bucket

一个包含 Amazon S3 存储桶（其中包含文件）的名称的必需文本字符串。

file_path

包含 Amazon S3 文件名（包含文件路径）的必填文本字符串。

region

一个包含存储桶所在的 AWS 区域的可选文本字符串。有关 AWS 区域名称和关联值的列表，请参阅区域及可用区。

目前，此值必须是导出的数据库集群的同一个 AWS 区域。默认值为导出的数据库集群的 AWS 区域。

options

一个包含 PostgreSQL COPY 命令的参数的可选文本字符串。这些参数指定了在导出时复制数据的方式。有关更多详细信息，请参阅 PostgreSQL COPY 文档。

kms_key 文本

一个可选文本字符串，其中包含要将数据导出到的 S3 存储桶的客户托管 KMS 密钥。

输出参数

```
aws_s3.query_export_to_s3(
```

```
    OUT rows_uploaded bigint,
    OUT files_uploaded bigint,
    OUT bytes_uploaded bigint
)
```

rows_uploaded

针对给定查询成功上传到 Amazon S3 的表行的数目。

files_uploaded

已上传到 Amazon S3 的文件的数目。以约 6 GB 的大小创建文件。创建的每个附加文件的名称都追加了 _part*XX*。根据需要，*XX* 表示 2，然后表示 3，依此类推。

bytes_uploaded

已上传到 Amazon S3 的字节的总数。

示例

```
psql=> SELECT * from aws_s3.query_export_to_s3('select * from sample_table', 'amzn-s3-
demo-bucket', 'sample-filepath');
psql=> SELECT * from aws_s3.query_export_to_s3('select * from sample_table', 'amzn-s3-
demo-bucket', 'sample-filepath','us-west-2');
psql=> SELECT * from aws_s3.query_export_to_s3('select * from sample_table', 'amzn-s3-
demo-bucket', 'sample-filepath','us-west-2','format text');
```

aws_commons.create_s3_uri

创建 aws_commons._s3_uri_1 结构来保存 Amazon S3 文件信息。在 aws_commons.create_s3_uri 函数的 s3_info 参数中使用 aws_s3.query_export_to_s3 函数的结果。有关使用 aws_commons.create_s3_uri 函数的示例，请参阅指定要导出到的 Amazon S3 文件路径。

语法

```
aws_commons.create_s3_uri(
   bucket text,
   file_path text,
   region text
)
```

输入参数

bucket

包含文件的 Amazon S3 存储桶名称的必需文本字符串。

file_path

包含 Amazon S3 文件名（包含文件路径）的必填文本字符串。

region

一个包含文件所在的 AWS 区域的必需文本字符串。有关 AWS 区域名称和关联值的列表，请参阅区域及可用区。

Amazon S3 访问故障排除

如果在尝试将数据导出到 Amazon S3 时遇到连接问题，请首先确认与数据库实例关联的 VPC 安全组的出站访问规则是否允许网络连接。具体而言，安全组必须有一条规则允许数据库实例将 TCP 流量发送到端口 443 和任何 IPv4 地址（0.0.0/0）。有关更多信息，请参阅 通过创建安全组提供对 VPC 中数据库集群的访问。

另请参阅以下建议：

- 对 Amazon Aurora 身份和访问权限问题进行故障排除
- Amazon Simple Storage Service 用户指南中的排查 Amazon S3 的问题
- IAM 用户指南中的排查 Amazon S3 和 IAM 的问题

从 Aurora PostgreSQL 数据库集群中调用 AWS Lambda 函数

AWS Lambda 是事件驱动型计算服务，无需您预置或管理服务器即可运行代码。该服务可与许多 AWS 服务搭配使用，其中包括 Aurora PostgreSQL。例如，您可以使用 Lambda 函数处理来自数据库的事件通知，或者在将新文件上传到 Amazon S3 时从文件中加载数据。若要了解 Lambda 的更多信息，请参阅《AWS Lambda 开发人员指南》中的什么是 AWS Lambda？

> **Note**
>
> Aurora PostgreSQL 11.9 及更高版本（包括 Aurora Serverless v2）支持调用 AWS Lambda 函数。

设置 Aurora PostgreSQL 使用 Lambda 函数的过程包含多个步骤，其中涉及 AWS Lambda、IAM、VPC 和 Aurora PostgreSQL 数据库集群。下文对必要步骤进行了总结。

有关 Lambda 函数的更多信息，请参阅《AWS 开发人员指南https://docs.aws.amazon.com/lambda/latest/dg/getting-started.html》中的 Lambda 入门和 AWS Lambda Lambda 函数。

主题

- 步骤 1：配置 Aurora PostgreSQL 数据库集群，实现与 AWS Lambda 的出站连接
- 步骤 2：为 Aurora PostgreSQL 数据库集群和 AWS Lambda 配置 IAM
- 步骤 3：为 Aurora PostgreSQL 数据库集群安装 aws_lambda 扩展
- 步骤 4：将 Lambda 帮助程序函数与 Aurora PostgreSQL 数据库集群搭配使用（可选）
- 步骤 5：从 Aurora PostgreSQL 数据库集群调用 Lambda 函数
- 步骤 6：授予其他用户调用 Lambda 函数的权限
- 示例：从 Aurora PostgreSQL 数据库集群调用 Lambda 函数
- Lambda 函数错误消息
- AWS Lambda 函数和参数参考

步骤 1：配置 Aurora PostgreSQL 数据库集群，实现与 AWS Lambda 的出站连接

Lambda 函数始终在 AWS Lambda 服务拥有的 Amazon VPC 中运行。Lambda 将向此 VPC 应用网络访问和安全规则，并且会自动维护和监控 VPC。Aurora PostgreSQL 数据库集群向 Lambda 服务的 VPC 发送网络流量。其配置方式取决于 Aurora 数据库集群的主数据库实例是公有实例，还是私有实例。

- 公有 Aurora PostgreSQL 数据库集群 – 如果数据库集群的主数据库实例位于 VPC 的公有子网中，并且该实例的“PublicyAccessible”属性为 `true`，则该实例是公有的。若要查找此属性的值，您可以使用 describe-db-instances AWS CLI 命令。您也可以使用 AWS Management Console 打开 Connectivity & security（连接和安全性）选项卡，然后检查 Publicly accessible（公开访问）是否为 Yes（是）。要验证实例是否在您的 VPC 的公有子网中，您可以使用 AWS Management Console或 AWS CLI。

  要设置对 Lambda 的访问权限，您可以使用 AWS Management Console 或 AWS CLI 在 VPC 的安全组上创建出站规则。出站规则指定 TCP 可以使用端口 443 将数据包发送到任何 IPv4 地址（0.0.0.0/0）。
- 私有 Aurora PostgreSQL 数据库集群 – 在这种情况下，实例的“PublicyAccessible”属性为 `false` 或它位于私有子网中。要允许实例使用 Lambda，您可以使用网络地址转换（NAT）网关。有关更多

信息，请参阅 NAT 网关。或者，您可以使用用于 Lambda 的 VPC 端点配置 VPC。有关更多信息，请参阅《Amazon VPC 用户指南》中的 VPC 端点。该端点响应 Aurora PostgreSQL 数据库集群对 Lambda 函数发出的调用。

您的 VPC 现在可以在网络级别与 AWS Lambda VPC 交互。接下来，您使用 IAM 配置权限。

步骤 2：为 Aurora PostgreSQL 数据库集群和 AWS Lambda 配置 IAM

从 Aurora PostgreSQL 数据库集群调用 Lambda 函数需要特定权限。若要配置必要权限，建议创建允许调用 Lambda 函数的 IAM 策略，将该策略分配给一个角色，然后将该角色应用于数据库集群。这种方法授予数据库集群代表您调用指定 Lambda 函数的权限。以下步骤说明如何使用 AWS CLI 执行此操作。

配置 IAM 权限以将集群与 Lambda 搭配使用

1. 使用 create-policy AWS CLI 命令创建允许 Aurora PostgreSQL 数据库集群调用指定 Lambda 函数的 IAM 策略。（语句 ID (Sid) 是策略语句的可选描述，对使用没有影响。）此策略授予 Aurora 数据库集群调用指定 Lambda 函数所需的最低权限。

```
aws iam create-policy  --policy-name rds-lambda-policy --policy-document '{
    "Version": "2012-10-17",
    "Statement": [
        {
        "Sid": "AllowAccessToExampleFunction",
        "Effect": "Allow",
        "Action": "lambda:InvokeFunction",
        "Resource": "arn:aws:lambda:aws-region:444455556666:function:my-function"
        }
    ]
}'
```

   您也可以使用允许您调用任何 Lambda 函数的预定义 AWSLambdaRole 策略。有关更多信息，请参阅 Lambda 的基于身份的 IAM 策略

2. 使用 create-role AWS CLI 命令创建该策略可在运行时担任的 IAM 角色。

```
aws iam create-role  --role-name rds-lambda-role --assume-role-policy-document '{
    "Version": "2012-10-17",
    "Statement": [
        {
        "Effect": "Allow",
```

```
        "Principal": {
            "Service": "rds.amazonaws.com"
        },
        "Action": "sts:AssumeRole"
        }
    ]
}'
```

3. 使用 attach-role-policy AWS CLI 命令将策略应用于角色。

```
aws iam attach-role-policy \
    --policy-arn arn:aws:iam::444455556666:policy/rds-lambda-policy \
    --role-name rds-lambda-role --region aws-region
```

4. 使用 add-role-to-db-cluster AWS CLI 命令，将该角色应用于 Aurora PostgreSQL 数据库集群。最后一步允许数据库集群的数据库用户调用 Lambda 函数。

```
aws rds add-role-to-db-cluster \
       --db-cluster-identifier my-cluster-name \
       --feature-name Lambda \
       --role-arn  arn:aws:iam::444455556666:role/rds-lambda-role   \
       --region aws-region
```

完成 VPC 和 IAM 配置后，便可以安装 aws_lambda 扩展。（请注意，您可以随时安装扩展，但只有在您设置正确的 VPC 支持和 IAM 权限之后，aws_lambda 扩展才会对 Aurora PostgreSQL 数据库集群的功能添加内容。）

步骤 3：为 Aurora PostgreSQL 数据库集群安装 **aws_lambda** 扩展

要将 AWS Lambda 与 Aurora PostgreSQL 数据库集群结合使用，请将 aws_lambda PostgreSQL 扩展添加到 Aurora PostgreSQL 数据库集群。此扩展允许 Aurora PostgreSQL 数据库集群能从 PostgreSQL 调用 Lambda 函数。

为 Aurora PostgreSQL 数据库集群安装 **aws_lambda** 扩展

使用 PostgreSQL psql 命令行或 pgAdmin 工具连接到 Aurora PostgreSQL 数据库集群。

1. 以具有 rds_superuser 权限的用户身份，连接到 Aurora PostgreSQL 数据库集群。默认 postgres 用户如示例所示。

```
psql -h cluster-instance.444455556666.aws-region.rds.amazonaws.com -U postgres -p
 5432
```

2. 安装 aws_lambda 扩展。aws_commons 扩展也是必要项。其为 PostgreSQL 的 aws_lambda 和许多其他 Aurora 扩展提供帮助程序函数。如果其尚未安装到 Aurora PostgreSQL 数据库集群上，则会按如下所示一并安装该扩展和 aws_lambda。

```
CREATE EXTENSION IF NOT EXISTS aws_lambda CASCADE;
NOTICE:  installing required extension "aws_commons"
CREATE EXTENSION
```

aws_lambda 扩展已安装在 Aurora PostgreSQL 数据库集群的主数据库实例中。现在，您可以创建易于使用结构来调用 Lambda 函数。

步骤 4：将 Lambda 帮助程序函数与 Aurora PostgreSQL 数据库集群搭配使用（可选）

您可以使用 aws_commons 扩展中的帮助程序函数来准备实体，以便更轻松地从 PostgreSQL 调用这些实体。为此，您需要获得有关 Lambda 函数的以下信息：

- Function name（函数名称）：Lambda 函数的名称、Amazon Resource Name (ARN)、版本或别名。在 步骤 2：为集群和 Lambda 配置 IAM 中创建的 IAM 策略需要 ARN，所以建议您使用函数的 ARN。
- AWS 区域 —（可选）如果与 Aurora PostgreSQL 数据库集群 不在同一个区域中，则此区域为 Lambda 函数所在的 AWS 区域。

您可以使用 aws_commons.create_lambda_function_arn 函数保存 Lambda 函数名称信息。此帮助程序函数创建了 aws_commons._lambda_function_arn_1 复合结构，其中包含调用函数所需的详细信息。在下文中，您可以找到三种替代方法来设置此复合结构。

```
SELECT aws_commons.create_lambda_function_arn(
   'my-function',
   'aws-region'
) AS aws_lambda_arn_1 \gset
```

```
SELECT aws_commons.create_lambda_function_arn(
   '111122223333:function:my-function',
   'aws-region'
```

```
) AS lambda_partial_arn_1 \gset
```

```
SELECT aws_commons.create_lambda_function_arn(
   'arn:aws:lambda:aws-region:111122223333:function:my-function'
) AS lambda_arn_1 \gset
```

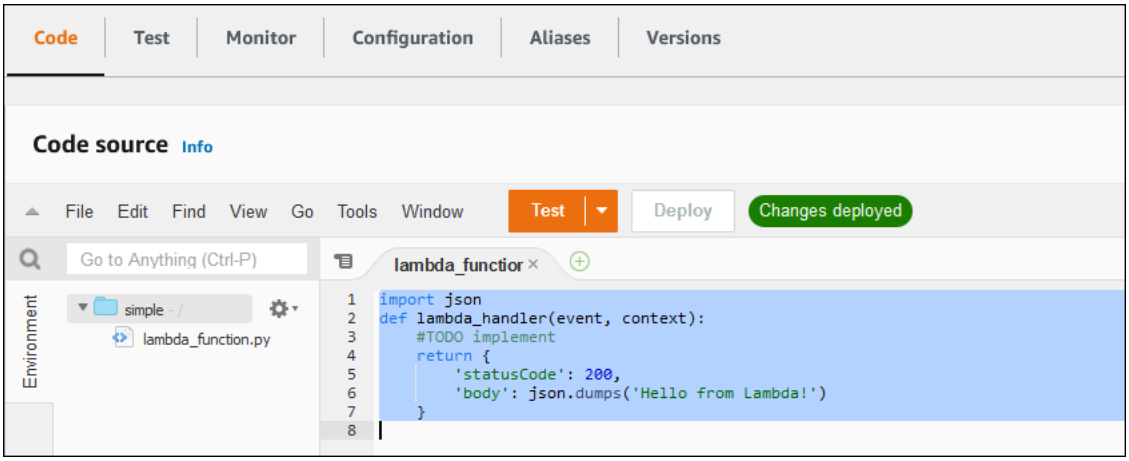
这些值均可用于 aws_lambda.invoke 函数调用。有关示例，请参阅 步骤 5：从 Aurora PostgreSQL 数据库集群调用 Lambda 函数。

步骤 5：从 Aurora PostgreSQL 数据库集群调用 Lambda 函数

aws_lambda.invoke 函数采用同步还是异步调用方式取决于 invocation_type。此参数的两个可用选项分别为 RequestResponse（默认）和 Event，如下所示。

- **RequestResponse**：此为同步调用类型。如果调用时未指定调用类型，默认此调用方式。响应有效负载包含 aws_lambda.invoke 函数的结果。如果工作流程需要接收 Lambda 函数的结果才能继续进行操作，请使用此调用类型。
- **Event**：此为异步调用类型。响应不包含包含结果的有效负载。如果工作流程不需要 Lambda 函数的结果即可继续进行操作，请使用此调用类型。

要对设置进行简单测试，您可以使用 psql 连接到数据库实例，然后从命令行调用示例函数。假设在 Lambda 服务上设置了一个基本函数，例如下面屏幕截图中所示的简单 Python 函数。



调用示例函数

1. 使用 psql 或 pgAdmin 连接到主数据库实例。

   ```
   psql -h cluster.444455556666.aws-region.rds.amazonaws.com -U postgres -p 5432
   ```

2. 使用函数的 ARN 调用该函数。

```
SELECT * from
 aws_lambda.invoke(aws_commons.create_lambda_function_arn('arn:aws:lambda:aws-
region:444455556666:function:simple', 'us-west-1'), '{"body": "Hello from
 Postgres!"}'::json );
```

响应如下所示。

```
status_code |                          payload                          |
 executed_version | log_result
-------------+-----------------------------------------------------------
+------------------+------------
         200 | {"statusCode": 200, "body": "\"Hello from Lambda!\""} | $LATEST
     |
(1 row)
```

如果调用尝试不成功，请参阅 Lambda 函数错误消息 。

步骤 6：授予其他用户调用 Lambda 函数的权限

在程序的这一步骤中，仅当您是 rds_superuser 才能调用 Lambda 函数。要允许其他用户调用您创建的任何函数，您需要向他们授予权限。

要授予调用 Lambda 函数的其他权限，请执行以下操作：

1. 使用 psql 或 pgAdmin 连接到主数据库实例。

```
psql -h cluster.444455556666.aws-region.rds.amazonaws.com -U postgres -p 5432
```

2. 运行以下 SQL 命令：

```
postgres=>  GRANT USAGE ON SCHEMA aws_lambda TO db_username;
GRANT EXECUTE ON ALL FUNCTIONS IN SCHEMA aws_lambda TO db_username;
```

示例：从 Aurora PostgreSQL 数据库集群调用 Lambda 函数

在下文中，您可以找到调用 aws_lambda.invoke 函数的一些示例。大多数示例都使用您在 步骤 4：将 Lambda 帮助程序函数与 Aurora PostgreSQL 数据库集群搭配使用（可选） 中创建的复合结构

aws_lambda_arn_1 来简化函数详细信息的传递。有关异步调用的示例，请参阅 示例：Lambda 函数的异步（事件）调用。列出的其他示例均使用同步调用。

要了解有关 Lambda 调用类型的更多信息，请参阅《AWS Lambda 开发人员指南》中的调用 Lambda 函数。有关 aws_lambda_arn_1 的更多信息，请参阅 aws_commons.create_lambda_function_arn。

示例列表

- 示例：Lambda 函数的同步 (RequestResponse) 调用
- 示例：Lambda 函数的异步（事件）调用
- 示例：在函数响应中捕获 Lambda 执行日志
- 示例：在 Lambda 函数中包含客户端上下文
- 示例：调用 Lambda 函数的特定版本

示例：Lambda 函数的同步 (RequestResponse) 调用

以下是 Lambda 函数同步调用的两个示例。这些 aws_lambda.invoke 函数调用的结果相同。

```
SELECT * FROM aws_lambda.invoke('aws_lambda_arn_1', '{"body": "Hello from
 Postgres!"}'::json);
```

```
SELECT * FROM aws_lambda.invoke('aws_lambda_arn_1', '{"body": "Hello from
 Postgres!"}'::json, 'RequestResponse');
```

参数如下所述：

- :'aws_lambda_arn_1'：此参数使用 aws_commons.create_lambda_function_arn 帮助程序函数标识在 步骤 4：将 Lambda 帮助程序函数与 Aurora PostgreSQL 数据库集群搭配使用（可选）中创建的复合结构。您还可以通过内联方式在 aws_lambda.invoke 调用中创建此结构，如下所示。

  ```
  SELECT * FROM aws_lambda.invoke(aws_commons.create_lambda_function_arn('my-function',
   'aws-region'),
  '{"body": "Hello from Postgres!"}'::json
  );
  ```

- '{"body": "Hello from PostgreSQL!"}'::json – 要传递到 Lambda 函数的 JSON 负载。
- 'RequestResponse' – Lambda 调用类型。

示例：Lambda 函数的异步（事件）调用

以下是异步 Lambda 函数调用的示例。Event 调用类型使用指定的输入负载计划 Lambda 函数调用并立即返回。在某些不依赖于 Lambda 函数结果的工作流程中使用 Event 调用类型。

```
SELECT * FROM aws_lambda.invoke('aws_lambda_arn_1', '{"body": "Hello from
 Postgres!"}'::json, 'Event');
```

示例：在函数响应中捕获 Lambda 执行日志

您可以使用 aws_lambda.invoke 函数调用中的 log_type 参数，在函数响应中包含执行日志的最后 4 kB。默认情况下，此参数设置为 None，但您可以指定 Tail 在响应中捕获 Lambda 执行日志的结果，如下所示。

```
SELECT *, select convert_from(decode(log_result, 'base64'), 'utf-8') as log FROM
 aws_lambda.invoke(:'aws_lambda_arn_1', '{"body": "Hello from Postgres!"}'::json,
 'RequestResponse', 'Tail');
```

将 aws_lambda.invoke 函数的 log_type 参数设置为 Tail，以在响应中包含执行日志。log_type 参数的默认值为 None。

返回的 log_result 是 base64 编码的字符串。您可以使用 decode 和 convert_from PostgreSQL 函数的组合来解码内容。

有关 log_type 的更多信息，请参阅 aws_lambda.invoke。

示例：在 Lambda 函数中包含客户端上下文

aws_lambda.invoke 函数具有 context 参数，可用于传递独立于有效负载的信息，如下所示。

```
SELECT *, convert_from(decode(log_result, 'base64'), 'utf-8') as log FROM
 aws_lambda.invoke(:'aws_lambda_arn_1', '{"body": "Hello from Postgres!"}'::json,
 'RequestResponse', 'Tail');
```

要包含客户端上下文，请将 JSON 对象用于 aws_lambda.invoke 函数的 context 参数。

有关 context 参数的更多信息，请参阅 aws_lambda.invoke 参考。

示例：调用 Lambda 函数的特定版本

通过在 aws_lambda.invoke 调用中包含 qualifier 参数，您可以指定 Lambda 函数的特定版本。在下文中，您可以找到一个使用 '*custom_version*' 作为版本别名完成此操作的示例。

```
SELECT * FROM aws_lambda.invoke('aws_lambda_arn_1', '{"body": "Hello from
 Postgres!"}'::json, 'RequestResponse', 'None', NULL, 'custom_version');
```

您还可以改为提供包含 Lambda 函数名称详细信息的 Lambda 函数限定符，如下所示。

```
SELECT * FROM aws_lambda.invoke(aws_commons.create_lambda_function_arn('my-
function:custom_version', 'us-west-2'),
'{"body": "Hello from Postgres!"}'::json);
```

有关 `qualifier` 和其他参数的详细信息，请参阅 aws_lambda.invoke 参考。

Lambda 函数错误消息

在下面的列表中，您可以找到有关错误消息的信息，以及可能的原因和解决方案。

- VPC 配置问题

  在尝试连接时，VPC 配置问题可能会引发以下错误消息：

  ```
  ERROR:  invoke API failed
  DETAIL: AWS Lambda client returned 'Unable to connect to endpoint'.
  CONTEXT:  SQL function "invoke" statement 1
  ```

  导致此错误的常见原因是 VPC 安全组配置不当。确保在 VPC 安全组的端口 443 上打开 TCP 的出站规则，以便 VPC 能够连接到 Lambda VPC。

- 缺乏调用 Lambda 函数所需的权限

  如果看到以下任一错误消息，说明调用此函数的用户（角色）没有适当的权限。

  ```
  ERROR:  permission denied for schema aws_lambda
  ```

  ```
  ERROR:  permission denied for function invoke
  ```

  必须向用户（角色）授予特定权限才能调用 Lambda 函数。有关更多信息，请参阅 步骤 6：授予其他用户调用 Lambda 函数的权限。

- 对 Lambda 函数中的错误处理不当

  如果 Lambda 函数在请求处理过程中抛出异常，则 `aws_lambda.invoke` 会失败并显示 PostgreSQL 错误，如下所示。

```
SELECT * FROM aws_lambda.invoke('aws_lambda_arn_1', '{"body": "Hello from
 Postgres!"}'::json);
ERROR:  lambda invocation failed
DETAIL:  "arn:aws:lambda:us-west-2:555555555555:function:my-function" returned error
 "Unhandled", details: "<Error details string>".
```

务必处理 Lambda 函数或 PostgreSQL 应用程序中的错误。

AWS Lambda 函数和参数参考

以下是通过 Aurora PostgreSQL 调用 Lambda 所用函数和参数的参考。

函数和参数

- aws_lambda.invoke
- aws_commons.create_lambda_function_arn
- aws_lambda 参数

aws_lambda.invoke

为 Aurora PostgreSQL 数据库集群 运行 Lambda 函数。

有关调用 Lambda 函数的更多详细信息，另请参阅 AWS Lambda 开发人员指南中的调用。

语法

JSON

```
aws_lambda.invoke(
IN function_name TEXT,
IN payload JSON,
IN region TEXT DEFAULT NULL,
IN invocation_type TEXT DEFAULT 'RequestResponse',
IN log_type TEXT DEFAULT 'None',
IN context JSON DEFAULT NULL,
IN qualifier VARCHAR(128) DEFAULT NULL,
OUT status_code INT,
OUT payload JSON,
OUT executed_version TEXT,
OUT log_result TEXT)
```

```
aws_lambda.invoke(
IN function_name aws_commons._lambda_function_arn_1,
IN payload JSON,
IN invocation_type TEXT DEFAULT 'RequestResponse',
IN log_type TEXT DEFAULT 'None',
IN context JSON DEFAULT NULL,
IN qualifier VARCHAR(128) DEFAULT NULL,
OUT status_code INT,
OUT payload JSON,
OUT executed_version TEXT,
OUT log_result TEXT)
```

JSONB

```
aws_lambda.invoke(
IN function_name TEXT,
IN payload JSONB,
IN region TEXT DEFAULT NULL,
IN invocation_type TEXT DEFAULT 'RequestResponse',
IN log_type TEXT DEFAULT 'None',
IN context JSONB DEFAULT NULL,
IN qualifier VARCHAR(128) DEFAULT NULL,
OUT status_code INT,
OUT payload JSONB,
OUT executed_version TEXT,
OUT log_result TEXT)
```

```
aws_lambda.invoke(
IN function_name aws_commons._lambda_function_arn_1,
IN payload JSONB,
IN invocation_type TEXT DEFAULT 'RequestResponse',
IN log_type TEXT DEFAULT 'None',
IN context JSONB DEFAULT NULL,
IN qualifier VARCHAR(128) DEFAULT NULL,
OUT status_code INT,
OUT payload JSONB,
OUT executed_version TEXT,
OUT log_result TEXT
)
```

输入参数

function_name

Lambda 函数的标识名称。该值可以是函数名称、ARN 或部分 ARN。有关可能的格式的列表，请参阅 AWS Lambda 开发人员指南中的 Lambda 函数名称格式。

payload

Lambda 函数的输入。格式可以是 JSON 或 JSONB。有关更多信息，请参阅 PostgreSQL 文档中的 JSON 类型。

区域

（可选）函数的 Lambda 区域。默认情况下，Aurora 会从 `function_name` 的完整 ARN 中解析 AWS 区域，或使用 Aurora PostgreSQL 数据库实例区域。如果此区域值与 `function_name` ARN 中提供的值冲突，则会引发错误。

invocation_type

Lambda 函数的调用类型。 值区分大小写。可能的值包括：

- `RequestResponse` – 默认值。Lambda 函数的这种调用类型是同步的，并在结果中返回响应负载。当工作流程依赖于立即接收 Lambda 函数结果时，请使用 `RequestResponse` 调用类型。
- `Event` – Lambda 函数的这种调用类型是异步的，并在没有返回负载的情况下立即返回。如果您不需要在工作流程开始之前了解 Lambda 函数的结果，请使用 `Event` 调用类型。
- `DryRun` – 这种类型的调用在不允许 Lambda 函数的情况下测试访问权限。

log_type

要在 `log_result` 输出参数中返回的 Lambda 日志类型。 值区分大小写。可能的值包括：

- Tail – 返回的 `log_result` 输出参数将包含执行日志的最后 4 kB。
- None – 未返回 Lambda 日志信息。

context

JSON 或 JSONB 格式的客户端上下文。要使用的字段包括 `custom` 和 `env`。

限定符

标识要调用的 Lambda 函数版本的限定符。如果此值与 `function_name` ARN 中提供的值冲突，则会引发错误。

输出参数

status_code

HTTP 状态响应代码。有关更多信息，请参阅 AWS Lambda 开发人员指南中的 Lambda 调用响应元素。

payload

从运行的 Lambda 函数返回的信息。格式为 JSON 或 JSONB。

executed_version

运行的 Lambda 函数的版本。

log_result

如果 `log_type` 值在调用 Lambda 函数时为 `Tail`，则会返回执行日志信息。结果包含以 Base64 编码的执行日志的最后 4 kB。

aws_commons.create_lambda_function_arn

创建一个 `aws_commons._lambda_function_arn_1` 结构来保存 Lambda 函数名称信息。您可以在 aws_lambda.invoke `aws_commons.create_lambda_function_arn` 函数的 `function_name` 参数中使用 aws_lambda.invoke 函数的结果。

语法

```
aws_commons.create_lambda_function_arn(
    function_name TEXT,
    region TEXT DEFAULT NULL
    )
    RETURNS aws_commons._lambda_function_arn_1
```

输入参数

function_name

包含 Lambda 函数名称的必需文本字符串。该值可以是函数名称、部分 ARN 或完整 ARN。

区域

一个包含 Lambda 函数所在的AWS区域的可选文本字符串。有关 区域名称和关联值的列表，请参阅区域及可用区。

aws_lambda 参数

在该表中，您可以找到与 aws_lambda 函数关联的参数。

参数	描述
aws_lambda.connect_timeout_ms	这是一个动态参数，用于设置连接到 AWS Lambda 时的最大等待时间。默认值为 1000。此参数允许的值为 1 - 900000。
aws_lambda.request_timeout_ms	这是一个动态参数，它设置了等待来自 AWS Lambda 的响应时的最大等待时间。默认值为 3000。此参数允许的值为 1 - 900000。
aws_lambda.endpoint_override	指定可用于连接到 AWS Lambda 的端点。空字符串会选择该区域的默认 AWS Lambda 端点。必须重启数据库，此静态参数更改才能生效。

将 Aurora PostgreSQL 日志发布到 Amazon CloudWatch Logs

您可以配置 Aurora PostgreSQL 数据库集群，以定期将日志数据导出到 Amazon CloudWatch Logs。这样做时，Aurora PostgreSQL 数据库集群的 PostgreSQL 日志中的事件将自动发布 到 Amazon CloudWatch（作为 Amazon CloudWatch Logs）。在 CloudWatch 中，您可以在 Aurora PostgreSQL 数据库集群的日志组 中找到导出的日志数据。日志组包含一个或多个日志流，其中包含来自集群中每个实例的 PostgreSQL 日志中的事件。

通过将日志发布到 CloudWatch Logs，您可以将集群的 PostgreSQL 日志记录保存在高持久性存储中。借助 CloudWatch Logs 中提供的日志数据，您可以评估和改进集群的操作情况。还可以使用 CloudWatch 来创建告警和查看指标。要了解更多信息，请参阅在 Amazon CloudWatch 中监控日志事件。

> **Note**
>
> 将您的 PostgreSQL 日志发布到 CloudWatch Logs 会消耗存储空间，并且您需要为该存储空间支付费用。请务必删除所有不再需要的 CloudWatch Logs。

对于现有 Aurora PostgreSQL 数据库集群关闭导出日志选项，不会影响 CloudWatch Logs 中已经保存的任何数据。根据您的日志保留设置，现有日志在 CloudWatch Logs 中仍然可用。要了解有关 CloudWatch Logs 的更多信息，请参阅什么是 Amazon CloudWatch Logs？

对于以下版本，Aurora PostgreSQL 支持将日志发布到 CloudWatch Logs。

- 16.1 及所有更高版本
- 15.2 及更高的 15 版本
- 14.3 及更高的 14 版本
- 13.3 及更高的 13 版本
- 12.8 及更高的 12.x 版本
- 11.9 及更高的 11 版本

有关开启将日志发布到 CloudWatch Logs 的选项、监控 CloudWatch Logs 中的日志事件以及使用 CloudWatch Logs Insights 分析日志的信息，请参阅以下主题。

主题

- 开启将日志发布到 Amazon CloudWatch 的选项
- 在 Amazon CloudWatch 中监控日志事件
- 使用 CloudWatch Logs Insights 分析 Aurora PostgreSQL 日志

开启将日志发布到 Amazon CloudWatch 的选项

要将 Aurora PostgreSQL 数据库集群的 PostgreSQL 日志发布到 CloudWatch Logs，请选择集群的 Log export（日志导出）选项。在创建 Aurora PostgreSQL 数据库集群时，您可以选择 Log export（日志导出）设置。或者，您可以稍后修改集群。当您修改现有集群时，其每个实例的 PostgreSQL 日志将从那时起发布到 CloudWatch 集群。对于 Aurora PostgreSQL，PostgreSQL 日志（`postgresql.log`）是唯一发布到 Amazon CloudWatch 的日志。

您可以使用 AWS Management Console、AWS CLI 或 RDS API 为您的 Aurora PostgreSQL 数据库集群开启 Log export（日志导出）功能。

控制台

您可以选择 Log export（日志导出）选项，以开始将 PostgreSQL 日志从 Aurora PostgreSQL 数据库集群发布到 CloudWatch Logs。

从控制台开启 Log export（日志导出）功能

1. 通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Databases（数据库）。
3. 选择要将其日志数据发布到 CloudWatch Logs 的 Aurora PostgreSQL 数据库集群。
4. 选择修改。
5. 在 Log exports（日志导出）部分中，选择 PostgreSQL log（PostgreSQL 日志）。
6. 选择 Continue (继续)，然后选择摘要页面上的 Modify cluster (修改集群)。

AWS CLI

您可以开启日志导出选项，以开始使用 AWS CLI 将 Aurora PostgreSQL 日志发布到 Amazon CloudWatch Logs。为此，请使用以下选项运行 AWS CLI 命令 modify-db-cluster：

- `--db-cluster-identifier`—数据库集群标识符。
- `--cloudwatch-logs-export-configuration` — 为数据库集群导出到 CloudWatch Logs 而设置的日志类型的配置设置。

您还可以运行以下 AWS CLI 命令之一来发布 Aurora PostgreSQL 日志：

- create-db-cluster
- restore-db-cluster-from-s3
- restore-db-cluster-from-snapshot
- restore-db-cluster-to-point-in-time

通过以下选项运行上述 AWS CLI 命令之一：

- `--db-cluster-identifier`—数据库集群标识符。
- `--engine` — 数据库引擎。
- `--enable-cloudwatch-logs-exports` — 为数据库集群导出到 CloudWatch Logs 而启用的日志类型的配置设置。

根据您运行的 AWS CLI 命令，可能需要其他选项。

Example

以下命令创建一个 Aurora PostgreSQL 数据库集群以将日志文件发布到 CloudWatch Logs。

对于 Linux、macOS 或 Unix：

```
aws rds create-db-cluster \
    --db-cluster-identifier my-db-cluster \
    --engine aurora-postgresql \
    --enable-cloudwatch-logs-exports postgresql
```

对于 Windows：

```
aws rds create-db-cluster ^
    --db-cluster-identifier my-db-cluster ^
    --engine aurora-postgresql ^
    --enable-cloudwatch-logs-exports postgresql
```

Example

以下命令修改现有的 Aurora PostgreSQL 数据库集群以将日志文件发布到 CloudWatch Logs。`--cloudwatch-logs-export-configuration` 值是 JSON 对象。此对象的键是 `EnableLogTypes`，其值为 `postgresql` 和 `instance`。

对于 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster \
    --db-cluster-identifier my-db-cluster \
    --cloudwatch-logs-export-configuration '{"EnableLogTypes":
["postgresql","instance"]}'
```

对于 Windows：

```
aws rds modify-db-cluster ^
    --db-cluster-identifier my-db-cluster ^
    --cloudwatch-logs-export-configuration '{\"EnableLogTypes\":[\"postgresql\",
\"instance\"]}'
```

> **Note**
>
> 使用 Windows 命令提示符时，确保在 JSON 代码中转义双引号 (")，方法是使用反斜杠 (\) 作为其前缀。

Example

以下示例修改现有的 Aurora PostgreSQL 数据库集群以禁用将日志文件发布到 CloudWatch Logs。`--cloudwatch-logs-export-configuration` 值是 JSON 对象。此对象的键是 `DisableLogTypes`，其值为 `postgresql` 和 `instance`。

对于 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbinstance \
    --cloudwatch-logs-export-configuration '{"DisableLogTypes":
["postgresql","instance"]}'
```

对于 Windows：

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbinstance ^
    --cloudwatch-logs-export-configuration "{\"DisableLogTypes\":[\"postgresql\",
\"instance\"]}"
```

> **Note**
>
> 使用 Windows 命令提示符时，必须在 JSON 代码中转义双引号 (")，方法是使用反斜杠 (\) 作为其前缀。

RDS API

您可以开启日志导出选项，以开始使用 RDS API 发布 Aurora PostgreSQL 日志。为此，您可以使用以下选项运行 ModifyDBCluster 操作：

- `DBClusterIdentifier` – 数据库集群标识符。
- `CloudwatchLogsExportConfiguration` – 为数据库集群导出到 CloudWatch Logs 而启用的日志类型的配置设置。

您还可以通过运行以下 RDS API 操作之一，使用 RDS API 发布 Aurora PostgreSQL 日志：

- CreateDBCluster
- RestoreDBClusterFromS3
- RestoreDBClusterFromSnapshot
- RestoreDBClusterToPointInTime

使用以下参数运行 RDS API 操作：

- `DBClusterIdentifier`—数据库集群标识符。
- `Engine` — 数据库引擎。
- `EnableCloudwatchLogsExports` — 为数据库集群导出到 CloudWatch Logs 而启用的日志类型的配置设置。

根据您运行的 AWS CLI 命令，可能需要其他参数。

在 Amazon CloudWatch 中监控日志事件

随着 Aurora PostgreSQL 日志事件已发布并可用作 Amazon CloudWatch Logs，您可以使用 Amazon CloudWatch 查看和监控事件。有关监控的更多信息，请参阅查看发送到 CloudWatch Logs 的日志数据。

当您开启日志导出时，将使用前缀 `/aws/rds/cluster/` 以及 Aurora PostgreSQL 的名称和日志类型自动创建一个新的日志组，如下模式所示。

```
/aws/rds/cluster/your-cluster-name/postgresql
```

例如，假设一个名为 `docs-lab-apg-small` 的 Aurora PostgreSQL 数据库集群将其日志导出到 Amazon CloudWatch Logs。它在 Amazon CloudWatch 中的日志组名称如下所示。

```
/aws/rds/cluster/docs-lab-apg-small/postgresql
```

如果存在具有指定名称的日志组，Aurora 将使用该日志组为 Aurora 数据库集群导出日志数据。Aurora PostgreSQL 数据库集群中的每个数据库实例都会将其 PostgreSQL 日志作为不同的日志流上载到日志组。您可以使用 Amazon CloudWatch 中提供的各种图形和分析工具，检查日志组及其日志流。

例如，您可以在 Aurora PostgreSQL 数据库集群的日志事件中搜索信息，并使用 CloudWatch Logs 控制台、AWS CLI 或 CloudWatch Logs API 来筛选事件。有关更多信息，请参阅《Amazon CloudWatch Logs 用户指南》中的搜索和筛选日志数据。

原定设置情况下，创建新的日志组时，对其保留期使用 Never expire（永不过期）。您可以使用 CloudWatch Logs 控制台、AWS CLI 或 CloudWatch Logs API 更改日志保留期。要了解更多信息，请参阅《Amazon CloudWatch Logs 用户指南》中的更改 CloudWatch Logs 中的日志数据留存。

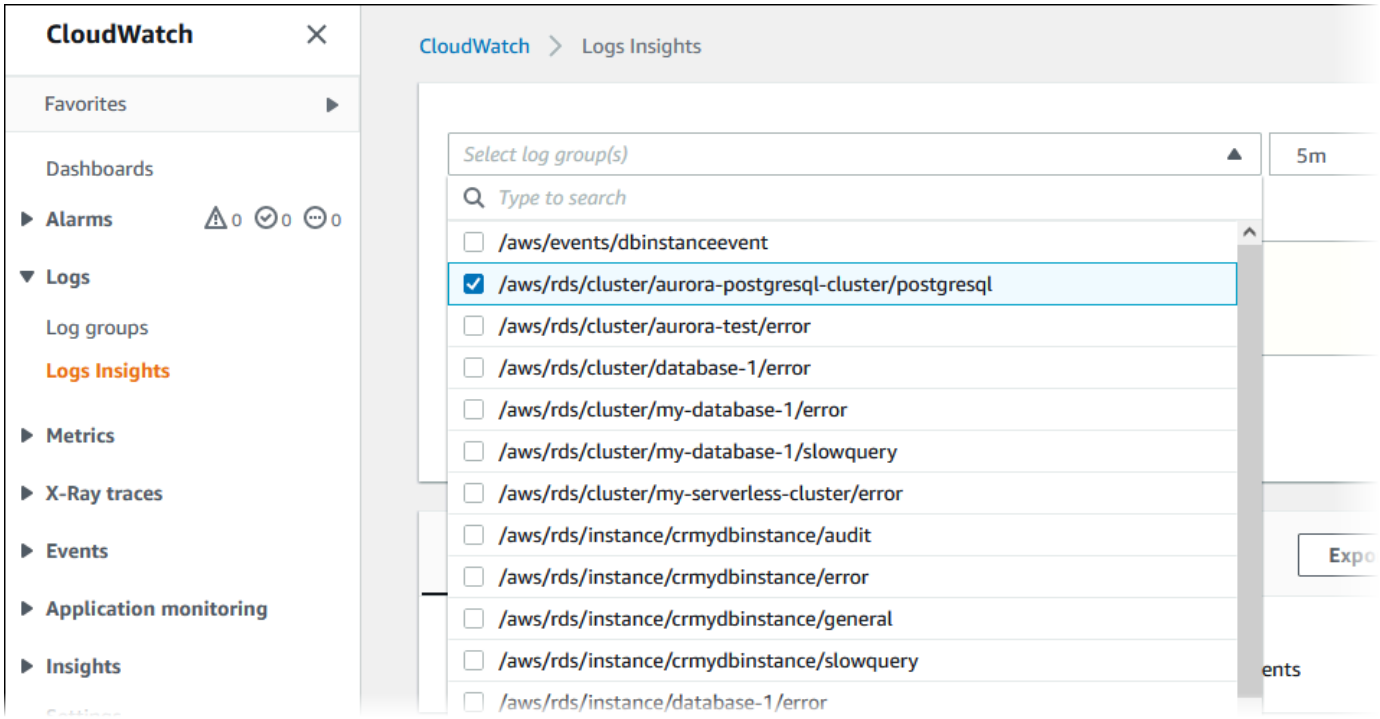
> **Tip**
>
> 您可以使用自动化配置（例如 AWS CloudFormation）创建具有预定义日志保留期、指标筛选条件和访问权限的日志组。

使用 CloudWatch Logs Insights 分析 Aurora PostgreSQL 日志

通过将 Aurora PostgreSQL 数据库集群中的 PostgreSQL 日志发布为 CloudWatch Logs，您可以使用 CloudWatch Logs Insights 在 Amazon CloudWatch Logs 中交互式搜索和分析日志数据。CloudWatch Logs Insights 包括查询语言、示例查询和其他用于分析日志数据的工具，以便您可以识别潜在问题并验证修复方法。要了解更多信息，请参阅《Amazon CloudWatch Logs 用户指南》中的使用 CloudWatch Logs Insights 分析日志数据。

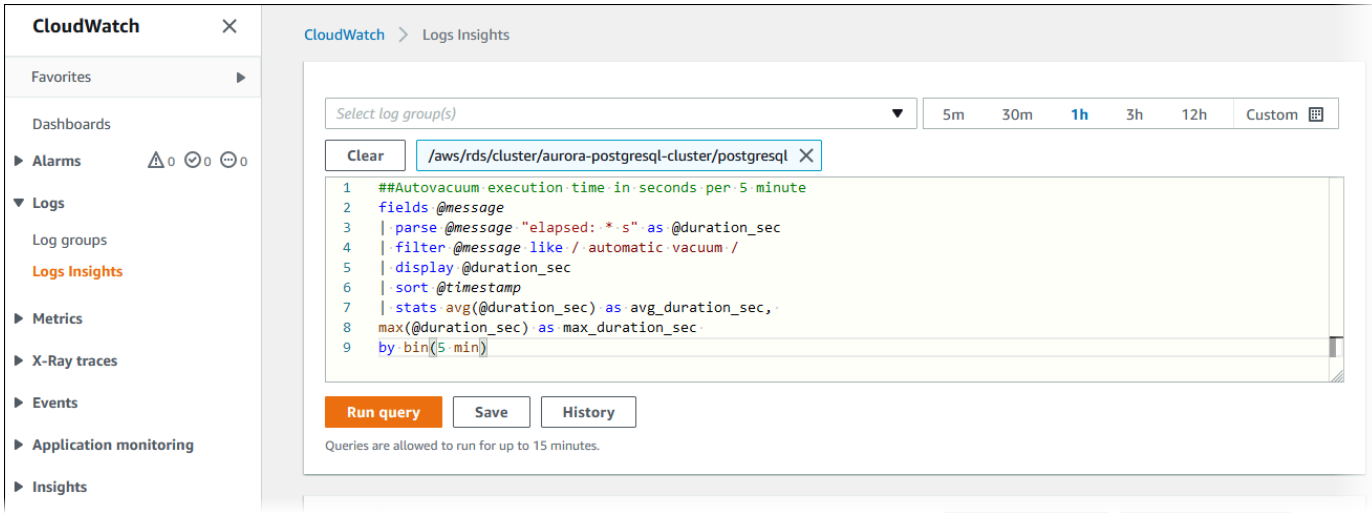
要使用 CloudWatch Logs Insights 分析 PostgreSQL 日志

1. 通过 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，打开 Logs（日志）并选择 Log insights（日志洞察）。
3. 在 Select log group(s)（选择日志组）中，为您的 Aurora PostgreSQL 数据库集群选择日志组。

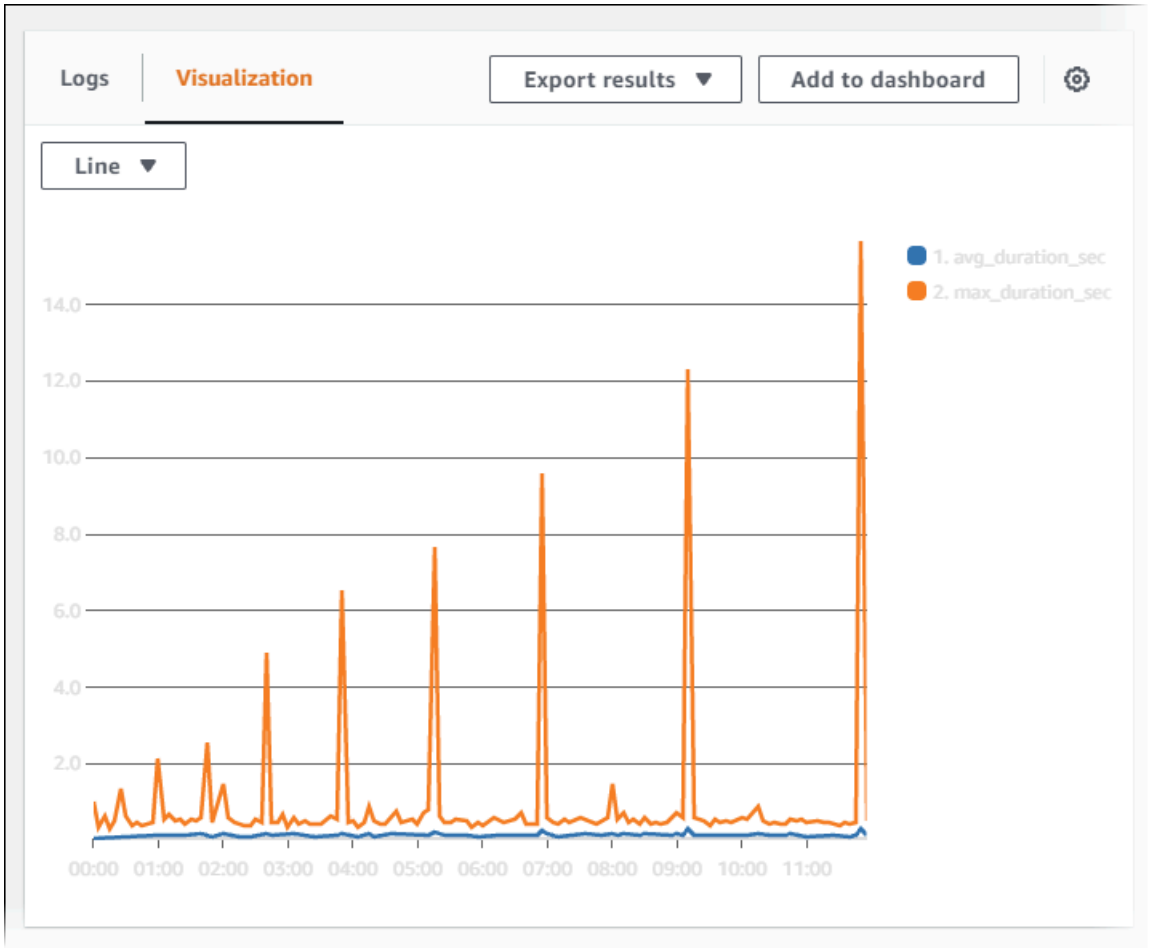


4. 在查询编辑器中，删除当前显示的查询，输入以下内容，然后选择 Run query（运行查询）。

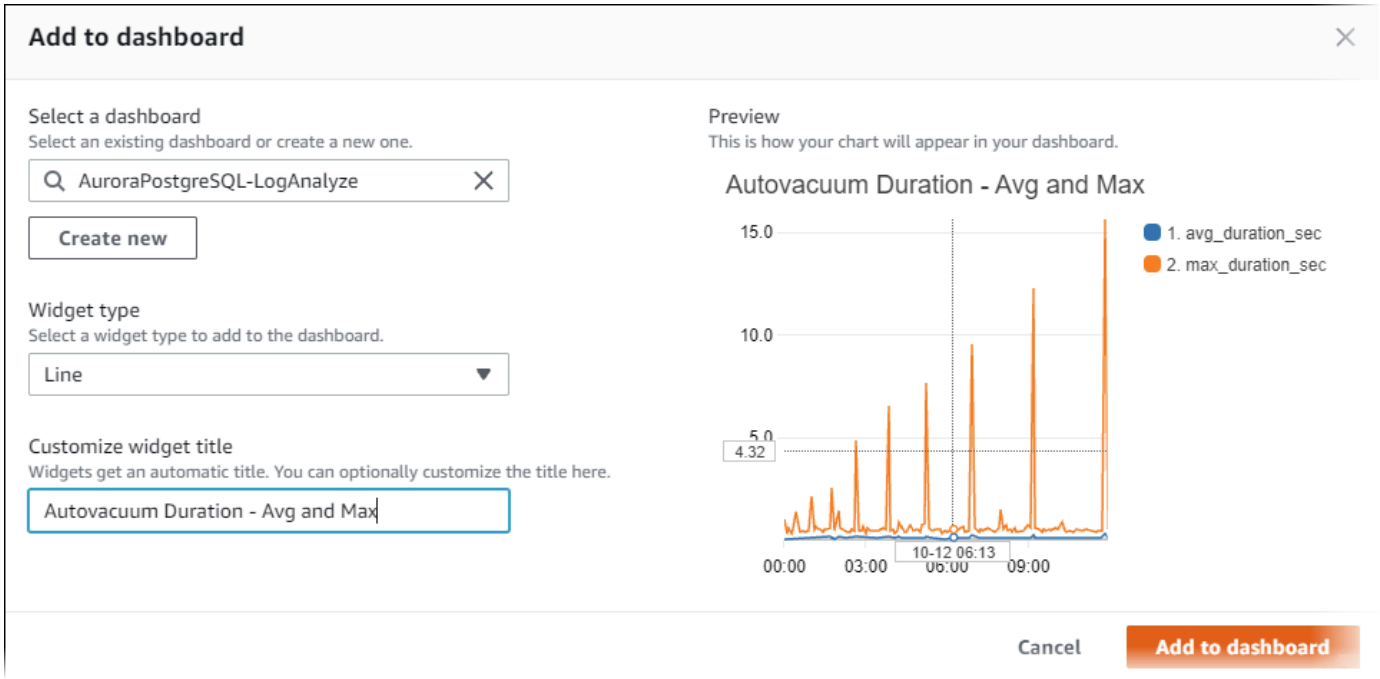
```
##Autovacuum execution time in seconds per 5 minute
fields @message
| parse @message "elapsed: * s" as @duration_sec
| filter @message like / automatic vacuum /
| display @duration_sec
| sort @timestamp
| stats avg(@duration_sec) as avg_duration_sec,
max(@duration_sec) as max_duration_sec
by bin(5 min)
```



5. 选择 Visualization (可视化) 选项卡。



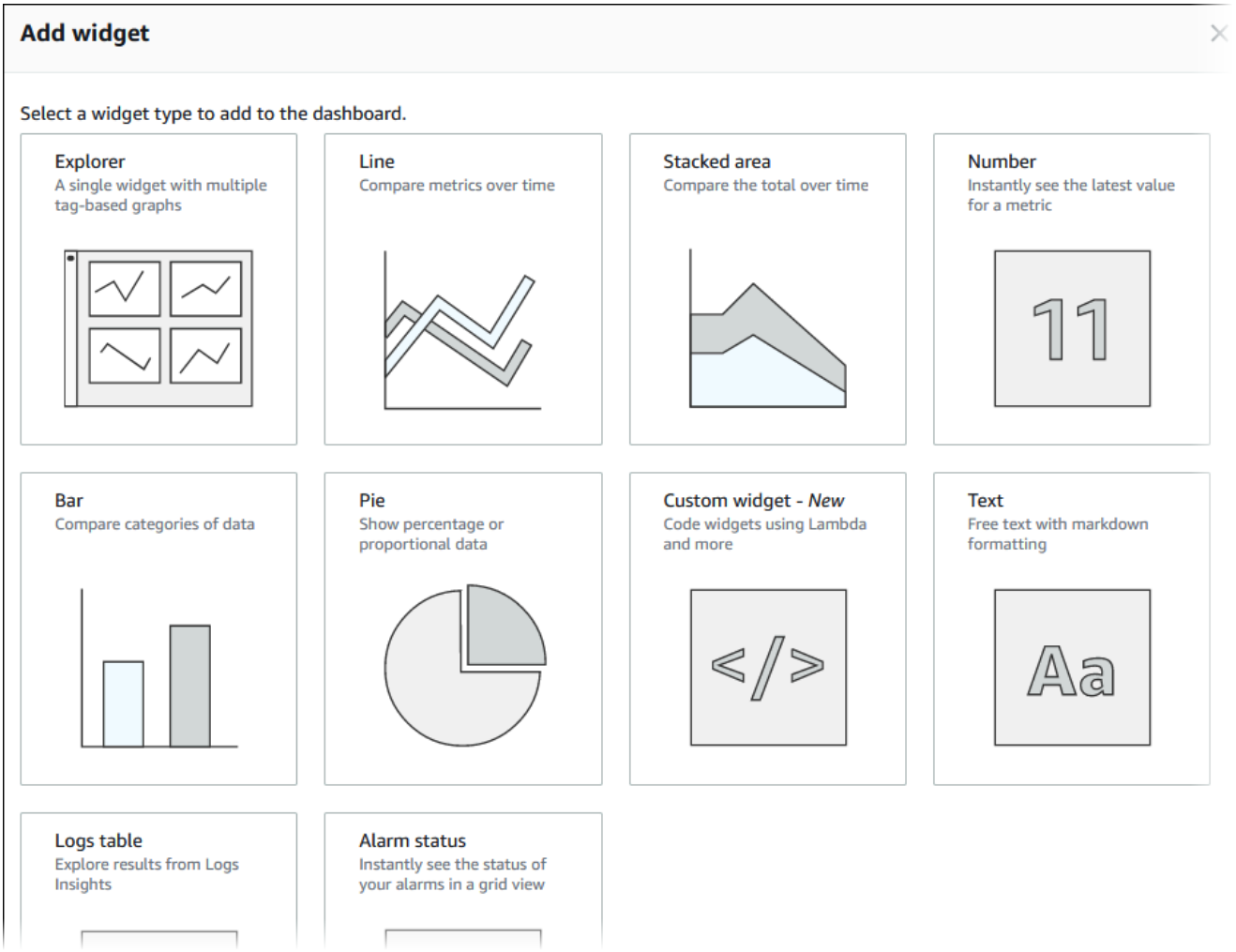
6. 选择 Add to dashboard（添加到控制面板）。
7. 在 Select a dashboard（选择控制面板）中，选择一个控制面板或输入名称以创建新的控制面板。
8. 在 Widget type（小组件类型）中，为您的可视化选择一个小组件类型。



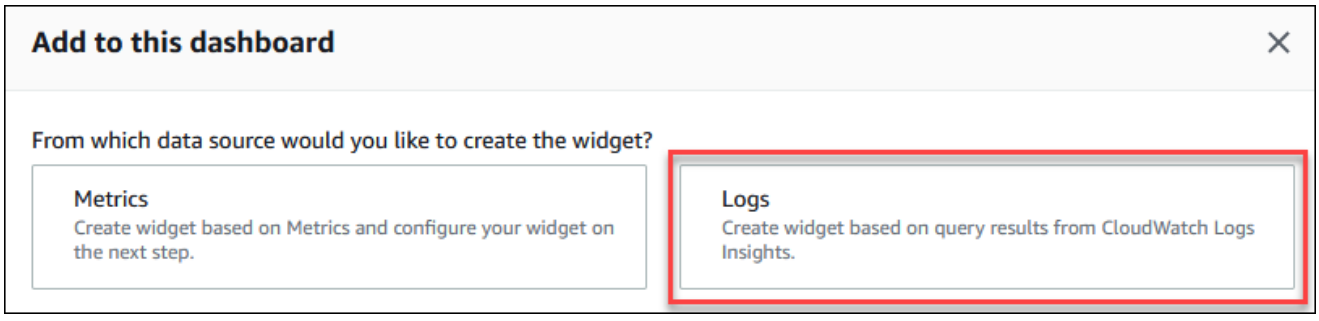
9. （可选）根据您的日志查询结果添加更多小组件。

   a. 选择 Add widget。

   b. 选择小组件类型，例如 Line。



c. 在 Add to this dashboard（添加到此控制面板）窗口中，选择 Logs（日志）。

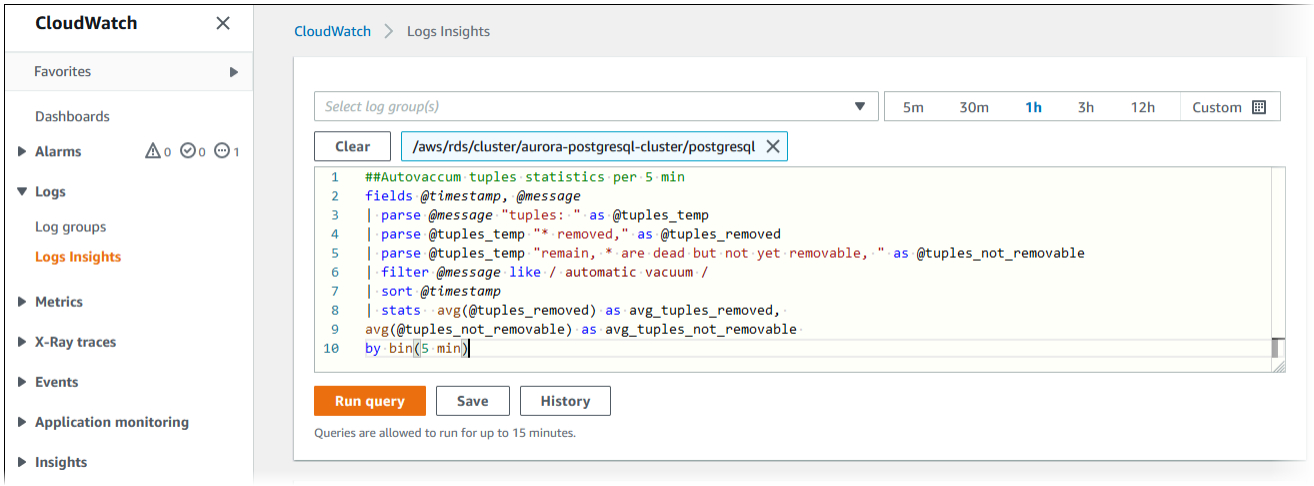


d. 在 Select log group(s)（选择日志组）中，为您的数据库集群选择日志组。

e. 在查询编辑器中，删除当前显示的查询，输入以下内容，然后选择 Run query（运行查询）。

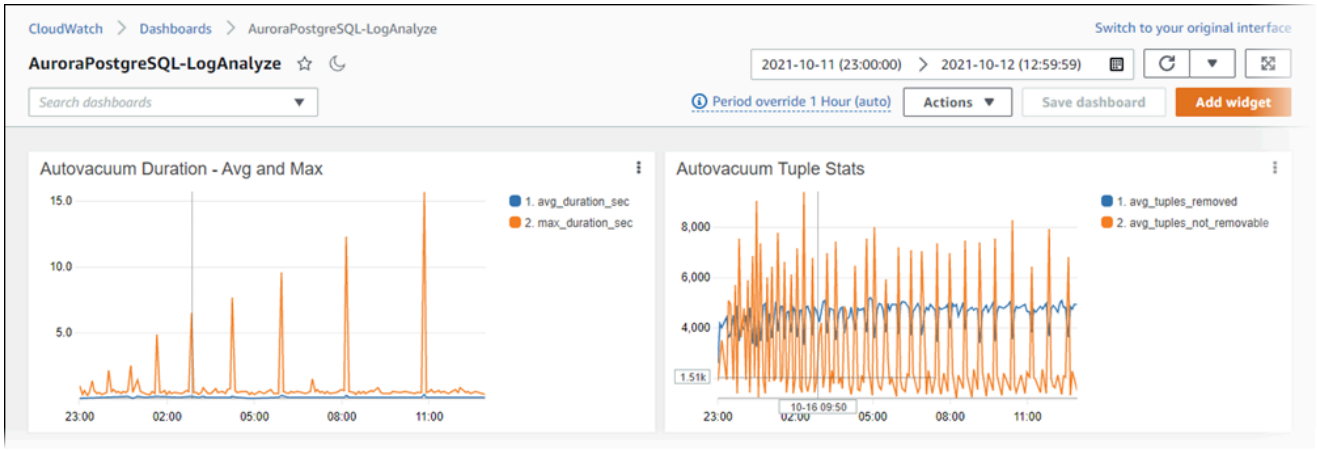
```
##Autovacuum tuples statistics per 5 min
fields @timestamp, @message
| parse @message "tuples: " as @tuples_temp
| parse @tuples_temp "* removed," as @tuples_removed
```

```
| parse @tuples_temp "remain, * are dead but not yet removable, " as
 @tuples_not_removable
| filter @message like / automatic vacuum /
| sort @timestamp
| stats  avg(@tuples_removed) as avg_tuples_removed,
avg(@tuples_not_removable) as avg_tuples_not_removable
by bin(5 min)
```



f. 选择 Create widget。

您的控制面板应类似于下图。



监控 Aurora PostgreSQL 的查询执行计划和峰值内存

您可以监控 Aurora PostgreSQL 数据库实例中的查询执行计划，以检测导致当前数据库负载的执行计划，并使用 aurora_compute_plan_id 参数跟踪执行计划随时间推移的性能统计数据。每当执行查询时，都会为查询使用的执行计划分配一个标识符，同一计划的后续执行将使用相同的标识符。

在 Aurora PostgreSQL 14.10、15.5 及更高版本中，数据库参数组中 aurora_compute_plan_id 默认为 OFF。要分配计划标识符，请在参数组中将 aurora_compute_plan_id 设置为 ON。

此计划标识符用于多个不同用途的实用程序中。

可以监控数据库实例中的查询峰值内存使用量，来检测导致以下版本中数据库内存使用量高的查询：

- 16.3 及所有更高版本
- 15.7 及更高版本
- 14.12 及更高版本

每当查询运行时，都会跟踪查询使用的峰值内存。查询通常会运行多次；可以查看每个查询跨所有运行的平均、最小和最大内存使用量值。

主题

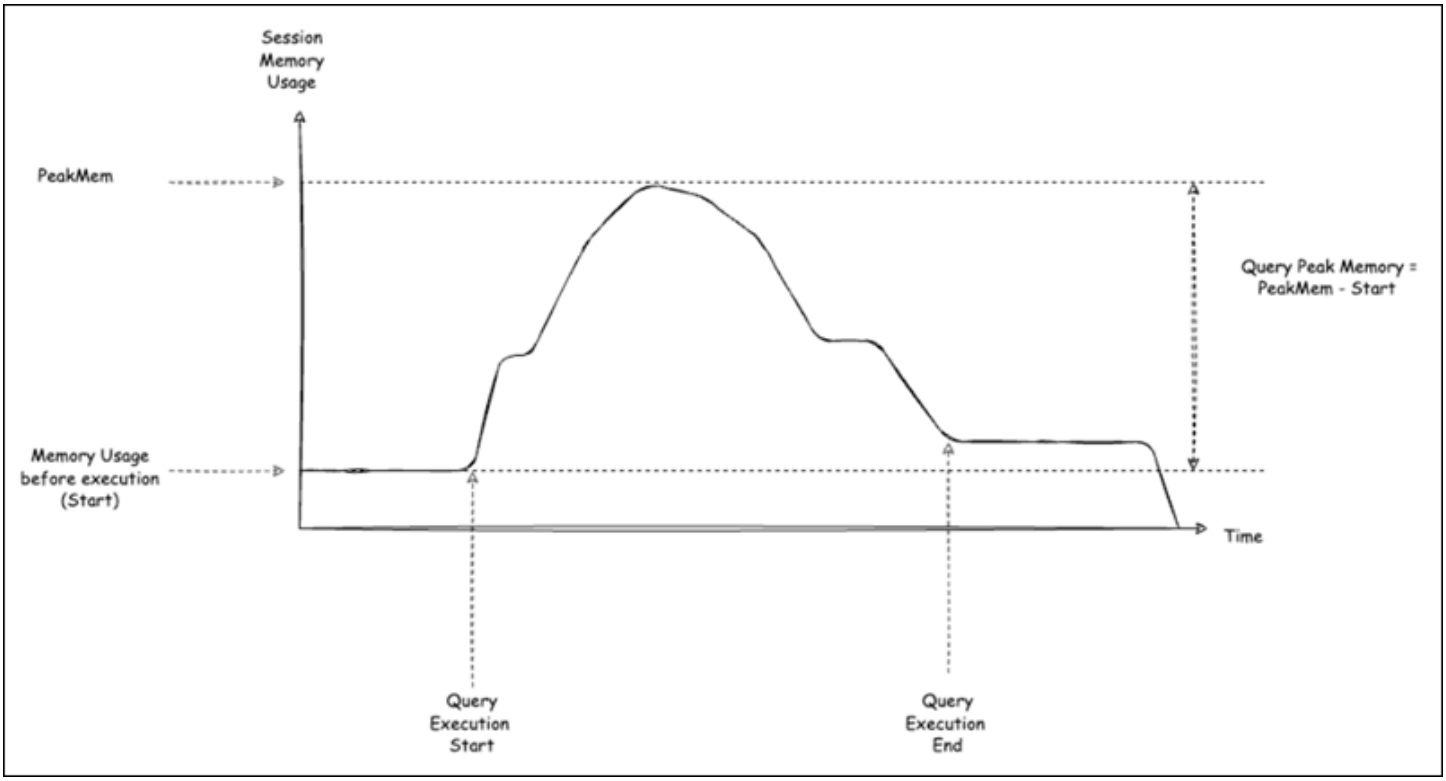
- 使用 Aurora 函数访问查询执行计划和峰值内存
- Aurora PostgreSQL 查询执行计划的参数参考

使用 Aurora 函数访问查询执行计划和峰值内存

使用 aurora_compute_plan_id，您可以使用以下函数访问执行计划：

- aurora_stat_activity
- aurora_stat_plans

查询峰值内存不包括在查询处理开始之前分配的内存。在每个查询的计划和执行阶段，单独跟踪和报告峰值内存使用量。



可以使用以下函数访问查询峰值内存统计数据：

- aurora_stat_statements
- aurora_stat_plans

有关这些函数的更多信息，请参阅Aurora PostgreSQL 函数参考。

Aurora PostgreSQL 查询执行计划的参数参考

您可以使用数据库参数组中的以下参数监控查询执行计划。

参数

- aurora_compute_plan_id
- aurora_stat_plans.minutes_until_recapture
- aurora_stat_plans.calls_until_recapture
- aurora_stat_plans.with_costs
- aurora_stat_plans.with_analyze
- aurora_stat_plans.with_timing
- aurora_stat_plans.with_buffers

- aurora_stat_plans.with_wal
- aurora_stat_plans.with_triggers

> **Note**
>
> aurora_stat_plans.with_* 参数的配置仅对新捕获的计划生效。

aurora_compute_plan_id

aurora_compute_plan_id 是一个配置参数，用于控制在查询执行期间是否分配计划标识符。

默认	允许值	描述
off	0（关闭）	设置为 off 以防止分配计划标识符。
	1（开启）	设置为 on 以分配计划标识符。

aurora_stat_plans.minutes_until_recapture

重新捕获计划之前要经过的分钟数。默认值为 0，表示将禁用重新捕获计划。如果超过了 aurora_stat_plans.calls_until_recapture 阈值，仍然可以重新捕获该计划。

默认	允许值	描述
0	0-1073741823	设置重新捕获计划之前要经过的分钟数。

aurora_stat_plans.calls_until_recapture

重新捕获计划之前调用该计划的次数。默认值为 0，表示在多次调用后将禁用重新捕获计划。如果超过了 aurora_stat_plans.minutes_until_recapture 阈值，仍然可以重新捕获该计划。

默认	允许值	描述
0	0-1073741823	设置重新捕获计划之前的调用次数。

aurora_stat_plans.with_costs

捕获 EXPLAIN 计划时包含估计成本。允许的值是 on 和 off。默认为 on。

默认	允许值	描述
on	0（关闭）	不显示每个计划节点的估计成本和行。
	1（开启）	显示每个计划节点的估计成本和行。

aurora_stat_plans.with_analyze

使用 ANALYZE 控制 EXPLAIN 计划。此模式仅在第一次捕获计划时使用。允许的值是 on 和 off。默认为 off。

默认	允许值	描述
off	0（关闭）	不包括计划的实际运行时间统计数据。
	1（开启）	包括计划的实际运行时间统计数据。

aurora_stat_plans.with_timing

使用 ANALYZE 时，将在解释中捕获计划时间。默认为 on。

默认	允许值	描述
on	0（关闭）	不包括实际启动时间和在每个计划节点上花费的时间。
	1（开启）	包括实际启动时间和在每个计划节点上花费的时间。

aurora_stat_plans.with_buffers

使用 ANALYZE 时，将在解释中捕获计划缓冲区的使用情况。默认为 off。

默认	允许值	描述
off	0（关闭）	不包括有关缓冲区使用情况的信息。
	1（开启）	包括有关缓冲区使用情况的信息。

aurora_stat_plans.with_wal

使用 ANALYZE 时，将在解释中捕获计划 wal 的使用情况。默认为 `off`。

默认	允许值	描述
off	0（关闭）	不包括有关 WAL 记录生成的信息。
	1（开启）	包括有关 WAL 记录生成的信息。

aurora_stat_plans.with_triggers

使用 `ANALYZE` 时，将在解释中捕获计划触发器执行统计信息。默认为 `off`。

默认	允许值	描述
off	0（关闭）	不包括触发器执行统计数据。
	1（开启）	包括触发器执行统计数据。

管理 Aurora PostgreSQL 的查询执行计划

Aurora PostgreSQL 查询计划管理是一项可选功能，可与 Amazon Aurora PostgreSQL 兼容版数据库集群结合使用。此功能打包为 apg_plan_mgmt 扩展，您可以将其安装在 Aurora PostgreSQL 数据库集群中。查询计划管理可让您管理优化器为 SQL 应用程序生成的查询执行计划。apg_plan_mgmt AWS 扩展基于 PostgreSQL 数据库引擎的原生查询处理功能而构建。

接下来，您可以找到有关 Aurora PostgreSQL 查询计划管理功能、如何设置此功能以及如何将其与 Aurora PostgreSQL 数据库集群结合使用的信息。在开始之前，我们建议您查看适用于您的 Aurora PostgreSQL 版本的特定 apg_plan_mgmt 扩展版本的所有版本注释。有关更多信息，请参阅《Aurora PostgreSQL 版本注释》中的 Aurora PostgreSQL apg_plan_mgmt 扩展版本。

主题

- Aurora PostgreSQL 查询计划管理概览
- Aurora PostgreSQL 查询计划管理的最佳实践
- Aurora PostgreSQL 查询计划管理
- 捕获 Aurora PostgreSQL 执行计划
- 使用 Aurora PostgreSQL 托管式计划
- 在 dba_plans 视图中检查 Aurora PostgreSQL 查询计划
- 改进 Aurora PostgreSQL 查询计划
- 删除 Aurora PostgreSQL 查询计划
- 导出和导入 Aurora PostgreSQL 的托管式计划
- Aurora PostgreSQL 查询计划管理的参数参考
- Aurora PostgreSQL 查询计划管理的函数参考
- Aurora PostgreSQL 兼容版的 apg_plan_mgmt.dba_plans 视图参考
- 查询计划管理高级功能

Aurora PostgreSQL 查询计划管理概览

Aurora PostgreSQL 查询计划管理旨在确保计划的稳定性，而无论对数据库的更改是否可能导致查询计划回归。当优化程序在系统或数据库更改后为给定的 SQL 语句选择次优计划时，就会发生查询计划回归。更改统计数据、约束、环境设置、查询参数绑定以及升级 PostgreSQL 数据库引擎都可能导致计划回归。

使用 Aurora PostgreSQL 查询计划管理，您可以控制更改查询执行计划的方式和时间。Aurora PostgreSQL 查询计划管理的优势包括以下几点。

- 通过强制优化程序从少量的已知良好计划中选择，提高计划的稳定性。
- 集中优化计划，然后在全局范围内分发最佳计划。
- 确定未使用的索引，并评估创建或删除索引的影响。
- 自动检测优化程序发现的最低成本的新计划。
- 尝试使用风险较小的新优化程序功能，因为您可以选择仅批准将提高性能的计划更改。

您可以主动使用查询计划管理提供的工具，为某些查询指定最佳计划。或者，您可以使用查询计划管理来应对不断变化的情况，并避免计划回归。有关更多信息，请参阅 Aurora PostgreSQL 查询计划管理的最佳实践。

主题

- 支持的 SQL 语句
- 查询计划管理限制
- 查询计划管理术语
- Aurora PostgreSQL 查询计划管理版本
- 开启 Aurora PostgreSQL 查询计划管理
- 升级 Aurora PostgreSQL 查询计划管理
- 关闭 Aurora PostgreSQL 查询计划管理

支持的 SQL 语句

查询计划管理支持以下类型的 SQL 语句。

- 任何 SELECT、INSERT、UPDATE 或 DELETE 语句，而不考虑复杂性。
- 已准备好语句。有关更多信息，请参阅 PostgreSQL 文档中的 PREPARE。
- 动态语句，包括那些在即时模式下运行的语句。有关更多信息，请参阅 PostgreSQL 文档中的动态 SQL 和 EXECUTE IMMEDIATE。
- 嵌入式 SQL 命令和语句。有关更多信息，请参阅 PostgreSQL 文档中的嵌入式 SQL 命令。
- 命名函数内的语句。有关更多信息，请参阅 PostgreSQL 文档中的 CREATE FUNCTION。
- 包含临时表的语句。
- 过程和 DO 块内的语句。

您可以在手动模式下将查询计划管理与 EXPLAIN 结合使用来捕获计划，而无需实际运行它。有关更多信息，请参阅 分析优化程序的已选择计划。要了解有关查询计划管理的模式（手动、自动）的更多信息，请参阅 捕获 Aurora PostgreSQL 执行计划。

Aurora PostgreSQL 查询计划管理支持所有 PostgreSQL 语言功能，包括分区表、继承、行级安全性和递归公用表表达式（CTE）。要了解有关这些 PostgreSQL 语言功能的更多信息，请参阅 PostgreSQL 文档中的表分区、行安全策略和 WITH 查询（公用表表达式）以及其他主题。

有关 Aurora PostgreSQL 查询计划管理功能的不同版本的信息，请参阅《Aurora PostgreSQL 版本注释》中的 Aurora PostgreSQL apg_plan_mgmt 扩展版本。

查询计划管理限制

Aurora PostgreSQL 查询计划管理的当前版本存在以下限制。

- 对于引用系统关系的语句不捕获计划 – 不捕获引用系统关系的语句，如 pg_class。这是特意设计的，目的是防止捕获大量由系统生成的供内部使用的计划。这也适用于视图内的系统表。
- 您的 Aurora PostgreSQL 数据库集群可能需要更大的数据库实例类 - 根据工作负载不同，查询计划管理可能需要具有超过 2 个 vCPU 的数据库实例类。max_worker_processes 的数量受数据库实例类大小的限制。2-vCPU 数据库实例类（例如 db.t3.medium）提供的 max_worker_processes 的数量可能不足应对给定的工作负载。如果您使用查询计划管理，我们建议您为 Aurora PostgreSQL 数据库集群选择一个具有超过 2 个 vCPU 的数据库实例类。

  如果数据库实例类无法支持工作负载，则查询计划管理会引发一条错误消息，如下所示。

  ```
  WARNING: could not register plan insert background process
  HINT: You may need to increase max_worker_processes.
  ```

  在这种情况下，您应该将 Aurora PostgreSQL 数据库集群纵向扩展到具有更多内存的数据库实例类大小。有关更多信息，请参阅 数据库实例类支持的数据库引擎。
- 已存储在会话中的计划不受影响 - 查询计划管理提供了一种在不更改应用程序代码的情况下影响查询计划的方法。但是，如果通用计划已存储在现有会话中，并且要更改其查询计划，则必须先在数据库集群参数组中将 plan_cache_mode 设置为 force_custom_plan。
- 在以下情况下，apg_plan_mgmt.dba_plans 和 pg_stat_statements 中的 queryid 可能分叉：
  - 对象存储在 apg_plan_mgmt.dba_plans 中后会被删除并重新创建。
  - apg_plan_mgmt.plans 表是从另一个集群导入的。

有关 Aurora PostgreSQL 查询计划管理功能的不同版本的信息，请参阅《Aurora PostgreSQL 版本注释》中的 Aurora PostgreSQL apg_plan_mgmt 扩展版本。

查询计划管理术语

本主题中使用了以下术语。

托管式语句

优化程序在查询计划管理下捕获的 SQL 语句。托管式语句在 apg_plan_mgmt.dba_plans 视图中存储了一个或多个查询执行计划。

计划基准

给定托管式语句的一组已批准计划。也就是说，托管式语句在 status 视图的 dba_plan 列中具有“已批准”状态的所有计划。

计划历史记录

给定托管式语句的所有已捕获计划的集合。计划历史记录包含为该语句捕获的所有计划，无论其状态如何。

查询计划回归

当优化程序选择的计划比在对数据库环境进行给定更改（例如，新的 PostgreSQL 版本或更改统计数据）之前选择的计划更不理想时，就会出现这种情况。

Aurora PostgreSQL 查询计划管理版本

所有当前可用的 Aurora PostgreSQL 版本均支持查询计划管理。有关更多信息，请参阅《Aurora PostgreSQL 版本注释》中的 Amazon Aurora PostgreSQL 更新的列表。

安装 apg_plan_mgmt 扩展时，查询计划管理功能会添加到您的 Aurora PostgreSQL 数据库集群中。不同版本的 Aurora PostgreSQL 支持不同版本的 apg_plan_mgmt 扩展。我们建议您将查询计划管理扩展升级到 Aurora PostgreSQL 版本的最新发行版。

> **Note**
>
> 有关每个 apg_plan_mgmt 扩展版本的版本注释，请参阅《Aurora PostgreSQL 版本注释》中的 Aurora PostgreSQL apg_plan_mgmt 扩展版本。

通过使用 psql 连接到实例并使用元命令 \dx 列出扩展，可以识别集群上运行的版本，如下所示。

```
labdb=> \dx
                        List of installed extensions
     Name      | Version |    Schema     |                            Description
---------------+---------+---------------
+-------------------------------------------------------------------------
 apg_plan_mgmt | 1.0     | apg_plan_mgmt | Amazon Aurora with PostgreSQL compatibility
 Query Plan Management
 plpgsql       | 1.0     | pg_catalog    | PL/pgSQL procedural language
(2 rows)
```

输出显示该集群正在使用 1.0 版本的扩展。只有某些 apg_plan_mgmt 版本适用于给定的 Aurora PostgreSQL 版本。在某些情况下，您可能需要将 Aurora PostgreSQL 数据库集群升级到新的次要版本或应用补丁，以便可以升级到最新版本的查询计划管理。输出中显示的 apg_plan_mgmt 版本 1.0 来自 Aurora PostgreSQL 版本 10.17 数据库集群，该集群没有可用的更高 apg_plan_mgmt 版本。在这种情况下，Aurora PostgreSQL 数据库集群应升级到最新版本的 PostgreSQL。

有关将 Aurora PostgreSQL 数据库集群升级到新版本 PostgreSQL 的更多信息，请参阅 Amazon Aurora PostgreSQL 的数据库引擎更新。

要了解如何升级 apg_plan_mgmt 扩展，请参阅 升级 Aurora PostgreSQL 查询计划管理。

开启 Aurora PostgreSQL 查询计划管理

为 Aurora PostgreSQL 数据库集群设置查询计划管理涉及安装扩展和更改多个数据库集群参数设置。您需要 rds_superuser 权限才能安装 apg_plan_mgmt 扩展和为 Aurora PostgreSQL 数据库集群开启此功能。

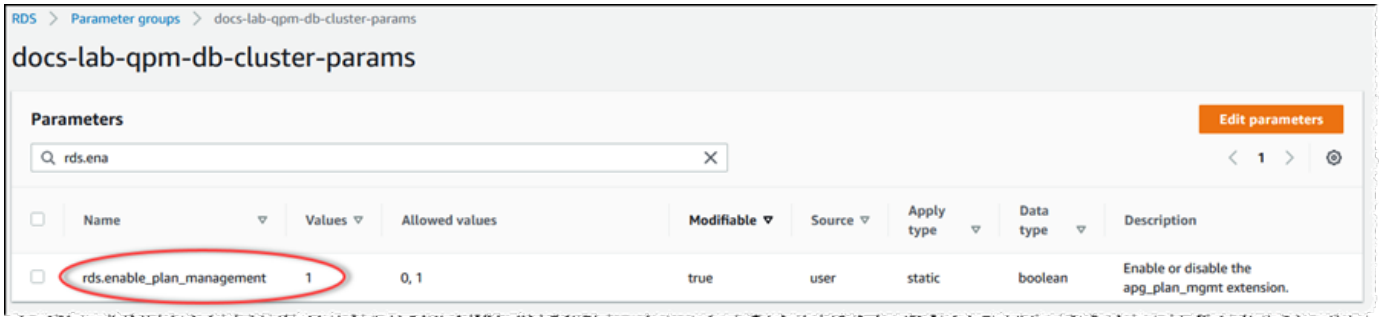
安装该扩展会创建一个新角色 apg_plan_mgmt。此角色可让数据库用户查看、管理和维护查询计划。作为具有 rds_superuser 权限的管理员，请务必根据需要将 apg_plan_mgmt 角色授予数据库用户。

只有具有 rds_superuser 角色的用户才能完成以下过程。rds_superuser 对于创建 apg_plan_mgmt 扩展及其 apg_plan_mgmt 角色是必需的。用户必须已授予了 apg_plan_mgmt 角色才能管理 apg_plan_mgmt 扩展。

为 Aurora PostgreSQL 数据库集群开启查询计划管理

以下步骤为提交到 Aurora PostgreSQL 数据库集群的所有 SQL 语句开启查询计划管理。这称为自动模式。要了解有关模式之间的差异的更多信息，请参阅 捕获 Aurora PostgreSQL 执行计划。

1. 通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 为 Aurora PostgreSQL 数据库集群创建自定义数据库集群参数组。您需要更改某些参数，才能激活查询计划管理并设置其行为。有关更多信息，请参阅 在 Amazon Aurora 中创建数据库参数组。
3. 打开自定义数据库集群参数组，并将 `rds.enable_plan_management` 参数设置为 1，如下图所示。



   有关更多信息，请参阅 在 Amazon Aurora 中修改数据库集群参数组中的参数。
4. 创建自定义数据库参数组，用于在实例级别设置查询计划参数。有关更多信息，请参阅 在 Amazon Aurora 中创建数据库集群参数组。
5. 修改 Aurora PostgreSQL 数据库集群的写入器实例，以使用自定义数据库参数组。有关更多信息，请参阅 修改数据库集群中的数据库实例。
6. 修改 Aurora PostgreSQL 数据库集群，以使用自定义数据库集群参数组。有关更多信息，请参阅 使用控制台、CLI 和 API 修改数据库集群。
7. 重启数据库实例以启用自定义参数组设置。
8. 使用 `psql` 或 pgAdmin 连接到 Aurora PostgreSQL 数据库集群的数据库实例端点。以下示例对于 `rds_superuser` 角色使用原定设置 `postgres` 账户。

```
psql --host=cluster-instance-1.111122223333.aws-region.rds.amazonaws.com --
port=5432 --username=postgres --password --dbname=my-db
```

9. 为数据库实例创建 `apg_plan_mgmt` 扩展，如下所示。

```
labdb=> CREATE EXTENSION apg_plan_mgmt;
CREATE EXTENSION
```

> **Tip**
>
> 在应用程序的模板数据库中安装 `apg_plan_mgmt` 扩展。缺省设置模板数据库名为 `template1`。要了解更多信息，请参阅 PostgreSQL 文档中的模板数据库。

10. 将 apg_plan_mgmt.capture_plan_baselines 参数更改为 automatic。此设置使优化程序为已计划或执行两次或更多次的每个 SQL 语句生成计划。

    > **Note**
    >
    > 查询计划管理还具有手动模式，可用于特定的 SQL 语句。要了解更多信息，请参阅捕获 Aurora PostgreSQL 执行计划。

11. 将 apg_plan_mgmt.use_plan_baselines 参数的值更改为“on”。该参数使优化程序可以从其计划基准中为该语句选择计划。要了解更多信息，请参阅使用 Aurora PostgreSQL 托管式计划。

    > **Note**
    >
    > 您可以修改会话的任一动态参数的值，且无需重启实例。

完成查询计划管理的设置后，请务必将 apg_plan_mgmt 角色授予需要查看、管理或维护查询计划的任何数据库用户。

升级 Aurora PostgreSQL 查询计划管理

我们建议您将查询计划管理扩展升级到 Aurora PostgreSQL 版本的最新发行版。

1. 以具有 rds_superuser 权限的用户身份连接到 Aurora PostgreSQL 数据库集群的写入器实例。如果您在设置实例时保留原定设置名称，则以 postgres 身份进行连接。此示例说明如何使用 psql，但如果您愿意，也可以使用 pgAdmin。

   ```
   psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --
   username=postgres --password
   ```

2. 运行以下查询以升级扩展。

   ```
   ALTER EXTENSION apg_plan_mgmt UPDATE TO '2.1';
   ```

3. 使用 apg_plan_mgmt.validate_plans 函数更新所有计划的哈希。优化程序会验证所有已批准、未批准和已拒绝的计划，以确保它们仍然是新版本扩展的可行计划。

   ```
   SELECT apg_plan_mgmt.validate_plans('update_plan_hash');
   ```

   要了解有关使用此函数的更多信息，请参阅 验证计划。

4. 使用 apg_plan_mgmt.reload 函数，通过 dba_plans 视图中经过验证的计划刷新共享内存中的任何计划。

```
SELECT apg_plan_mgmt.reload();
```

要了解有关可用于查询计划管理的所有函数的更多信息，请参阅 Aurora PostgreSQL 查询计划管理的函数参考。

关闭 Aurora PostgreSQL 查询计划管理

您可以随时通过关闭 apg_plan_mgmt.use_plan_baselines 和 apg_plan_mgmt.capture_plan_baselines 来禁用查询计划管理。

```
labdb=> SET apg_plan_mgmt.use_plan_baselines = off;

labdb=> SET apg_plan_mgmt.capture_plan_baselines = off;
```

Aurora PostgreSQL 查询计划管理的最佳实践

使用查询计划管理功能，您可以控制更改查询执行计划的方式和时间。作为 DBA，使用 QPM 时的主要目标包括防止在数据库发生更改时性能下降，以及控制是否允许优化程序使用新计划。在下面，您可以找到一些使用查询计划管理的推荐最佳实践。主动式和反应式计划管理方法的不同之处在于如何以及何时批准使用新计划。

目录

- 主动式计划管理有助于防止性能下降
  - 确保主要版本升级后的计划稳定性
- 反应式计划管理可检测和修复性能下降

主动式计划管理有助于防止性能下降

为了防止出现计划性能倒退，您可以通过运行一个过程来改进计划基线，该过程将新发现的计划的性能与已批准计划的现有基线的性能进行比较，然后自动批准最快的一组计划作为新的基线。这样，随着发现更快的计划，计划的基线会随着时间推移而提高。

1. 在开发环境中，确定对性能或系统吞吐量造成最大影响的 SQL 语句。然后捕获这些语句中的计划，如手动捕获特定 SQL 语句的计划和自动捕获计划中所述。

2. 将捕获的计划从开发环境导出并导入到生产环境中。有关更多信息，请参阅“导出和导入 Aurora PostgreSQL 的托管式计划”。
3. 在生产环境中，运行您的应用程序并强制使用批准的托管计划。有关更多信息，请参阅“使用 Aurora PostgreSQL 托管式计划”。在应用程序运行时，随着优化程序的发现还可添加新计划。有关更多信息，请参阅“自动捕获计划”。
4. 分析未批准的计划并批准那些执行良好的计划。有关更多信息，请参阅“评估计划性能”。
5. 在应用程序持续运行时，优化程序开始相应使用新计划。

确保主要版本升级后的计划稳定性

PostgreSQL 的每个主要版本都包括对查询优化程序的增强和更改，旨在提高性能。但是，优化器在早期版本中生成的查询执行计划可能会导致较新的升级版本性能下降。在主要版本升级后，您可以使用查询计划管理来解决这些性能问题并确保计划的稳定性。

即使同一个语句存在多个批准计划，优化器也始终使用成本最低的批准计划。升级后，优化器可能会发现新的计划，但它们将被保存为未批准的计划。这些计划只有在使用反应式计划管理和 unapproved_plan_execution_threshold 参数获得批准后才会执行。您可以使用反应式计划管理和 evolve_plan_baselines 参数来最大限度地提高计划的稳定性。这将新计划的绩效与旧计划的绩效进行比较，然后批准或拒绝比次优计划至少快 10% 的计划。

升级后，您可以使用 `evolve_plan_baselines` 函数来比较使用查询参数绑定进行升级前后的计划性能。以下步骤假定您一直在生产环境中使用经批准的托管计划，详情请参阅 使用 Aurora PostgreSQL 托管式计划。

1. 升级前，请在运行查询计划管理器的情况下运行应用程序。当应用程序运行时，在优化程序发现新计划时添加它们。有关更多信息，请参阅 自动捕获计划。
2. 评估每个计划的性能。有关更多信息，请参阅 评估计划性能。
3. 升级后，请再次使用 `evolve_plan_baselines` 函数分析经批准的计划。比较使用查询参数绑定前后的性能。如果新计划较快，您可以将其添加到经批准的计划中。如果它比相同参数绑定的另一个计划更快，那么您可以将较慢的计划标记为“已被拒绝”。

   有关更多信息，请参阅 批准更好的计划。有关此函数的参考信息，请参阅apg_plan_mgmt.evolve_plan_baselines。

有关更多信息，请参阅使用 Amazon Aurora PostgreSQL 兼容版查询计划管理确保主要版本升级后性能一致。

> **Note**
>
> 使用逻辑复制或 AWS DMS 执行主要版本升级时，请确保复制 apg_plan_mgmt 模式，以确保将现有计划复制到升级后的实例。有关逻辑复制的更多信息，请参阅 使用逻辑复制对 Aurora PostgreSQL 执行主要版本升级。

反应式计划管理可检测和修复性能下降

通过在应用程序运行时进行监控，您可以检测造成性能下降的计划。检测到性能下降时，您可以按照以下步骤手动拒绝或修复糟糕的计划：

1. 在应用程序运行时，强制使用托管计划并自动添加新发现的计划作为未批准的计划。有关更多信息，请参阅“使用 Aurora PostgreSQL 托管式计划”和“自动捕获计划”。
2. 监控正在运行应用程序的性能下降。
3. 在您发现计划性能下降时，请将计划的状态设置为 rejected。下次优化程序运行 SQL 语句时，它会自动忽略拒绝的计划，并改为使用不同的已批准计划。有关更多信息，请参阅“拒绝或禁用速度较慢的计划”。

   在某些情况下，您可能会偏好修复糟糕的计划而不是拒绝、禁用或删除该计划。使用 pg_hint_plan 扩展来试验改进计划。通过 pg_hint_plan，您使用特殊注释告知优化程序覆盖其通常创建计划的方式。有关更多信息，请参阅 使用 pg_hint_plan 修复计划。

Aurora PostgreSQL 查询计划管理

为 Aurora PostgreSQL 数据库集群开启查询计划管理后，优化程序会为其多次处理的任何 SQL 语句生成和存储查询执行计划。优化程序始终将托管式语句首次生成的计划的状态设置为 Approved，并将它存储在 dba_plans 视图中。

为托管式语句保存的一组已批准计划称为其计划基准。随着应用程序运行，优化程序可能会为托管式语句生成其他计划。优化程序将其他捕获的计划的状态设置为 Unapproved。

稍后，您可以确定 Unapproved 计划的执行是否良好，并将其更改为 Approved、Rejected 或 Preferred。为此，您可以使用 apg_plan_mgmt.evolve_plan_baselines 函数或 apg_plan_mgmt.set_plan_status 函数。

当优化程序为 SQL 语句生成计划时，查询计划管理会将该计划保存在 apg_plan_mgmt.plans 表中。已被授予 apg_plan_mgmt 角色的数据库用户可以通过查询 apg_plan_mgmt.dba_plans 视图

来查看计划详细信息。例如，以下查询列出了非生产 Aurora PostgreSQL 数据库集群的视图中当前计划的详细信息。

- sql_hash – SQL 语句的标识符，它是 SQL 语句的标准化文本的哈希值。
- plan_hash – 计划的唯一标识符，它是 sql_hash 和计划的哈希值的组合。
- status – 计划的状态。优化程序可以运行已批准的计划。
- enabled – 指示计划是否准备就绪，可供使用（true）或不可以使用（false）。
- plan_outline – 计划的表示，用于重新创建实际的执行计划。树结构中的运算符映射到 EXPLAIN 输出中的运算符。

apg_plan_mgmt.dba_plans 视图具有更多的列，其中包含计划的所有详细信息，例如上次使用计划的时间。有关完整的详细信息，请参阅 Aurora PostgreSQL 兼容版的 apg_plan_mgmt.dba_plans 视图参考。

标准化和 SQL 哈希

在 apg_plan_mgmt.dba_plans 视图中，您可以通过 SQL 哈希值来确定托管式语句。SQL 哈希以 SQL 语句的标准化表示形式进行计算，消除了一些差异（例如文本值）。

每条 SQL 语句的标准化过程都会保留空格和大小写，以便您仍然可以阅读和理解 SQL 语句的要点。标准化会删除或替换以下项目。

- 前导块注释
- EXPLAIN 关键字和 EXPLAIN 选项，以及 EXPLAIN ANALYZE
- 尾随空格
- 所有文本

以下面的语句为例。

```
/*Leading comment*/ EXPLAIN SELECT /* Query 1 */ * FROM t WHERE x > 7 AND y = 1;
```

查询计划管理按以下所示对此语句进行标准化：

```
SELECT /* Query 1 */ * FROM t WHERE x > CONST AND y = CONST;
```

标准化允许使用相同的 SQL 哈希来处理可能只有文本值或参数值不同的相似 SQL 语本。换句话说，同一个 SQL 哈希可能存在多个计划，在每种不同的条件下都有一个最优计划。

> **Note**
>
> 用于不同模式的单个 SQL 语句具有不同的计划，因为它在运行时绑定到特定的模式。计划程序使用有关模式绑定的统计数据来选择最优计划。

要了解有关优化程序如何选择计划的更多信息，请参阅 使用 Aurora PostgreSQL 托管式计划。在该节中，您可以了解到如何在实际使用计划之前使用 EXPLAIN 和 EXPLAIN ANALYZE 预览计划。有关更多信息，请参阅 分析优化程序的已选择计划。有关概述选择计划的过程的图像，请参阅 优化程序如何选择要运行的计划。

捕获 Aurora PostgreSQL 执行计划

Aurora PostgreSQL 查询计划管理提供两种不同的模式来捕获查询执行计划：即自动或手动。您可以通过将 apg_plan_mgmt.capture_plans_baselines 的值设置为 automatic 或 manual 来选择模式。您可以使用手动计划捕获为特定的 SQL 语句捕获执行计划。或者，您可以使用自动计划捕获来捕获应用程序运行时多次执行的所有（或者速度最慢）计划。

捕获计划时，优化程序将托管语句首次捕获的计划的状态设置为 approved。优化程序将为托管语句捕获的任何其他计划的状态设置为 unapproved。但是，有时可能有多个计划保存为 approved 状态。在为一个语句并行创建了多个计划并在提交语句的第一个计划之前，可能发生这种情况。

要控制可以捕获并存储在 dba_plans 视图中的最大计划数量，请在数据库实例级参数组中设置 apg_plan_mgmt.max_plans 参数。修改 apg_plan_mgmt.max_plans 参数后需要重启数据库实例，才能让新值生效。有关更多信息，请参阅 apg_plan_mgmt.max_plans 参数。

手动捕获特定 SQL 语句的计划

如果您已知要管理的 SQL 语句集，请将语句放入 SQL 脚本文件，然后手动捕获计划。下面显示了如何为 SQL 语句集手动捕获查询计划的 psql 示例。

```
psql> SET apg_plan_mgmt.capture_plan_baselines = manual;
psql> \i my-statements.sql
psql> SET apg_plan_mgmt.capture_plan_baselines = off;
```

在捕获各个 SQL 语句的计划之后，优化程序添加新行到 apg_plan_mgmt.dba_plans 视图。

我们建议您在 SQL 脚本文件中使用 EXPLAIN 或 EXPLAIN EXECUTE 语句。请确保在参数值中加入足够的变体以捕获所有感兴趣的计划。

如果您知道比优化程序最低成本计划更好的计划，则可以强制优化程序使用更好的计划。要这样做，请指定一个或多个优化程序提示。有关更多信息，请参阅“使用 pg_hint_plan 修复计划”。要比较 unapproved 和 approved 计划的性能并批准、拒绝或删除它们，请参阅评估计划性能。

自动捕获计划

对于类似以下的情况，请使用自动计划捕获：

- 您不知道要管理的具体 SQL 语句。
- 您有数百乃至数千个 SQL 语句需要管理。
- 您的应用程序使用客户端 API。例如，JDBC 使用未命名的预编译语句或者批量模式语句，无法在 psql 中表达。

自动捕获计划

1. 通过在数据库实例级参数组中将 apg_plan_mgmt.capture_plan_baselines 设置为 automatic 来启用自动计划捕获。有关更多信息，请参阅 在 Amazon Aurora 中修改数据库参数组中的参数。
2. 随着应用程序运行，优化程序捕获各个至少运行两次的 SQL 语句的计划。

   随着应用程序使用默认查询计划管理参数设置运行，优化程序捕获各个至少运行两次的 SQL 语句的计划。在使用默认值的同时捕获所有计划具有非常小的运行时开销，并且可在生产中启用。

禁用自动计划捕获

- 从数据库实例级参数组，将 apg_plan_mgmt.capture_plan_baselines 参数设置为 off。

要评估未批准计划的性能并批准、拒绝或删除这些计划，请参阅评估计划性能。

使用 Aurora PostgreSQL 托管式计划

要让优化程序为托管语句使用捕获的计划，请将参数 `apg_plan_mgmt.use_plan_baselines` 设置为 `true`。以下是本地实例的示例。

```
SET apg_plan_mgmt.use_plan_baselines = true;
```

当应用程序运行时，此设置会导致优化程序为每个托管式语句使用有效且启用的最低成本、首选或已批准计划。

分析优化程序的已选择计划

当 `apg_plan_mgmt.use_plan_baselines` 参数设置为 `true` 时，您可以使用 EXPLAIN ANALYZE SQL 语句，使优化程序在要运行语句时显示所使用的计划。示例如下：

```
EXPLAIN ANALYZE EXECUTE rangeQuery (1,10000);
```

```
                                                    QUERY PLAN
----------------------------------------------------------------------------
 Aggregate  (cost=393.29..393.30 rows=1 width=8) (actual time=7.251..7.251 rows=1
 loops=1)
   ->  Index Only Scan using t1_pkey on t1 t  (cost=0.29..368.29 rows=10000 width=0)
 (actual time=0.061..4.859 rows=10000 loops=1)
Index Cond: ((id >= 1) AND (id <= 10000))
         Heap Fetches: 10000
 Planning time: 1.408 ms
 Execution time: 7.291 ms
 Note: An Approved plan was used instead of the minimum cost plan.
 SQL Hash: 1984047223, Plan Hash: 512153379
```

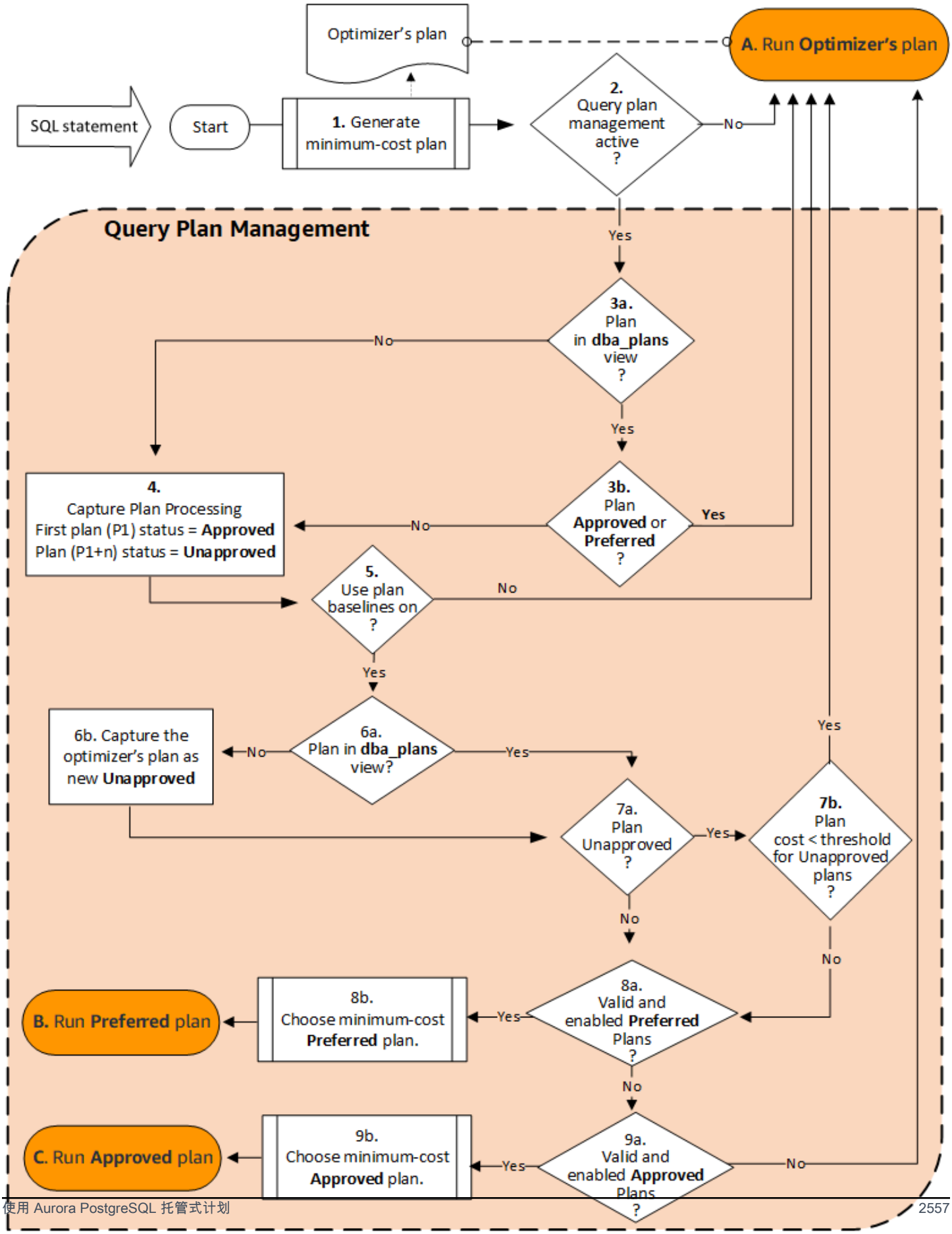
输出显示了将运行的基准中的已批准计划。但是，输出还显示它发现了更低成本的计划。在这种情况下，您通过启用自动计划捕获（如自动捕获计划中所述），捕获这个最低成本计划。

优化程序始终将新计划捕获为 `Unapproved`。使用 `apg_plan_mgmt.evolve_plan_baselines` 函数可比较计划，并将其更改为已批准、已拒绝或已禁用。有关更多信息，请参阅 评估计划性能。

优化程序如何选择要运行的计划

执行计划的成本是优化程序用于比较不同计划进行的估算。计算计划的成本时，优化程序包括该计划所需的 CPU 和 I/O 操作等因素。要了解有关 PostgreSQL 查询计划程序成本估计的更多信息，请参阅 PostgreSQL 文档中的查询计划。

下图显示了在查询计划管理处于活动状态以及未处于活动状态时，如何为给定的 SQL 语句选择计划。


Optimizer's plan
A. Run Optimizer's plan
SQL statement
Start
1. Generate minimum-cost plan
2. Query plan management active ?
No
Query Plan Management
Yes
3a. Plan in dba_plans view ?
No
Yes
4. Capture Plan Processing
First plan (P1) status = Approved
Plan (P1+n) status = Unapproved
3b. Plan Approved or Preferred ?
No
Yes
5. Use plan baselines on ?
No
Yes
6a. Plan in dba_plans view?
No
Yes
6b. Capture the optimizer's plan as new Unapproved
7a. Plan Unapproved ?
Yes
7b. Plan cost < threshold for Unapproved plans ?
Yes
No
No
8a. Valid and enabled Preferred Plans ?
Yes
8b. Choose minimum-cost Preferred plan.
B. Run Preferred plan
No
9a. Valid and enabled Approved Plans ?
Yes
9b. Choose minimum-cost Approved plan.
C. Run Approved plan
No

流程如下：

1. 优化程序为 SQL 语句生成最低成本计划。
2. 如果查询计划管理未处于活动状态，则优化程序的计划将立即运行（A. 运行优化程序的计划）。当 `apg_plan_mgmt.capture_plan_baselines` 和 `apg_plan_mgmt.use_plan_baselines` 参数都处于原定设置（分别为“off”和“false”）时，查询计划管理处于非活动状态。

   否则，查询计划管理处于活动状态。在这种情况下，在选择计划之前，将进一步评估 SQL 语句及其优化程序的计划。

   > **Tip**
   >
   > 具有 `apg_plan_mgmt` 角色的数据库用户可以主动比较计划，更改计划的状态，并根据需要强制使用特定的计划。有关更多信息，请参阅 改进 Aurora PostgreSQL 查询计划。

3. SQL 语句可能已经具有过去由查询计划管理存储的计划。计划以及用于创建这些计划的 SQL 语句的相关信息一起存储在 `apg_plan_mgmt.dba_plans` 中。有关计划的信息包括其状态。计划的状态可以决定是否使用该计划，如下所示。

   a. 如果该计划不在 SQL 语句的存储计划中，这意味着这是优化程序第一次为给定的 SQL 语句生成该特定的计划。该计划将发送到“捕获计划处理（4）”。

   b. 如果该计划在存储的计划中，且其状态为“已批准”或“首选”，则运行该计划（A. 运行优化程序的计划）。

      如果该计划在存储的计划中，但它既不是“已批准”，也不是“首选”，则该计划将发送到“捕获计划处理（4）”。

4. 首次捕获给定 SQL 语句的计划时，该计划的状态始终设置为“已批准（P1）”。如果优化程序随后为同一 SQL 语句生成相同的计划，则该计划的状态将更改为“未批准（P1+n）”。

   捕获计划并更新其状态后，将在下一步（5）继续进行评估。

5. 计划的基准 由 SQL 语句的历史记录及其在不同状态下的计划组成。查询计划管理可以在选择计划时考虑基准，具体取决于是否开启了使用计划基准选项，如下所示。

   - 当 `apg_plan_mgmt.use_plan_baselines` 参数设置为其原定设置值（`false`）时，“使用计划基准”处于“关闭”状态。该计划在运行之前不会与基准进行比较（A. 运行计划程序的计划）。
   - 当 `apg_plan_mgmt.use_plan_baselines` 参数设置为 `true` 时，“使用计划基准”为“开启”。使用基准（6）进一步评估该计划。

6. 将该计划与基准中用于此语句的其他计划进行比较。

a. 如果优化程序的计划属于基准中的计划，则检查其状态（7a）。

b. 如果优化程序的计划不属于基准中的计划，则该计划将作为新的 Unapproved 计划添加到语句的计划中。

7. 检查计划的状态以确定其是否为“未批准”。

a. 如果计划的状态为“未批准”，则将该计划的估计成本与为未批准的执行计划阈值指定的成本估计值进行比较。

- 如果计划的估计成本低于阈值，则即使它是未批准的计划，优化程序也会使用它（A. 运行优化程序的计划）。通常，优化程序不会运行未批准的计划。但是，当 apg_plan_mgmt.unapproved_plan_execution_threshold 参数指定成本阈值时，优化程序会将未批准计划的成本与阈值进行比较。如果估计的成本低于阈值，则优化程序运行此计划。有关更多信息，请参阅 apg_plan_mgmt.unapproved_plan_execution_threshold。
- 如果计划的估计成本不低于阈值，则检查该计划的其他属性（8a）。

b. 如果计划的状态为“未批准”以外的任何状态，则检查它的其他属性（8a）。

8. 优化程序不会使用禁用的计划。也就是其 enable 属性设置为“f”（false）的计划。优化程序也不会使用状态为“已拒绝”的计划。

   优化程序无法使用任何无效的计划。随着时间的推移，当计划所依赖的对象（如索引和表分区）被移除或删除时，计划可能会变为无效。

a. 如果语句具有任何已启用且有效的首选计划，则优化程序从为此 SQL 语句存储的首选计划中选择最低成本计划。然后，优化程序运行最低成本的首选计划。

b. 如果语句没有任何已启用且有效的首选计划，则将在下一步（9）中对其进行评估。

9. 如果语句具有任何已启用且有效的已批准计划，则优化程序从为此 SQL 语句存储的已批准计划中选择最低成本计划。然后，优化程序运行最低成本的已批准计划。

   如果语句没有任何有效且已启用的已批准计划，则优化程序将使用最低成本计划（A. 运行优化程序的计划）。

在 dba_plans 视图中检查 Aurora PostgreSQL 查询计划

已被授予 apg_plan_mgmt 角色的数据库用户和管理员可以查看和管理存储在 apg_plan_mgmt.dba_plans 中的计划。Aurora PostgreSQL 数据库集群的管理员（具有 rds_superuser 权限的用户）必须将此角色显式授予需要处理查询计划管理的数据库用户。

apg_plan_mgmt 视图包含 Aurora PostgreSQL 数据库集群的写入器实例上每个数据库的所有托管式 SQL 语句的计划历史记录。此视图可让您检查计划、计划的状态、上次使用时间以及所有其他相关详细信息。

如标准化和 SQL 哈希中所讨论，每个托管式计划均由 SQL 哈希值和计划哈希值的组合来标识。通过这些标识符，您可以使用 Amazon RDS Performance Insights 等工具来跟踪单独计划的性能。有关 Performance Insights 的更多信息，请参阅使用 Amazon RDS Performance Insights。

列出托管计划

要列出托管计划，请在 apg_plan_mgmt.dba_plans 视图上使用 SELECT 语句。以下示例显示 dba_plans 视图中的一些列，例如 status，以标识已批准和未批准计划。

```
SELECT sql_hash, plan_hash, status, enabled, stmt_name
FROM apg_plan_mgmt.dba_plans;

 sql_hash   | plan_hash |   status   | enabled | stmt_name
------------+-----------+------------+---------+------------
 1984047223 | 512153379 | Approved   | t       | rangequery
 1984047223 | 512284451 | Unapproved | t       | rangequery
 (2 rows)
```

为便于阅读，所显示的查询和输出仅列出了 dba_plans 视图中的若干列。有关完整信息，请参阅 Aurora PostgreSQL 兼容版的 apg_plan_mgmt.dba_plans 视图参考。

改进 Aurora PostgreSQL 查询计划

通过评估计划绩效并修复计划来改进查询计划管理。有关改进查询计划的更多信息，请参阅以下主题。

主题

- 评估计划性能
- 使用 pg_hint_plan 修复计划

评估计划性能

在优化程序将计划作为未批准计划捕获之后，使用 apg_plan_mgmt.evolve_plan_baselines 函数根据其实际性能来比较计划。根据性能试验的结果，您可以将计划的状态从未批准更改为已批准或已拒绝。如果计划不满足您的要求，您可以改为考虑使用 apg_plan_mgmt.evolve_plan_baselines 函数来临时禁用计划。

批准更好的计划

以下示例演示如何使用 apg_plan_mgmt.evolve_plan_baselines 函数将托管计划的状态更改为已批准。

```
SELECT apg_plan_mgmt.evolve_plan_baselines (
   sql_hash,
   plan_hash,
   min_speedup_factor := 1.0,
   action := 'approve'
)
FROM apg_plan_mgmt.dba_plans WHERE status = 'Unapproved';
```

```
NOTICE:     rangequery (1,10000)
NOTICE:     Baseline   [ Planning time 0.761 ms, Execution time 13.261 ms]
NOTICE:     Baseline+1 [ Planning time 0.204 ms, Execution time 8.956 ms]
NOTICE:     Total time benefit: 4.862 ms, Execution time benefit: 4.305 ms
NOTICE:     Unapproved -> Approved
evolve_plan_baselines
----------------------
0
(1 row)
```

输出显示 rangequery 语句的性能报告，参数绑定 1 和 10,000。新的未批准计划 (Baseline +1) 比以前的已批准最佳计划 (Baseline) 要好。要确认新计划现在已 Approved，请查看 apg_plan_mgmt.dba_plans 视图。

```
SELECT sql_hash, plan_hash, status, enabled, stmt_name
FROM apg_plan_mgmt.dba_plans;
```

```
sql_hash  | plan_hash |  status  | enabled | stmt_name
------------+-----------+----------+---------+------------
1984047223 | 512153379 | Approved | t       | rangequery
1984047223 | 512284451 | Approved | t       | rangequery
(2 rows)
```

托管计划现在包含两个已批准计划，这些计划是语句的计划基线。您还可以直接调用 apg_plan_mgmt.set_plan_status 函数，以直接将计划的状态字段设置为 'Approved'、'Rejected'、'Unapproved' 或 'Preferred'。

拒绝或禁用速度较慢的计划

要拒绝或禁用计划，请将 'reject' 或 'disable' 作为操作参数传递给 apg_plan_mgmt.evolve_plan_baselines 函数。此示例禁用任何已捕获并且其速度比语句的最佳 Unapproved 计划慢至少 10% 的 Approved 计划。

```
SELECT apg_plan_mgmt.evolve_plan_baselines(
sql_hash,  -- The managed statement ID
plan_hash, -- The plan ID
1.1,       -- number of times faster the plan must be
'disable'  -- The action to take. This sets the enabled field to false.
)
FROM apg_plan_mgmt.dba_plans
WHERE status = 'Unapproved' AND   -- plan is Unapproved
origin = 'Automatic';       -- plan was auto-captured
```

您也可以直接将计划设置为已拒绝或已禁用。要直接将计划的已启用字段设置为 true 或 false，请调用 apg_plan_mgmt.set_plan_enabled 函数。要直接将计划的状态字段设置为 'Approved'、'Rejected'、'Unapproved' 或 'Preferred'，请调用 apg_plan_mgmt.set_plan_status 函数。

要删除无效和您认为会保持无效的计划，请使用 apg_plan_mgmt.validate_plans 函数。此函数可让您删除或禁用无效计划。有关更多信息，请参阅 验证计划。

使用 pg_hint_plan 修复计划

查询优化程序经过精心设计，用于查找所有语句的最优计划，大多数情况下优化程序可以找到较好的计划。但是，有时候您可能已知存在比优化程序所生成计划好得多的计划。使优化程序生成所需计划的两种建议方法包括使用 pg_hint_plan 扩展，或者在 PostgreSQL 中设置 Grand Unified Configuration (GUC) 变量：

- pg_hint_plan 扩展 – 使用 PostgreSQL 的 pg_hint_plan 扩展指定“提示”以修改计划程序的工作方式。要安装 pg_hint_plan 扩展并了解有关如何使用该扩展的更多信息，请参阅 pg_hint_plan 文档。
- GUC 变量 – 覆盖一个或多个成本模型参数或其他优化程序参数，例如 from_collapse_limit 或 GEQO_threshold。

在您使用这些技术之一来强制查询优化程序使用计划时，您还可以使用查询计划管理来捕获和强制使用新计划。

您可以使用 pg_hint_plan 扩展来更改联接顺序、联接方法或者 SQL 语句的访问路径。您使用具有特殊 pg_hint_plan 语法的 SQL 注释来修改优化程序如何创建计划。例如，假设问题 SQL 语句具有双向联接。

```
SELECT *
FROM t1, t2
WHERE t1.id = t2.id;
```

然后，假设优化程序选择联接顺序（t1、t2），而您知道联接顺序（t2、t1）速度更快。以下提示强制优化程序使用速度更快的联接顺序 (t2, t1)。包括 EXPLAIN，以使优化程序为 SQL 语句生成计划，但不运行语句。（未显示输出。）

```
/*+ Leading ((t2 t1)) */ EXPLAIN SELECT *
FROM t1, t2
WHERE t1.id = t2.id;
```

以下步骤显示如何使用 pg_hint_plan。

修改优化程序生成的计划并使用 pg_hint_plan 捕获计划

1. 启用手动捕获模式。

   ```
   SET apg_plan_mgmt.capture_plan_baselines = manual;
   ```

2. 指定感兴趣 SQL 语句的提示。

   ```
   /*+ Leading ((t2 t1)) */ EXPLAIN SELECT *
   FROM t1, t2
   WHERE t1.id = t2.id;
   ```

   在此语句运行后，优化程序在 apg_plan_mgmt.dba_plans 视图中捕获计划。捕获的计划不包括特殊 pg_hint_plan 注释语法，因为查询计划管理通过删除前导注释来标准化语句。

3. 使用 apg_plan_mgmt.dba_plans 视图查看托管计划。

   ```
   SELECT sql_hash, plan_hash, status, sql_text, plan_outline
   FROM apg_plan_mgmt.dba_plans;
   ```

4. 将计划的状态设置为 Preferred。这样做可以确保优化程序将选择运行它，而不是在最低成本计划未处于 Approved 或 Preferred 状态时从一组已批准计划中选择。

```
SELECT apg_plan_mgmt.set_plan_status(sql-hash, plan-hash, 'preferred' );
```

5. 禁用手动计划捕获并强制托管计划的使用。

```
SET apg_plan_mgmt.capture_plan_baselines = false;
SET apg_plan_mgmt.use_plan_baselines = true;
```

现在，当原始 SQL 语句运行时，优化程序将选择 Approved 或 Preferred 计划。如果最低成本计划不是 Approved 或 Preferred，则优化程序将选择 Preferred 计划。

删除 Aurora PostgreSQL 查询计划

删除未使用的执行计划或无效的计划。有关删除计划的更多信息，请参阅以下各节。

主题

- 删除计划
- 验证计划

删除计划

如果计划已超过一个月（具体来说，为 32天）未使用，则会自动删除。这是 apg_plan_mgmt.plan_retention_period 参数的原定设置。您可以将计划保留期更改为较长的时间段，也可以更改为从值 1 开始的较短时间段。通过从当前日期中减去 last_used 日期来计算自上次使用计划以来的天数。last_used 日期是优化程序选择计划作为最低成本计划或运行该计划的最近日期。计划的这一日期存储在 apg_plan_mgmt.dba_plans 视图中。

建议您删除长时间未使用或者无用的计划。每个计划都具有 last_used 日期，优化程序在每次执行某个计划或选择计划作为语句的最低成本计划时，都会更新该日期。检查上次 last_used 日期以确定可以安全删除的计划。

以下查询返回一个具有三列的表，其中包含有关计划总数、无法删除的计划数和成功删除的计划数的计数。此查询有一个嵌套查询示例，旨在说明如何使用 apg_plan_mgmt.delete_plan 函数来删除过去 31 天内未选为最低成本计划且其状态不为 Rejected 的所有计划。

```
SELECT (SELECT COUNT(*) from apg_plan_mgmt.dba_plans) total_plans,
       COUNT(*) FILTER (WHERE result = -1) failed_to_delete,
```

```
       COUNT(*) FILTER (WHERE result = 0) successfully_deleted
       FROM (
            SELECT apg_plan_mgmt.delete_plan(sql_hash, plan_hash) as result
            FROM apg_plan_mgmt.dba_plans
            WHERE last_used < (current_date - interval '31 days')
            AND status <> 'Rejected'
            ) as dba_plans ;
```

```
 total_plans | failed_to_delete | successfully_deleted
-------------+------------------+----------------------
           3 |                0 |                    2
```

有关更多信息，请参阅 apg_plan_mgmt.delete_plan。

要删除无效和您认为会保持无效的计划，请使用 apg_plan_mgmt.validate_plans 函数。此函数可让您删除或禁用无效计划。有关更多信息，请参阅 验证计划。

> **Important**
>
> 如果您未删除多余的计划，则最终可能会耗尽为查询计划管理留出的共享内存。要控制可供托管计划使用的内存量，请使用 apg_plan_mgmt.max_plans 参数。在自定义数据库参数组中设置此参数并重启数据库实例，让更改生效。有关更多信息，请参阅 apg_plan_mgmt.max_plans 参数。

验证计划

使用 apg_plan_mgmt.validate_plans 函数来删除或禁用无效的计划。

在删除了所依赖的对象（例如索引或表）时，托管计划会成为无效或过时。但是，如果重新创建了删除的对象，计划可能仅临时无效。如果某个无效计划以后会变成有效，建议您禁用无效的计划或者不执行任何操作，而不是删除它。

要查找并删除所有无效且在过去一周中未使用的计划，请使用 apg_plan_mgmt.validate_plans 函数，如下所示。

```
SELECT apg_plan_mgmt.validate_plans(sql_hash, plan_hash, 'delete')
FROM apg_plan_mgmt.dba_plans
```

```
WHERE last_used < (current_date - interval '7 days');
```

要直接启用或禁用计划，请使用 apg_plan_mgmt.set_plan_enabled 函数。

导出和导入 Aurora PostgreSQL 的托管式计划

您可以导出托管计划并将其导入到其他数据库实例。

导出托管计划

授权用户可以将 apg_plan_mgmt.plans 表的任意子集复制到其他表，然后使用 pg_dump 命令来保存。以下是示例。

```
CREATE TABLE plans_copy AS SELECT *
FROM apg_plan_mgmt.plans [ WHERE predicates ] ;
```

```
% pg_dump --table apg_plan_mgmt.plans_copy -Ft mysourcedatabase > plans_copy.tar
```

```
DROP TABLE apg_plan_mgmt.plans_copy;
```

导入托管计划

1. 将所导出的托管计划的 .tar 文件，复制到将要还原计划的系统中。
2. 使用 pg_restore 命令将 tar 文件复制到新表。

   ```
   % pg_restore --dbname mytargetdatabase -Ft plans_copy.tar
   ```

3. 将 plans_copy 表与 apg_plan_mgmt.plans 表合并，如以下示例所示。

   > **Note**
   >
   > 有些情况下，您可能从 apg_plan_mgmt 扩展的一个版本转储并还原到其他版本。在这些情况下，计划表中的列可能不同。如果是这样，明确命名列而不是使用 SELECT *。

   ```
   INSERT INTO apg_plan_mgmt.plans SELECT * FROM plans_copy
    ON CONFLICT ON CONSTRAINT plans_pkey
    DO UPDATE SET
   ```

```
status = EXCLUDED.status,
enabled = EXCLUDED.enabled,
-- Save the most recent last_used date
--
last_used = CASE WHEN EXCLUDED.last_used > plans.last_used
THEN EXCLUDED.last_used ELSE plans.last_used END,
-- Save statistics gathered by evolve_plan_baselines, if it ran:
--
estimated_startup_cost = EXCLUDED.estimated_startup_cost,
estimated_total_cost = EXCLUDED.estimated_total_cost,
planning_time_ms = EXCLUDED.planning_time_ms,
execution_time_ms = EXCLUDED.execution_time_ms,
total_time_benefit_ms = EXCLUDED.total_time_benefit_ms,
execution_time_benefit_ms = EXCLUDED.execution_time_benefit_ms;
```

4. 将托管计划重新加载到共享内存中，删除临时计划表。

```
SELECT apg_plan_mgmt.reload(); -- refresh shared memory
DROP TABLE plans_copy;
```

Aurora PostgreSQL 查询计划管理的参数参考

您可以使用本节中列出的参数设置 apg_plan_mgmt 扩展的首选项。这些参数可在自定义数据库集群参数以及与 Aurora PostgreSQL 数据库集群关联的数据库参数组中使用。这些参数控制查询计划管理功能的行为及其影响优化程序的方式。有关设置查询计划管理的信息，请参阅开启 Aurora PostgreSQL 查询计划管理。如果没有按照本节中的详细说明设置 apg_plan_mgmt 扩展，则更改以下参数将不起作用。有关修改参数的信息，请参阅在 Amazon Aurora 中修改数据库集群参数组中的参数和Amazon Aurora 数据库实例的数据库参数组。

参数

- apg_plan_mgmt.capture_plan_baselines
- apg_plan_mgmt.plan_capture_threshold
- apg_plan_mgmt.explain_hashes
- apg_plan_mgmt.log_plan_enforcement_result
- apg_plan_mgmt.max_databases
- apg_plan_mgmt.max_plans
- apg_plan_mgmt.plan_hash_version

- apg_plan_mgmt.plan_retention_period
- apg_plan_mgmt.unapproved_plan_execution_threshold
- apg_plan_mgmt.use_plan_baselines
- auto_explain.hashes

apg_plan_mgmt.capture_plan_baselines

捕获优化程序为每条 SQL 语句生成的查询执行计划，并将它们存储在 dba_plans 视图中。原定设置情况下，可存储的最大计划数为 10000（由 apg_plan_mgmt.max_plans 参数指定）。有关参考信息，请参阅 apg_plan_mgmt.max_plans。

您可以在自定义数据库集群参数组或自定义数据库参数组中设置此参数。更改此参数的值不需要重启。

默认	允许值	描述
off	自动	在会话级别或参数组中应用此设置，以捕获使用两次或更多次的计划。
	手动	在会话级别或参数组中应用此设置，以捕获使用一次或更多次的计划。
	off	关闭计划捕获。

有关更多信息，请参阅 捕获 Aurora PostgreSQL 执行计划。

apg_plan_mgmt.plan_capture_threshold

指定一个阈值，这样如果查询执行计划的总成本低于该阈值，便不会在 apg_plan_mgmt.dba_plans 视图中捕获该计划。

更改此参数的值不需要重启。

默认	允许值	描述
0	0 - 1.79769e+308	设置用于捕获计划的 apg_plan_mgmt 查询计划总执行成本的阈值。

有关更多信息，请参阅 在 dba_plans 视图中检查 Aurora PostgreSQL 查询计划。

apg_plan_mgmt.explain_hashes

指定 EXPLAIN [ANALYZE] 是否在其输出的结尾显示 sql_hash 和 plan_hash。更改此参数的值不需要重启。

默认	允许值	描述
0	0（关闭）	EXPLAIN 不显示 sql_hash 和 plan_hash（不带 hashes true 选项）。
	1（开启）	EXPLAIN 显示 sql_hash 和 plan_hash（不带 hashes true 选项）。

apg_plan_mgmt.log_plan_enforcement_result

指定是否必须记录结果，以查看 QPM 托管式计划是否得到正确使用。使用存储的通用计划时，日志文件中不会写入任何记录。更改此参数的值不需要重启。

默认	允许值	描述
none	none	未在日志文件中显示任何计划实施结果。
	on_error	仅当 QPM 无法使用托管式计划时，才在日志文件中显示计划实施结果。
	全部	在日志文件中显示所有计划实施结果，不论成功的还是失败的。

apg_plan_mgmt.max_databases

在您的 Aurora PostgreSQL 数据库集群的写入器实例上，指定可以使用查询计划管理的最大数据库数量。原定设置情况下，最多 10 个数据库可以使用查询计划管理。如果实例上的数据库数量超过 10 个，则可以更改此设置的值。要了解给定实例上有多少数据库，请使用 psql 连接到该实例。然后，使用 psql 元命令 \l 列出数据库。

更改此参数的值要求您重启实例才能使设置生效。

默认	允许值	描述
10	10-2147483647	实例上可以使用查询计划管理的最大数据库数量。

您可以在自定义数据库集群参数组或自定义数据库参数组中设置此参数。

apg_plan_mgmt.max_plans

设置查询计划管理器可以在 `apg_plan_mgmt.dba_plans` 视图中维持的 SQL 语句的最大数量。我们建议为所有的 Aurora PostgreSQL 版本将此参数设置为 `10000` 或更高的值。

您可以在自定义数据库集群参数组或自定义数据库参数组中设置此参数。更改此参数的值要求您重启实例才能使设置生效。

默认	允许值	描述
10000	10-2147483647	`apg_plan_mgmt.dba_plans` 视图中可存储的最大计划数。 Aurora PostgreSQL 版本 10 及更早版本的原定设置值为 1000。

有关更多信息，请参阅 在 dba_plans 视图中检查 Aurora PostgreSQL 查询计划。

apg_plan_mgmt.plan_hash_version

指定 plan_hash 计算旨在涵盖的使用案例。`apg_plan_mgmt.plan_hash_version` 的较高版本涵盖了较低版本的所有功能。例如，版本 3 涵盖了版本 2 支持的使用案例。

更改此参数的值后，必须调用 `apg_plan_mgmt.validate_plans('update_plan_hash')`。它使用安装的 apg_plan_mgmt 和 plans 表中的条目更新每个数据库中的 plan_hash 值。有关更多信息，请参阅 验证计划。

默认	允许值	描述
1	1	原定设置的 plan_hash 计算。
	2	已修改 plan_hash 计算以支持多架构。
	3	已修改 plan_hash 计算以支持多架构和分区表。

默认	允许值	描述
	4	已修改 plan_hash 计算以支持并行运算符和具体化节点。

apg_plan_mgmt.plan_retention_period

指定要在 apg_plan_mgmt.dba_plans 视图中保留计划的天数，此天数之后将自动删除计划。原定设置情况下，自上次使用计划（apg_plan_mgmt.dba_plans 视图中的 last_used 列）起 32 天后，计划将被删除。您可以将此设置更改为任意数字，1 及更多。

更改此参数的值要求您重启实例才能使设置生效。

默认	允许值	描述
32	1-2147483647	自上次使用计划后至删除计划的最大天数。

有关更多信息，请参阅 在 dba_plans 视图中检查 Aurora PostgreSQL 查询计划。

apg_plan_mgmt.unapproved_plan_execution_threshold

指定一个成本阈值，如果低于该阈值，优化器可以使用未批准的计划。默认情况下，该阈值为 0，所以优化器不运行未批准的计划。将此参数设置为极低的成本阈值（例如 100）可以避免普通计划的计划实施开销。您也可以使用反应式计划管理将此参数设置为一个非常大的值，比如 10000000。这样，优化器就可以在没有计划实施开销的情况下使用所有选定的计划。但是，当发现错误的计划时，您可以手动将其标记为“已拒绝”，这样下次就不会使用该计划。

此参数的值表示运行给定计划的成本估计。如果未批准的计划低于该估计成本，则优化程序会将其用于 SQL 语句。您可以在 dba_plans 视图中查看已捕获的计划及其状态（已批准、未批准）。要了解更多信息，请参阅在 dba_plans 视图中检查 Aurora PostgreSQL 查询计划。

更改此参数的值不需要重启。

默认	允许值	描述
0	0-2147483647	估计的计划开销，如果低于该值，则将使用未批准的计划。

有关更多信息，请参阅 使用 Aurora PostgreSQL 托管式计划。

apg_plan_mgmt.use_plan_baselines

指定优化程序应使用捕获的并存储在 apg_plan_mgmt.dba_plans 视图中的已批准计划之一。原定设置情况下，此参数处于关闭状态（false），这会导致优化程序使用其生成的最低成本计划，而无需进行任何进一步评估。启用此参数（将其设置为 true）会强制优化程序从其计划基准中为该语句选择查询执行计划。有关更多信息，请参阅 使用 Aurora PostgreSQL 托管式计划。要查找详细说明此过程的图像，请参阅优化程序如何选择要运行的计划。

您可以在自定义数据库集群参数组或自定义数据库参数组中设置此参数。更改此参数的值不需要重启。

默认	允许值	描述
false	true	使用 apg_plan_mgmt.dba_plans 中的“已批准”、“首选”或“未批准”计划。如果这些计划都不符合优化程序的所有评估标准，则它可以使用自己生成的最低成本计划。有关更多信息，请参阅 优化程序如何选择要运行的计划。
	false	使用优化程序生成的最低成本计划。

您可以根据需要，评估不同的已捕获计划的响应时间以及变更计划状态。有关更多信息，请参阅 改进 Aurora PostgreSQL 查询计划。

auto_explain.hashes

指定 auto_explain 输出是否显示 sql_hash 和 plan_hash。更改此参数的值不需要重启。

默认	允许值	描述
0（关闭）	0（关闭）	auto_explain 结果不显示 sql_hash 和 plan_hash 。
	1（开启）	auto_explain 结果显示 sql_hash 和 plan_hash 。

Aurora PostgreSQL 查询计划管理的函数参考

apg_plan_mgmt 扩展提供以下函数。

函数

- apg_plan_mgmt.copy_outline
- apg_plan_mgmt.delete_plan
- apg_plan_mgmt.evolve_plan_baselines
- apg_plan_mgmt.get_explain_plan
- apg_plan_mgmt.plan_last_used
- apg_plan_mgmt.reload
- apg_plan_mgmt.set_plan_enabled
- apg_plan_mgmt.set_plan_status
- apg_plan_mgmt.update_plans_last_used
- apg_plan_mgmt.validate_plans

apg_plan_mgmt.copy_outline

将给定的 SQL 计划哈希和计划大纲复制到目标 SQL 计划哈希和大纲，从而覆盖目标的计划哈希和大纲。此功能在 apg_plan_mgmt 2.3 及更高版本中可用。

语法

```
apg_plan_mgmt.copy_outline(
    source_sql_hash,
    source_plan_hash,
    target_sql_hash,
    target_plan_hash,
    force_update_target_plan_hash
)
```

返回值

复制成功时返回 0。对于无效输入引发异常。

参数

参数	描述
source_sql_hash	要复制到目标查询的与 plan_hash 关联的 sql_hash ID。

参数	描述
source_plan_hash	要复制到目标查询的 plan_hash ID。
target_sql_hash	要使用源计划哈希和大纲更新的查询的 sql_hash ID。
target_plan_hash	要使用源计划哈希和大纲更新的查询的 plan_hash ID。
force_update_target_plan_hash	（可选）即使源计划对 target_sql_hash 不可重现，查询的 target_plan_hash ID 也会更新。设置为 True 时，该函数可用于在关系名称和列一致的架构之间复制计划。

使用说明

此函数允许您将使用提示的计划哈希和计划大纲复制到其他类似语句，从而无需在目标语句中每次出现时都使用内联提示语句。如果更新的目标查询导致计划无效，则此函数会引发错误并回滚尝试的更新。

apg_plan_mgmt.delete_plan

删除托管计划。

语法

```
apg_plan_mgmt.delete_plan(
    sql_hash,
    plan_hash
)
```

返回值

如果删除成功则返回 0，如果删除失败则返回 -1。

参数

参数	描述
sql_hash	计划的托管 SQL 语句的 sql_hash ID。

参数	描述
plan_hash	托管计划的 plan_hash ID。

apg_plan_mgmt.evolve_plan_baselines

验证已经批准的计划速度是否更快，或者查询优化程序确定作为最低成本计划的计划是否速度更快。

语法

```
apg_plan_mgmt.evolve_plan_baselines(
    sql_hash,
    plan_hash,
    min_speedup_factor,
    action
)
```

返回值

速度比最佳已批准计划要慢的计划数量。

参数

参数	描述
sql_hash	计划的托管 SQL 语句的 sql_hash ID。
plan_hash	托管计划的 plan_hash ID。使用 NULL 表示所有计划具有相同的 sql_hash ID 值。
min_speedup_factor	最低加速系数可能是计划必须比已经批准的最佳计划要快的倍数，达到此数字才能批准计划。或者，此系数可以是比计划必须达到才能拒绝或禁用它的速度慢的倍数。 这是正浮点数值。
action	函数执行的操作。包括下列有效值。大小写没有影响。 • 'disable' – 禁用不符合最低加速系数的各个匹配计划。

参数	描述
	• 'approve' – 启用满足最低加速系数的各个匹配计划并将其状态设置为 approved。 • 'reject' – 对于不满足最低加速系数的各个匹配计划，将其状态设置为 rejected。 • NULL – 函数仅返回由于不满足最低加速系数而没有性能优势的计划数。

使用说明

根据规划加上执行时间是否比最佳已批准计划速度快（达到您设置的系数），将特定计划设置为已批准、已拒绝或已禁用。操作参数可以设置为 'approve' 或 'reject' 以自动批准或拒绝满足性能标准的计划。此外，可以将其设置为 ''（空字符串）以进行性能试验并生成报告，但不采取操作。

您可以避免无目标地为 apg_plan_mgmt.evolve_plan_baselines 函数近期在其中运行的计划重新运行此函数。要这样做，将计划限制为仅近期创建的未批准计划。此外，您可以避免在任何具有近期 apg_plan_mgmt.evolve_plan_baselines 时间戳的已批准计划上运行 last_verified 函数。

开展性能试验，以将各个计划的规划加上执行时间，相对于基线中的其他计划进行比较。部分情况下，某个语句只有一个计划并且该计划已批准。在这种情况下，将计划的规划加上执行时间，与不使用计划时的规划加上执行时间进行比较。

各个计划增加的好处（或坏处）记录在 apg_plan_mgmt.dba_plans 视图的 total_time_benefit_ms 列中。当此值为正数时，有可衡量的性能优势，可以将此计划包括在基线内。

除了收集各个候选计划的规划和执行时间之外，使用 last_verified 来更新 apg_plan_mgmt.dba_plans 视图的 current_timestamp 列。last_verified 时间戳可用于避免对某个最近已经验证了其性能的计划，再次运行此函数。

apg_plan_mgmt.get_explain_plan

为指定的 SQL 语句生成 EXPLAIN 语句文本。

语法

```
apg_plan_mgmt.get_explain_plan(
    sql_hash,
```

```
    plan_hash,
    [explainOptionList]
)
```

返回值

返回指定 SQL 语句的运行时统计数据。使用不带 explainOptionList 以返回一个简单的 EXPLAIN 计划。

参数

参数	描述
sql_hash	计划的托管 SQL 语句的 sql_hash ID。
plan_hash	托管计划的 plan_hash ID。
explainOptionList	逗号分隔的解释选项列表。有效值包括 'analyze' 、'verbose' 、'buffers' 、'hashes' 和 'format json'。如果 explainOptionList 列表为 NULL 或空字符串 ('')，此函数会生成 EXPLAIN 语句，不带任何统计数据。

使用说明

对于 explainOptionList，您可以使用与 EXPLAIN 语句一起使用的任何相同选项。Aurora PostgreSQL 优化程序将您为 EXPLAIN 语句提供的选项列表连接在一起。

apg_plan_mgmt.plan_last_used

从共享内存返回指定计划的 last_used 日期。

> **Note**
>
> 数据库集群中的主数据库实例上共享内存中的值始终为最新值。该值仅定期刷新到 apg_plan_mgmt.dba_plans 视图的 last_used 列中。

语法

```
apg_plan_mgmt.plan_last_used(
    sql_hash,
    plan_hash
)
```

返回值

返回 last_used 日期。

参数

参数	描述
sql_hash	计划的托管 SQL 语句的 sql_hash ID。
plan_hash	托管计划的 plan_hash ID。

apg_plan_mgmt.reload

将计划从 apg_plan_mgmt.dba_plans 视图重新加载到共享内存中。

语法

```
apg_plan_mgmt.reload()
```

返回值

无。

参数

无。

使用说明

对于以下情况，调用 reload：

- 使用它来立即刷新只读副本的共享内存，而不是等待新计划传播到副本。
- 在导入托管计划后使用它。

apg_plan_mgmt.set_plan_enabled

启用或禁用托管计划。

语法

```
apg_plan_mgmt.set_plan_enabled(
    sql_hash,
    plan_hash,
    [true | false]
)
```

返回值

如果设置成功则返回 0，如果设置失败则返回 -1。

参数

参数	描述
sql_hash	计划的托管 SQL 语句的 sql_hash ID。
plan_hash	托管计划的 plan_hash ID。
enabled	布尔值 true 或 false： • 值为 true 启用计划。 • 值 false 禁用计划。

apg_plan_mgmt.set_plan_status

将托管计划的状态设置为 Approved、Unapproved、Rejected 或 Preferred。

语法

```
apg_plan_mgmt.set_plan_status(
    sql_hash,
    plan_hash,
    status
)
```

返回值

如果设置成功则返回 0，如果设置失败则返回 -1。

参数

参数	描述
sql_hash	计划的托管 SQL 语句的 sql_hash ID。
plan_hash	托管计划的 plan_hash ID。
status	具有以下值之一的字符串： • 'Approved' • 'Unapproved' • 'Rejected' • 'Preferred' 您使用的大小写并不重要，但 apg_plan_mgmt.dba_plans 视图中的状态值设置为首字母大写。有关这些值的更多信息，请参阅 status中的 Aurora PostgreSQL 兼容版的 apg_plan_mgmt.dba_plans 视图参考。

apg_plan_mgmt.update_plans_last_used

立即使用存储在共享内存中的 last_used 日期更新计划表。

语法

```
apg_plan_mgmt.update_plans_last_used()
```

返回值

无。

参数

无。

使用说明

调用 update_plans_last_used 以确保针对 dba_plans.last_used 列的查询使用最新的信息。如果 last_used 日期不会立即更新，后台进程会使用 last_used 日期每小时更新一次计划表（预设情况下）。

例如，如果具有某个特定 sql_hash 的语句开始运行缓慢，您可以确定自性能下降开始以来为该语句执行了哪些计划。为此，首先将共享内存中的数据刷新到磁盘，使 last_used 日期保持最新，然后查询性能下降的语句的 sql_hash 的所有计划。在查询中，请确保 last_used 日期大于或等于性能下降的开始日期。该查询将标识可能对性能下降负责的计划或一组计划。您可以使用 explainOptionList 被设置为 verbose, hashes 的 apg_plan_mgmt.get_explain_plan。您还可以使用 apg_plan_mgmt.evolve_plan_baselines 分析计划以及任何可能表现更好的替代计划。

update_plans_last_used 函数仅对数据库集群的主数据库实例具有影响。

apg_plan_mgmt.validate_plans

验证优化程序仍可重新创建计划。优化程序将验证 Approved、Unapproved 和 Preferred 计划，而无论是启用还是禁用了此计划。不验证 Rejected 计划。（可选）您可以使用 apg_plan_mgmt.validate_plans 函数来删除或禁用无效计划。

语法

```
apg_plan_mgmt.validate_plans(
    sql_hash,
    plan_hash,
    action)

apg_plan_mgmt.validate_plans(
```

```
    action)
```

返回值

无效计划的数量。

参数

参数	描述
sql_hash	计划的托管 SQL 语句的 sql_hash ID。
plan_hash	托管计划的 plan_hash ID。使用 NULL 表示同一个 sql_hash ID 值的所有计划。
action	函数为无效计划执行的操作。有效字符串值如下所示。大小写没有影响。 • 'disable' – 禁用各个无效计划。 • 'delete' – 删除各个无效计划。 • 'update_plan_hash' – 更新不能准确重现的计划的 plan_hash ID。该参数还允许您通过重新编写 SQL 来修复计划。然后，您可以将更好的计划注册为初始 SQL 的 Approved 计划。 • NULL – 函数仅返回无效计划数。不执行其他操作。 • " – 空字符串将生成一条消息，指示有效计划和无效计划的数量。 任何其他值作为空字符串处理。

使用说明

使用格式 validate_plans(action) 来验证整个 apg_plan_mgmt.dba_plans 视图中，所有托管语句的所有托管计划。

使用格式 validate_plans(sql_hash, plan_hash, action)，为使用 plan_hash 指定的托管语句，验证以 sql_hash 指定的托管计划。

使用格式 validate_plans(sql_hash, NULL, action) 来验证使用 sql_hash 指定的托管语句的所有托管计划。

Aurora PostgreSQL 兼容版的 apg_plan_mgmt.dba_plans 视图参考

apg_plan_mgmt.dba_plans 视图中的计划信息列包括下列内容。

dba_plans 列	描述
cardinality_error	对估计基数与实际基数之间错误的测量。基数是计划要处理的表行数。如果基数错误较大，则会增加计划并非最优的可能性。此列由 apg_plan_mgmt.evolve_plan_baselines 函数填充。
compatibility_level	此参数显示上次验证查询计划的时间。在 Aurora PostgreSQL 版本 12.19、13.15、14.12、15.7、16.3 及更高版本中，它显示 Aurora 版本号。对于较早的版本，它会显示功能特定的版本号。 **Note** 将此参数值保持为其默认设置。Aurora PostgreSQL 会自动设置和更新此值。
created_by	创建计划的已通过身份验证用户 (session_user)。
enabled	计划已启用还是已禁用的指示符。默认情况下启用所有计划。您可以禁用计划以防止由优化程序使用。要修改此值，请使用 apg_plan_mgmt.set_plan_enabled 函数。
environment_variables	优化程序在捕获计划时覆盖的 PostgreSQL Grand Unified Configuration (GUC) 参数和值。
estimated_startup_cost	在优化程序提供表行之前估算的优化程序设置成本。
estimated_total_cost	估算的优化程序提供最终表行的成本。
execution_time_benefit_ms	启用计划带来的执行时间效益，以毫秒为单位。此列由 apg_plan_mgmt.evolve_plan_baselines 函数填充。
execution_time_ms	计划将运行的估计时间，以毫秒为单位。此列由 apg_plan_mgmt.evolve_plan_baselines 函数填充。

dba_plans 列	描述
has_side_effects	此值指示 SQL 语句是数据操作语言 (DML) 语句还是包含 VOLATILE 函数的 SELECT 语句。
last_used	此值在计划执行或者计划是查询优化程序的最低成本计划时，更新为当前日期。此值存储在共享内存中，定期刷新到磁盘。要获取最新的值，请通过调用函数 apg_plan_mgmt.plan_last_used(sql_hash, plan_hash) 而不是读取 last_used 值，从共享内存中读取日期。有关其他信息，请参阅 apg_plan_mgmt.plan_retention_period 参数。
last_validated	验证计划可以由 apg_plan_mgmt.validate_plans 函数或 apg_plan_mgmt.evolve_plan_baselines 函数重新创建时的最近日期和时间。
last_verified	apg_plan_mgmt.evolve_plan_baselines 函数验证计划是指定参数的最佳性能计划的最近日期和时间。
origin	如何通过 apg_plan_mgmt.capture_plan_baselines 参数捕获的计划。包括下列有效值： M – 通过手动计划捕获来捕获的计划。 A – 通过自动计划捕获来捕获的计划。
param_list	在语句是预编译语句时，传递到语句的参数值。
plan_created	计划的创建日期和时间。
plan_hash	计划标识符。plan_hash 和 sql_hash 的组合唯一标识特定计划。
plan_outline	计划的表示，用于重新创建实际的执行计划，与数据库无关。树中的运算符对应于 EXPLAIN 输出中显示的运算符。
planning_time_ms	运行计划程序的实际时间，以毫秒为单位。此列由 apg_plan_mgmt.evolve_plan_baselines 函数填充。

dba_plans 列	描述
queryId	语句哈希，由 pg_stat_statements 扩展计算。这不是稳定或独立于数据库的标识符，因为它依赖于对象标识符 (OID)。捕获查询计划时，如果 compute_query_id 为 off，则该值将为 0。
sql_hash	SQL 语句文本的散列值，删除了文本以标准化。
sql_text	SQL 语句的完整文本。
status	计划的状态，确定优化程序如何使用计划。包括下列有效值。 • Approved – 可用的计划，优化程序可以选择该计划来运行。优化程序运行托管语句的已批准计划集（基线）中最低成本的计划。要将计划重置为已批准，请使用 apg_plan_mgmt.evolve_plan_baselines 函数。 • Unapproved – 您尚未验证以供使用的已捕获计划。有关更多信息，请参阅“评估计划性能”。 • Rejected – 优化程序不会使用的计划。有关更多信息，请参阅“拒绝或禁用速度较慢的计划”。 • Preferred – 您已确定作为首选计划的计划，该计划用于托管语句。 如果优化程序的最低成本计划不是已批准计划或首选计划，您可以减少计划实施开销。为此，请建立已审批计划 Preferred 的子集。当优化程序的最低成本不是 Approved 计划时，将在 Preferred 计划之前选择 Approved 计划。 要将计划重置为 Preferred ，请使用 apg_plan_mgmt.set_plan_status 函数。
stmt_name	PREPARE 语句内 SQL 语句的名称。对于未命名的预编译语句，此值是为字符串。对于非预编译语句，此值是为 NULL。

dba_plans 列	描述
total_time_benefit_ms	启用此计划带来的总时间效益，以毫秒为单位。此值同时考虑到规划时间和执行时间。 如果此值为负数，则启用此计划有负面效果。此列由 apg_plan_mgmt.evolve_plan_baselines 函数填充。

查询计划管理高级功能

以下是有关 Aurora PostgreSQL 查询计划管理（QPM）高级功能的信息：

主题

- 在副本中捕获 Aurora PostgreSQL 执行计划
- 支持表分区

在副本中捕获 Aurora PostgreSQL 执行计划

QPM（查询计划管理）允许您捕获由 Aurora 副本生成的查询计划并将其存储在 Aurora 数据库集群的主数据库实例上。您可以从所有 Aurora 副本中收集查询计划，并在主实例的中央永久表中维护一组最佳计划。然后，您可以在需要时将这些计划应用于其他副本。这将有助于您保持执行计划的稳定性，并跨数据库集群和引擎版本提高查询性能。

主题

- 先决条件
- 管理 Aurora 副本的计划捕获
- 故障排除

先决条件

在 Aurora 副本中开启 **capture_plan_baselines parameter** - 将 capture_plan_baselines 参数设置为自动或手动在 Aurora 副本中捕获计划。有关更多信息，请参阅 apg_plan_mgmt.capture_plan_baselines。

安装 postgres_fdw 扩展 - 必须安装 postgres_fdw 外部数据包装器扩展才能在 Aurora 副本中捕获计划。要安装扩展，请在每个数据库中运行以下命令。

```
postgres=> CREATE EXTENSION IF NOT EXISTS postgres_fw;
```

管理 Aurora 副本的计划捕获

开启 Aurora 副本的计划捕获

您必须具有 rds_superuser 权限才能在 Aurora 副本中创建或删除计划捕获。有关用户角色和权限的更多信息，请参阅了解 PostgreSQL 角色和权限。

要捕获计划，请在写入器数据库实例中调用 apg_plan_mgmt.create_replica_plan_capture 函数，如下所示：

```
postgres=> CALL apg_plan_mgmt.create_replica_plan_capture('endpoint', 'password');
```

- 端点：Aurora Global Database 写入器端点或 cluster_endpoint 为 Aurora 副本中的计划捕获提供失效转移支持。

  有关 Aurora Global Database 写入器端点的更多信息，请参阅查看 Amazon Aurora Global Database 的端点。

  有关集群端点的更多信息，请参阅 Amazon Aurora 的集群端点。
- password - 我们建议您在创建密码时遵循以下准则以增强安全性：
  - 必须至少包含 8 个字符。
  - 必须包含至少一个大写字母、一个小写字母和一个数字。
  - 必须至少包含一个特殊字符（?、!、#、<、>、* 等等）。

> **Note**
>
> 如果您更改了端点、密码或端口号，则必须使用端点和密码再次运行 apg_plan_mgmt.create_replica_plan_capture()，才能重新初始化计划捕获。否则，从 Aurora 副本中捕获计划将失败。

关闭 Aurora 副本的计划捕获

您可以通过在参数组中将 Aurora 副本中的参数 capture_plan_baselines 设置为 off，来将其关闭。

删除 Aurora 副本的计划捕获

您可以在 Aurora 副本中完全删除计划捕获，但要确保在删除之前执行相关操作。要删除计划捕获，请调用 `apg_plan_mgmt.remove_replica_plan_capture`，如下所示：

```
postgres=> CALL apg_plan_mgmt.remove_replica_plan_capture();
```

您必须再次调用 apg_plan_mgmt.create_replica_plan_capture()，才能使用端点和密码在 Aurora 副本中开启计划捕获。

故障排除

下文介绍了 Aurora 副本中未按预期捕获计划的排查思路和解决方法。

- 参数设置 - 检查 `capture_plan_baselines` 参数是否设置为正确的值以开启计划捕获。
- **`postgres_fdw` 扩展已安装** - 使用以下查询来检查 `postgres_fdw` 是否已安装。

  ```
  postgres=> SELECT * FROM pg_extension WHERE extname = 'postgres_fdw'
  ```

- create_replica_plan_capture() 已调用 - 使用以下命令检查用户映射是否退出。否则，请调用 `create_replica_plan_capture()` 以初始化该功能。

  ```
  postgres=> SELECT * FROM pg_foreign_server WHERE srvname =
   'apg_plan_mgmt_writer_foreign_server';
  ```

- 集群端点和端口号：检查端点和端口号是否合适。如果这些值不正确，则不会显示任何错误消息。

  使用以下命令验证是否在 create() 中使用了终端节点，并检查它位于哪个数据库中：

  ```
  postgres=> SELECT srvoptions FROM pg_foreign_server WHERE srvname =
   'apg_plan_mgmt_writer_foreign_server';
  ```

- reload() - 在 Aurora 副本中调用 apg_plan_mgmt.delete_plan() 之后，必须调用 apg_plan_mgmt.reload() 才能使删除函数生效。这将确保更改成功实施。
- Password - 您必须按照上述准则在 create_replica_plan_capture() 中输入密码。否则，您将会收到错误消息。有关更多信息，请参阅管理 Aurora 副本的计划捕获。使用其他符合要求的密码。
- 跨区域连接 - Aurora 全局数据库还支持 Aurora 副本中的计划捕获，其中写入器实例和 Aurora 副本可以位于不同的区域。确保在失效转移或切换事件发生后，使用 Aurora Global Database 写入器端点来保持连接。有关 Aurora Global Database 端点的更多信息，请参阅查看 Amazon Aurora Global

Database 的端点。写入器实例和跨区域副本必须能够使用 VPC 对等进行通信。有关更多信息，请参阅 VPC 对等。如果发生跨区域故障转移，则必须将终端节点重新配置为新的主数据库集群终端节点。

> **Note**
>
> 使用集群端点而不是 Aurora Global Database 写入器端点时，您将需要在执行全局失效转移或切换操作后更新集群端点。

支持表分区

Aurora PostgreSQL 查询计划管理（QPM）在以下版本中支持声明式表分区：

- 15.3 及更高的 15 版本
- 14.8 及更高的 14 版本
- 13.11 及更高的 13 版本

有关更多信息，请参阅表分区。

主题

- 设置表分区
- 捕获表分区的计划
- 强制执行表分区计划
- 命名约定

设置表分区

要在 Aurora PostgreSQL QPM 中设置表分区，请执行以下操作：

1. 在数据库集群参数组中将 `apg_plan_mgmt.plan_hash_version` 设置为 3 或更多。
2. 导航到使用查询计划管理且在 `apg_plan_mgmt.dba_plans` 视图中具有条目的数据库。
3. 调用 `apg_plan_mgmt.validate_plans('update_plan_hash')` 以更新 plans 表中的 `plan_hash` 值。
4. 对所有启用了查询计划管理且在 `apg_plan_mgmt.dba_plans` 视图中具有条目的数据库重复步骤 2-3。

有关这些参数的更多信息，请参阅Aurora PostgreSQL 查询计划管理的参数参考。

捕获表分区的计划

在 QPM 中，不同的计划以其 plan_hash 值来区分。要了解 plan_hash 如何变化，必须先了解类似的计划。

在 Append 节点级别累积的访问方法、去掉数字的索引名称和去掉数字的分区名称的组合必须是常量，才能将计划视为相同。在计划中访问的特定分区并不重要。在以下示例中，创建了一个包含 4 个分区的表 tbl_a。

```
postgres=>create table tbl_a(i int, j int, k int, l int, m int) partition by range(i);
CREATE TABLE
postgres=>create table tbl_a1 partition of tbl_a for values from (0) to (1000);
CREATE TABLE
postgres=>create table tbl_a2 partition of tbl_a for values from (1001) to (2000);
CREATE TABLE
postgres=>create table tbl_a3 partition of tbl_a for values from (2001) to (3000);
CREATE TABLE
postgres=>create table tbl_a4 partition of tbl_a for values from (3001) to (4000);
CREATE TABLE
postgres=>create index t_i on tbl_a using btree (i);
CREATE INDEX
postgres=>create index t_j on tbl_a using btree (j);
CREATE INDEX
postgres=>create index t_k on tbl_a using btree (k);
CREATE INDEX
```

以下计划被认为是相同的，因为无论查询查找的分区数量是多少，都使用单一扫描方法来扫描 tbl_a。

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 999 and j < 9910 and k > 50;

                        QUERY PLAN
-----------------------------------------------------------------------
Seq Scan on tbl_a1 tbl_a
    Filter: ((i >= 990) AND (i <= 999) AND (j < 9910) AND (k > 50))
SQL Hash: 1553185667, Plan Hash: -694232056
(3 rows)
```

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 1100 and j < 9910 and k > 50;

                        QUERY PLAN
------------------------------------------------------------------
Append
    ->  Seq Scan on tbl_a1 tbl_a_1
            Filter: ((i >= 990) AND (i <= 1100) AND (j < 9910) AND (k > 50))
    ->  Seq Scan on tbl_a2 tbl_a_2
            Filter: ((i >= 990) AND (i <= 1100) AND (j < 9910) AND (k > 50))
    SQL Hash: 1553185667, Plan Hash: -694232056
    (6 rows)
```

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 2100 and j < 9910 and k > 50;

                QUERY PLAN
-------------------------------------------------------------------------
 Append
   ->  Seq Scan on tbl_a1 tbl_a_1
         Filter: ((i >= 990) AND (i <= 2100) AND (j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 2100) AND (j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a3 tbl_a_3
         Filter: ((i >= 990) AND (i <= 2100) AND (j < 9910) AND (k > 50))
 SQL Hash: 1553185667, Plan Hash: -694232056
(8 rows)
```

以下 3 个计划也被认为是相同的，因为在父级别，访问方法、去除数字的索引名称和去除数字的分区名称是 SeqScan tbl_a、IndexScan (i_idx) tbl_a。

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 1100 and j < 9910 and k > 50;

                                QUERY PLAN
-------------------------------------------------------------------------
 Append
   ->  Seq Scan on tbl_a1 tbl_a_1
         Filter: ((i >= 990) AND (i <= 1100) AND (j < 9910) AND (k > 50))
   ->  Index Scan using tbl_a2_i_idx on tbl_a2 tbl_a_2
         Index Cond: ((i >= 990) AND (i <= 1100))
         Filter: ((j < 9910) AND (k > 50))
```

```
 SQL Hash: 1553185667, Plan Hash: -993736942
(7 rows)
```

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 2100 and j < 9910 and k > 50;

                                QUERY PLAN
---------------------------------------------------------------------------
 Append
   ->  Index Scan using tbl_a1_i_idx on tbl_a1 tbl_a_1
         Index Cond: ((i >= 990) AND (i <= 2100))
         Filter: ((j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 2100) AND (j < 9910) AND (k > 50))
   ->  Index Scan using tbl_a3_i_idx on tbl_a3 tbl_a_3
         Index Cond: ((i >= 990) AND (i <= 2100))
         Filter: ((j < 9910) AND (k > 50))
 SQL Hash: 1553185667, Plan Hash: -993736942
(10 rows)
```

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 3100 and j < 9910 and k > 50;

                                QUERY PLAN
---------------------------------------------------------------------------
 Append
   ->  Seq Scan on tbl_a1 tbl_a_1
         Filter: ((i >= 990) AND (i <= 3100) AND (j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 3100) AND (j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a3 tbl_a_3
         Filter: ((i >= 990) AND (i <= 3100) AND (j < 9910) AND (k > 50))
   ->  Index Scan using tbl_a4_i_idx on tbl_a4 tbl_a_4
         Index Cond: ((i >= 990) AND (i <= 3100))
         Filter: ((j < 9910) AND (k > 50))
 SQL Hash: 1553185667, Plan Hash: -993736942
(11 rows)
```

无论子分区中出现顺序和次数有何不同，在上述每种计划的父级别，访问方法、去除数字的索引名称和去除数字的分区名称都是常量。

但是，如果满足以下任何条件，则计划将被视为不同：

- 计划中使用了任何其他访问方法。

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between
 990 and 2100 and j < 9910 and k > 50;

                                QUERY PLAN
---------------------------------------------------------------------------
 Append
   ->  Seq Scan on tbl_a1 tbl_a_1
         Filter: ((i >= 990) AND (i <= 2100) AND (j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 2100) AND (j < 9910) AND (k > 50))
   ->  Bitmap Heap Scan on tbl_a3 tbl_a_3
         Recheck Cond: ((i >= 990) AND (i <= 2100))
         Filter: ((j < 9910) AND (k > 50))
         ->  Bitmap Index Scan on tbl_a3_i_idx
               Index Cond: ((i >= 990) AND (i <= 2100))
 SQL Hash: 1553185667, Plan Hash: 1134525070
(11 rows)
```

- 计划中有任何访问方法不再使用。

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between
 990 and 1100 and j < 9910 and k > 50;

                                QUERY PLAN
---------------------------------------------------------------------------
 Append
   ->  Seq Scan on tbl_a1 tbl_a_1
         Filter: ((i >= 990) AND (i <= 1100) AND (j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 1100) AND (j < 9910) AND (k > 50))
 SQL Hash: 1553185667, Plan Hash: -694232056
(6 rows)
```

- 与索引方法关联的索引已更改。

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between
 990 and 1100 and j < 9910 and k > 50;

                               QUERY PLAN
---------------------------------------------------------------------------
 Append
```

```
   ->  Seq Scan on tbl_a1 tbl_a_1
         Filter: ((i >= 990) AND (i <= 1100) AND (j < 9910) AND (k > 50))
   ->  Index Scan using tbl_a2_j_idx on tbl_a2 tbl_a_2
         Index Cond: (j < 9910)
         Filter: ((i >= 990) AND (i <= 1100) AND (k > 50))
 SQL Hash: 1553185667, Plan Hash: -993343726
(7 rows)
```

强制执行表分区计划

已批准的分区表计划通过位置对应关系来强制执行。这些计划并不特定于分区，可以在原始查询中引用的计划以外的分区上强制执行。对于访问的分区数量与原始批准的概况数量不同的查询，也可以强制执行计划。

例如，如果批准的概况适用于以下计划：

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 2100 and j < 9910 and k > 50;

                                QUERY PLAN
-------------------------------------------------------------------------
 Append
   ->  Index Scan using tbl_a1_i_idx on tbl_a1 tbl_a_1
         Index Cond: ((i >= 990) AND (i <= 2100))
         Filter: ((j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 2100) AND (j < 9910) AND (k > 50))
   ->  Index Scan using tbl_a3_i_idx on tbl_a3 tbl_a_3
         Index Cond: ((i >= 990) AND (i <= 2100))
         Filter: ((j < 9910) AND (k > 50))
 SQL Hash: 1553185667, Plan Hash: -993736942
(10 rows)
```

则也可以在引用 2 个、4 个或更多分区的 SQL 查询上强制执行此计划。对于 2 个和 4 个分区访问，这些场景可能会产生以下可能的计划：

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 1100 and j < 9910 and k > 50;

                                QUERY PLAN
-------------------------------------------------------------------------------
 Append
```

```
   ->  Index Scan using tbl_a1_i_idx on tbl_a1 tbl_a_1
         Index Cond: ((i >= 990) AND (i <= 1100))
         Filter: ((j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 1100) AND (j < 9910) AND (k > 50))
 Note: An Approved plan was used instead of the minimum cost plan.
 SQL Hash: 1553185667, Plan Hash: -993736942, Minimum Cost Plan Hash: -1873216041
(8 rows)
```

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 3100 and j < 9910 and k > 50;

                                QUERY PLAN
-------------------------------------------------------------------------
 Append
   ->  Index Scan using tbl_a1_i_idx on tbl_a1 tbl_a_1
         Index Cond: ((i >= 990) AND (i <= 3100))
         Filter: ((j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 3100) AND (j < 9910) AND (k > 50))
   ->  Index Scan using tbl_a3_i_idx on tbl_a3 tbl_a_3
         Index Cond: ((i >= 990) AND (i <= 3100))
         Filter: ((j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a4 tbl_a_4
         Filter: ((i >= 990) AND (i <= 3100) AND (j < 9910) AND (k > 50))
 Note: An Approved plan was used instead of the minimum cost plan.
 SQL Hash: 1553185667, Plan Hash: -993736942, Minimum Cost Plan Hash: -1873216041
(12 rows)
```

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 3100 and j < 9910 and k > 50;

                                QUERY PLAN
-------------------------------------------------------------------------------
 Append
   ->  Index Scan using tbl_a1_i_idx on tbl_a1 tbl_a_1
         Index Cond: ((i >= 990) AND (i <= 3100))
         Filter: ((j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 3100) AND (j < 9910) AND (k > 50))
   ->  Index Scan using tbl_a3_i_idx on tbl_a3 tbl_a_3
         Index Cond: ((i >= 990) AND (i <= 3100))
         Filter: ((j < 9910) AND (k > 50))
```

```
   ->  Index Scan using tbl_a4_i_idx on tbl_a4 tbl_a_4
         Index Cond: ((i >= 990) AND (i <= 3100))
         Filter: ((j < 9910) AND (k > 50))
 Note: An Approved plan was used instead of the minimum cost plan.
 SQL Hash: 1553185667, Plan Hash: -993736942, Minimum Cost Plan Hash: -1873216041
(14 rows)
```

考虑另一项已批准的计划，每个分区使用不同的访问方法：

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 2100 and j < 9910 and k > 50;

                                QUERY PLAN
---------------------------------------------------------------------------
 Append
   ->  Index Scan using tbl_a1_i_idx on tbl_a1 tbl_a_1
         Index Cond: ((i >= 990) AND (i <= 2100))
         Filter: ((j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 2100) AND (j < 9910) AND (k > 50))
   ->  Bitmap Heap Scan on tbl_a3 tbl_a_3
         Recheck Cond: ((i >= 990) AND (i <= 2100))
         Filter: ((j < 9910) AND (k > 50))
         ->  Bitmap Index Scan on tbl_a3_i_idx
               Index Cond: ((i >= 990) AND (i <= 2100))
 SQL Hash: 1553185667, Plan Hash: 2032136998
(12 rows)
```

在这种情况下，任何从两个分区进行读取的计划都将无法强制执行。除非批准的计划中的所有（访问方法、索引名称）组合都可用，否则该计划将无法强制执行。例如，以下计划具有不同的计划哈希值，在这些情况下无法强制执行批准的计划：

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 1900 and j < 9910 and k > 50;

                              QUERY PLAN
---------------------------------------------------------------------------
 Append
   ->  Bitmap Heap Scan on tbl_a1 tbl_a_1
         Recheck Cond: ((i >= 990) AND (i <= 1900))
         Filter: ((j < 9910) AND (k > 50))
         ->  Bitmap Index Scan on tbl_a1_i_idx
               Index Cond: ((i >= 990) AND (i <= 1900))
```

```
   ->  Bitmap Heap Scan on tbl_a2 tbl_a_2
         Recheck Cond: ((i >= 990) AND (i <= 1900))
         Filter: ((j < 9910) AND (k > 50))
         ->  Bitmap Index Scan on tbl_a2_i_idx
               Index Cond: ((i >= 990) AND (i <= 1900))
  Note: This is not an Approved plan.  No usable Approved plan was found.
  SQL Hash: 1553185667, Plan Hash: -568647260
(13 rows)
```

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 1900 and j < 9910 and k > 50;

                              QUERY PLAN
-----------------------------------------------------------------------
 Append
   ->  Index Scan using tbl_a1_i_idx on tbl_a1 tbl_a_1
         Index Cond: ((i >= 990) AND (i <= 1900))
         Filter: ((j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 1900) AND (j < 9910) AND (k > 50))
 Note: This is not an Approved plan.  No usable Approved plan was found.
 SQL Hash: 1553185667, Plan Hash: -496793743
(8 rows)
```

命名约定

为了让 QPM 强制执行声明式分区表计划，必须遵循父表、表分区和索引的特定命名规则：

- 父表名称 – 这些名称在字母或特殊字符方面必须有所不同，而不仅仅是数字方面的不同。例如，tA、tB 和 tC 是单独父表的可接受名称，而 t1、t2 和 t3 不是。
- 单个分区表名称 – 同一父表的分区表之间应只存在数字方面的不同。例如，tA 的可接受分区名称可以是 tA1、tA2 或 t1A、t2A 甚至多位数。

  任何其他差异（字母、特殊字符）均不能保证计划强制执行。
- 索引名称 – 在分区表层次结构中，请确保所有索引都具有唯一的名称。这意味着名称的非数字部分必须不同。例如，如果您有一个名为 tA 的分区表，其索引名称为 tA_col1_idx1，则不能再将其他索引命名为 tA_col1_idx2。但是，您可以具有名为 tA_a_col1_idx2 的索引，因为名称的非数字部分是唯一的。此规则适用于在父表和单个分区表上创建的索引。

不遵守上述命名约定可能会导致批准的计划无法强制执行。以下示例说明了这种无法强制执行的情况：

```
postgres=>create table t1(i int, j int, k int, l int, m int) partition by range(i);
CREATE TABLE
postgres=>create table t1a partition of t1 for values from (0) to (1000);
CREATE TABLE
postgres=>create table t1b partition of t1 for values from (1001) to (2000);
CREATE TABLE
postgres=>SET apg_plan_mgmt.capture_plan_baselines TO 'manual';
SET
postgres=>explain (hashes true, costs false) select count(*) from t1 where i > 0;

                           QUERY PLAN
-----------------------------------------------------------------------------
 Aggregate
   ->  Append
         ->  Seq Scan on t1a t1_1
               Filter: (i > 0)
         ->  Seq Scan on t1b t1_2
               Filter: (i > 0)
 SQL Hash: -1720232281, Plan Hash: -1010664377
(7 rows)
```

```
postgres=>SET apg_plan_mgmt.use_plan_baselines TO 'on';
SET
postgres=>explain (hashes true, costs false) select count(*) from t1 where i > 1000;

                           QUERY PLAN
-----------------------------------------------------------------------------
 Aggregate
   ->  Seq Scan on t1b t1
         Filter: (i > 1000)
 Note: This is not an Approved plan. No usable Approved plan was found.
 SQL Hash: -1720232281, Plan Hash: 335531806
(5 rows)
```

尽管这两个计划可能看起来相同，但由于子表的名称，它们的 Plan Hash 值有所不同。表名称因字母字符而异，而不仅仅是导致执行失败的数字。

使用扩展和外部数据包装器

要扩展 Aurora PostgreSQL 兼容版数据库集群的功能，您可以安装和使用各种 PostgreSQL 扩展。例如，如果您的使用案例要求在非常大的表中输入密集型数据，则可以安装 pg_partman 扩展以对数据进行分区，从而分散工作负载。

> **Note**
>
> 自 Aurora PostgreSQL 14.5 起，Aurora PostgreSQL 支持适用于 PostgreSQL 的可信语言扩展。此特征是作为扩展 pg_tle 实现的，您可以将其添加到 Aurora PostgreSQL 中。通过使用此扩展，开发人员可以在安全的环境中创建自己的 PostgreSQL 扩展，从而简化设置和配置要求以及新扩展的许多初步测试。有关更多信息，请参阅 使用适用于 PostgreSQL 的可信语言扩展。

在某些情况下，您可以将特定模块添加到 Aurora PostgreSQL 数据库集群的自定义数据库集群参数组中的 shared_preload_libraries 列表中，而不是安装扩展。通常，默认的数据库集群参数组仅加载 pg_stat_statements，但还有其他几个模块可供添加到此列表中。例如，您可以通过添加 pg_cron 模块来添加调度功能，详情请见使用 PostgreSQL pg_cron 扩展计划维护。再举一个例子，您可以通过加载 auto_explain 模块来记录查询执行计划。要了解更多信息，请参阅 AWS 知识中心中的记录查询执行计划。

提供对外部数据的访问权限的扩展更具体地称为外部数据包装器（FDW）。例如，oracle_fdw 扩展允许 Aurora PostgreSQL 数据库集群使用 Oracle 数据库。

您还可以通过在 rds.allowed_extensions 参数中列出扩展，精确指定可以在 Aurora PostgreSQL 数据库实例上安装哪些扩展。有关更多信息，请参阅限制 PostgreSQL 扩展的安装。

接下来，您可以找到有关设置和使用 Aurora PostgreSQL 可用的一些扩展、模块和 FDW 的信息。为简单起见，这些都称为“扩展”。您可以找到可与当前可用的 Aurora PostgreSQL 版本结合使用的扩展列表，请参阅《Aurora PostgreSQL 版本注释》中的 Amazon Aurora PostgreSQL 的扩展版本。

- 使用 lo 模块管理大型对象
- 使用 PostGIS 扩展管理空间数据
- 使用 pg_partman 扩展管理 PostgreSQL 分区
- 使用 PostgreSQL pg_cron 扩展计划维护
- 使用 pgAudit 记录数据库活动

- 使用 pglogical 跨实例同步数据
- 通过使用 oracle_fdw 扩展来使用 Oracle 数据库
- 通过使用 tds_fdw 扩展来使用 SQL Server 数据库

使用适用于 PostgreSQL 的 Amazon Aurora 委派扩展支持

使用适用于 PostgreSQL 的 Amazon Aurora 委派扩展支持，您可以将扩展管理委派给其他用户，而且不必让该用户成为 `rds_superuser`。通过这种委派扩展支持，将创建一个名为 `rds_extension` 的新角色，您必须将其分配给用户才能管理其他扩展。此角色可以创建、更新和删除扩展。

您可以通过在 `rds.allowed_extensions` 参数中列出扩展，指定可以在 Aurora PostgreSQL 数据库实例上安装哪些扩展。有关更多信息，请参阅将 PostgreSQL 扩展与 Amazon RDS for PostgreSQL 结合使用。

您可以通过 `rds.allowed_delegated_extensions` 参数来限制具有 `rds_extension` 角色的用户可以管理的可用扩展列表。

委派扩展支持在以下版本中可用：

- 所有更高版本
- 15.5 及更高的 15 版本
- 14.10 及更高的 14 版本
- 13.13 及更高的 13 版本
- 12.17 及更高的 12 版本

主题

- 为用户开启委派扩展支持
- 在适用于 PostgreSQL 的 Aurora 委派扩展支持中使用的配置
- 关闭对委派扩展的支持
- 使用 Amazon Aurora 委派扩展支持的优势
- 适用于 PostgreSQL 的 Aurora 委派扩展支持的限制
- 某些扩展所需的权限
- 安全考虑因素

- 已禁用删除扩展级联
- 可以使用委派扩展支持添加的扩展示例

为用户开启委派扩展支持

必须执行以下操作才能为用户启用委派扩展支持：

1. 向用户授予 **rds_extension** 角色：以 rds_superuser 身份连接到数据库并执行以下命令：

```
Postgres => grant rds_extension to user_name;
```

2. 设置可供委派用户管理的扩展列表：rds.allowed_delegated_extensions 允许您使用 rds.allowed_extensions 在数据库集群参数中指定可用扩展的子集。您可以在以下级别之一执行此操作：

   - 在集群或实例参数组中，通过 AWS Management Console或 API。有关更多信息，请参阅 Amazon Aurora 的参数组。
   - 在数据库级别使用以下命令：

```
alter database database_name set rds.allowed_delegated_extensions =
 'extension_name_1,
                   extension_name_2,...extension_name_n';
```

   - 在用户级别使用以下命令：

```
alter user user_name set rds.allowed_delegated_extensions = 'extension_name_1,
                   extension_name_2,...extension_name_n';
```

   > **Note**
   >
   > 更改 rds.allowed_delegated_extensions 动态参数后，无需重启数据库。

3. 允许委派用户访问在扩展创建过程中创建的对象：某些扩展创建的对象要求先授予其它权限，然后具有 rds_extension 角色的用户才能访问它们。rds_superuser 必须向委派用户授予对这些对象的访问权限。其中一个选项是使用事件触发器自动向委派用户授予权限。有关更多信息，请参阅关闭对委派扩展的支持中的事件触发器示例。

在适用于 PostgreSQL 的 Aurora 委派扩展支持中使用的配置

配置名称	描述	默认值	注意	谁可以修改或授予权限
rds.allowed_delegated_extensions	此参数限制 rds_extension 角色可以在数据库中管理的扩展。它必须是 rds.allowed_extensions 的子集。	空字符串	• 默认情况下，此参数为空字符串，这意味着没有向具有 rds_extension 的用户委派任何扩展。 • 如果用户有添加受支持的扩展的权限，则可以添加此扩展。为此，请将 rds.allowed_delegated_extensions 参数设置为一串以逗号分隔的扩展名称。通过向该参数添加扩展列表，可以显式确定具有 rds_extension 角色的用户可以安装的扩展。 • 如果设置为 *，则表示 rds_allowed_extensions 中列出的所有扩展都将委派给具有 rds_exten	rds_superuser

配置名称	描述	默认值	注意	谁可以修改或授予权限
			sion 角色的用户。 要了解有关设置此参数的更多信息，请参阅为用户开启委派扩展支持。	
rds.allowed_extensions	此参数可让客户限制可以在 Aurora PostgreSQL 数据库实例中安装的扩展。有关更多信息，请参阅限制 PostgreSQL 扩展的安装。	"*"	默认情况下，此参数设置为“*”，这意味着具有必要权限的用户能够创建 RDS for PostgreSQL 和 Aurora PostgreSQL 支持的所有扩展。 空表示无法在 Aurora PostgreSQL 数据库实例中安装任何扩展。	管理员

配置名称	描述	默认值	注意	谁可以修改或授予权限
rds-delegated_nsionow_drascac	此参数控制拥有 rds_extension 的用户使用级联选项删除扩展的能力。	off	默认情况下，将 rds-delegated_extension_allow_drop_cascade 设置为 off。这意味着不允许拥有 rds_extension 的用户使用级联选项删除扩展。 要授予该能力，rds.delegated_extension_allow_drop_cascade 参数应设置为 on。	rds_superuser

关闭对委派扩展的支持

部分关闭

委派用户无法创建新的扩展，但仍然可以更新现有的扩展。

- 在数据库集群参数组中将 rds.allowed_delegated_extensions 重置为默认值。
- 在数据库级别使用以下命令：

```
alter database database_name reset rds.allowed_delegated_extensions;
```

- 在用户级别使用以下命令：

```
alter user user_name reset rds.allowed_delegated_extensions;
```

完全关闭

撤销用户的 rds_extension 角色会将该用户恢复为标准权限。用户无法再创建、更新或删除扩展。

```
postgres => revoke rds_extension from user_name;
```

事件触发器示例

如果您希望允许具有 rds_extension 的委派用户使用扩展，而此类扩展要求对通过扩展创建过程所创建的对象设置权限，则可以自定义以下事件触发器示例，并仅添加您希望委派用户有权访问其全部功能的扩展。此事件触发器可以在 template1（默认模板）上创建，因此所有从 template1 创建的数据库都将具有该事件触发器。当委派用户安装扩展时，此触发器将自动授予对扩展创建的对象的所有权。

```
CREATE OR REPLACE FUNCTION create_ext()

  RETURNS event_trigger AS $$

DECLARE

  schemaname TEXT;
  databaseowner TEXT;

  r RECORD;

BEGIN

  IF tg_tag = 'CREATE EXTENSION' and current_user != 'rds_superuser' THEN
    RAISE NOTICE 'SECURITY INVOKER';
    RAISE NOTICE 'user: %', current_user;
    FOR r IN SELECT * FROM pg_event_trigger_ddl_commands()
    LOOP
        CONTINUE WHEN r.command_tag != 'CREATE EXTENSION' OR r.object_type !=
 'extension';

        schemaname = (
            SELECT n.nspname
            FROM pg_catalog.pg_extension AS e
            INNER JOIN pg_catalog.pg_namespace AS n
            ON e.extnamespace = n.oid
            WHERE e.oid = r.objid
        );
```

```
        databaseowner = (
            SELECT pg_catalog.pg_get_userbyid(d.datdba)
            FROM pg_catalog.pg_database d
            WHERE d.datname = current_database()
        );
        RAISE NOTICE 'Record for event trigger %, objid: %,tag: %, current_user: %,
 schema: %, database_owenr: %', r.object_identity, r.objid, tg_tag, current_user,
 schemaname, databaseowner;
        IF r.object_identity = 'address_standardizer_data_us' THEN
            EXECUTE format('GRANT SELECT, UPDATE, INSERT, DELETE ON TABLE %I.us_gaz TO
 %I WITH GRANT OPTION;', schemaname, databaseowner);
            EXECUTE format('GRANT SELECT, UPDATE, INSERT, DELETE ON TABLE %I.us_lex TO
 %I WITH GRANT OPTION;', schemaname, databaseowner);
            EXECUTE format('GRANT SELECT, UPDATE, INSERT, DELETE ON TABLE %I.us_rules
 TO %I WITH GRANT OPTION;', schemaname, databaseowner);
        ELSIF r.object_identity = 'dict_int' THEN
            EXECUTE format('ALTER TEXT SEARCH DICTIONARY %I.intdict OWNER TO %I;',
 schemaname, databaseowner);
        ELSIF r.object_identity = 'pg_partman' THEN
            EXECUTE format('GRANT SELECT, UPDATE, INSERT, DELETE ON TABLE
 %I.part_config TO %I WITH GRANT OPTION;', schemaname, databaseowner);
            EXECUTE format('GRANT SELECT, UPDATE, INSERT, DELETE ON TABLE
 %I.part_config_sub TO %I WITH GRANT OPTION;', schemaname, databaseowner);
            EXECUTE format('GRANT SELECT, UPDATE, INSERT, DELETE ON TABLE
 %I.custom_time_partitions TO %I WITH GRANT OPTION;', schemaname, databaseowner);
        ELSIF r.object_identity = 'postgis_topology' THEN
            EXECUTE format('GRANT SELECT, UPDATE, INSERT, DELETE ON ALL TABLES IN
 SCHEMA topology TO %I WITH GRANT OPTION;', databaseowner);
            EXECUTE format('GRANT USAGE, SELECT ON ALL SEQUENCES IN SCHEMA topology TO
 %I WITH GRANT OPTION;', databaseowner);
            EXECUTE format('GRANT EXECUTE ON ALL FUNCTIONS IN SCHEMA topology TO %I
 WITH GRANT OPTION;', databaseowner);
            EXECUTE format('GRANT USAGE ON SCHEMA topology TO %I WITH GRANT OPTION;',
 databaseowner);
        END IF;
    END LOOP;
  END IF;
END;
$$ LANGUAGE plpgsql SECURITY DEFINER;

CREATE EVENT TRIGGER log_create_ext ON ddl_command_end EXECUTE PROCEDURE create_ext();
```

使用 Amazon Aurora 委派扩展支持的优势

通过使用适用于 PostgreSQL 的 Amazon Aurora 委派扩展支持，您可以安全地将扩展管理委派给没有 `rds_superuser` 角色的用户。该功能具有以下优势：

- 您可以轻松地将扩展管理委派给您选择的用户。
- 这不需要 `rds_superuser` 角色。
- 支持为同一个数据库集群中的不同数据库提供不同的扩展集。

适用于 PostgreSQL 的 Aurora 委派扩展支持的限制

- 在扩展创建过程中创建的对象可能需要额外的权限才能使扩展正常运行。

某些扩展所需的权限

要创建、使用或更新以下扩展，委派用户应具有对以下函数、表和架构的必要权限。

需要所有权或权限的扩展	函数	表	架构	文本搜索词典	注释
addr tand er_d s		us_gaz、us _lex、us_l ex、l.us_rules			
amch	bt_index_ check、bt_				

需要所有权或权限的扩展	函数	表	架构	文本搜索词典	注释
	index_parent_check				
dict_i				intdict	
pg_pn		custom_time_partitions、part_config、part_config_sub			
pg_statem					
Postg	st_tileenvelope	spatial_ref_sys			
postg aster					
postg opolo		topology、layer	topology		委派用户必须是数据库所有者
log_f	create_foreign_table_for_log_file				

需要所有权或权限的扩展	函数	表	架构	文本搜索词典	注释
rds_t	role_password_encryption_type				
postg iger_ oder		geocode_settings_default、geocode_settings	tiger		
pg_fr acem	pg_freespace				
pg_vi lity	pg_visibility				

安全考虑因素

请记住，具有 rds_extension 角色的用户将能够管理他们拥有连接权限的所有数据库上的扩展。如果打算让委派用户管理单个数据库上的扩展，那么一个好的做法是撤销每个数据库上的公有权限，然后向委派用户显式授予对该特定数据库的连接权限。

有几个扩展可以允许用户访问多个数据库中的信息。在向 rds.allowed_delegated_extensions 添加这些扩展之前，请确保您授予 rds_extension 的用户具有跨数据库功能。例如，postgres_fdw 和 dblink 提供在同一实例或远程实例上跨数据库进行查询的功能。log_fdw 读取 postgres 引擎日志文件，这些文件适用于实例中的所有数据库，可能包含来自多个数据库的慢速

查询或错误消息。pg_cron 允许在数据库实例上运行计划的后台任务，并且可以将任务配置为在不同的数据库中运行。

已禁用删除扩展级联

具有 rds_extension 角色的用户使用级联选项删除扩展的能力由 rds.delegated_extension_allow_drop_cascade 参数控制。默认情况下，将 rds-delegated_extension_allow_drop_cascade 设置为 off。这意味着不允许具有 rds_extension 角色的用户使用级联选项删除扩展，如以下查询所示。

```
DROP EXTENSION CASCADE;
```

因为这将自动删除依赖于扩展的对象，进而删除依赖于这些对象的所有对象。尝试使用级联选项将会导致错误。

要授予该能力，rds.delegated_extension_allow_drop_cascade 参数应设置为 on。

更改 rds.delegated_extension_allow_drop_cascade 动态参数不需要重启数据库。您可以在以下级别之一执行此操作：

- 在集群或实例参数组中，通过 AWS Management Console或 API。
- 在数据库级别使用以下命令：

```
alter database database_name set rds.delegated_extension_allow_drop_cascade = 'on';
```

- 在用户级别使用以下命令：

```
alter role tenant_user set rds.delegated_extension_allow_drop_cascade = 'on';
```

可以使用委派扩展支持添加的扩展示例

- rds_tools

```
extension_test_db=> create extension rds_tools;
CREATE EXTENSION
extension_test_db=> SELECT * from rds_tools.role_password_encryption_type() where
 rolname = 'pg_read_server_files';
ERROR: permission denied for function role_password_encryption_type
```

- amcheck

```
extension_test_db=> CREATE TABLE amcheck_test (id int);
CREATE TABLE
extension_test_db=> INSERT INTO amcheck_test VALUES (generate_series(1,100000));
INSERT 0 100000
extension_test_db=> CREATE INDEX amcheck_test_btree_idx ON amcheck_test USING btree
 (id);
CREATE INDEX
extension_test_db=> create extension amcheck;
CREATE EXTENSION
extension_test_db=> SELECT bt_index_check('amcheck_test_btree_idx'::regclass);
ERROR: permission denied for function bt_index_check
extension_test_db=> SELECT bt_index_parent_check('amcheck_test_btree_idx'::regclass);
ERROR: permission denied for function bt_index_parent_check
```

- pg_freespacemap

```
extension_test_db=> create extension pg_freespacemap;
CREATE EXTENSION
extension_test_db=> SELECT * FROM pg_freespace('pg_authid');
ERROR: permission denied for function pg_freespace
extension_test_db=> SELECT * FROM pg_freespace('pg_authid',0);
ERROR: permission denied for function pg_freespace
```

- pg_visibility

```
extension_test_db=> create extension pg_visibility;
CREATE EXTENSION
extension_test_db=> select * from pg_visibility('pg_database'::regclass);
ERROR: permission denied for function pg_visibility
```

- postgres_fdw

```
extension_test_db=> create extension postgres_fdw;
CREATE EXTENSION
extension_test_db=> create server myserver foreign data wrapper postgres_fdw options
 (host 'foo', dbname 'foodb', port '5432');
ERROR: permission denied for foreign-data wrapper postgres_fdw
```

使用 lo 模块管理大型对象

lo 模块（扩展）适用于通过 JDBC 或 ODBC 驱动程序使用 PostgreSQL 数据库的数据库用户和开发人员。JDBC 和 ODBC 都希望数据库能够在对大型对象的引用发生更改时处理对大型对象的删除。但 PostgreSQL 并非如此。PostgreSQL 并不假定在对于对象的引用发生变化时应该删除该对象。结果是对象保留在磁盘上，未引用。lo 扩展包括一个函数，您可以使用该函数在引用发生变化时触发，以便在需要时删除对象。

> **Tip**
>
> 要确定数据库是否可以从 lo 扩展中受益，请使用 `vacuumlo` 实用程序检查孤立的大型对象。要在不采取任何操作的情况下获取孤立的大型对象的计数，请使用 `-n` 选项（无操作）运行此实用程序。要了解如何操作，请参阅下面的 vacuumlo utility。

lo 模块适用于 Aurora PostgreSQL 13.7、12.11、11.16、10.21 及更高的次要版本。

要安装模块（扩展），您需要 `rds_superuser` 权限。安装 lo 扩展会将以下内容添加到数据库中：

- `lo` – 这是一种大型对象 (lo) 数据类型，可用于二进制大型对象 (BLOB) 和其他大型对象。`lo` 数据类型是 `oid` 数据类型的域。换句话说，它是一个具有可选限制的对象标识符。有关更多信息，请参阅 PostgreSQL 文档中的对象标识符。简单来说，您可以使用 `lo` 数据类型，以区分包含大型对象引用的数据库列与其他对象标识符 (OID)。
- `lo_manage` – 这是一个函数，您可以在包含大型对象引用的表列上的触发器中使用。无论何时删除或修改引用大型对象的值，触发器都会取消 (`lo_unlink`) 对象与其引用之间的关联。仅当列是对大型对象的唯一数据库引用时，才对列使用触发器。

有关大型对象模块的更多信息，请参阅 PostgreSQL 文档中的 Lo。

安装 lo 扩展

在安装 lo 扩展之前，请确保您具有 `rds_superuser` 权限。

安装 lo 扩展

1. 使用 `psql` 连接到 Aurora PostgreSQL 数据库集群的主数据库实例。

```
psql --host=your-cluster-instance-1.666666666666.aws-region.rds.amazonaws.com --
port=5432 --username=postgres --password
```

出现提示时请输入密码。psql 客户端会建立连接并显示默认的管理连接数据库 postgres=>，作为提示符。

2. 按如下方式安装扩展。

```
postgres=> CREATE EXTENSION lo;
CREATE EXTENSION
```

您现在可以使用 lo 数据类型定义表中的列。例如，您可以创建包含光栅图像数据的表 (images)。您可以对列 raster 使用 lo 数据类型，如以下示例所示，它将创建一个表。

```
postgres=> CREATE TABLE images (image_name text, raster lo);
```

使用 lo_manage 触发器函数删除对象

在更新或删除 lo 时，可以在 lo 或其他大型对象列上的触发器中使用 lo_manage 函数来清理（并防止出现孤立对象）。

在引用大型对象的列上设置触发器

- 请执行以下操作之一：
  - 对此参数使用列名称，在每个列上创建一个 BEFORE UPDATE OR DELETE 触发器，以包含对大型对象的唯一引用。

    ```
    postgres=> CREATE TRIGGER t_raster BEFORE UPDATE OR DELETE ON images
        FOR EACH ROW EXECUTE FUNCTION lo_manage(raster);
    ```

  - 仅在更新列时才应用触发器。

    ```
    postgres=> CREATE TRIGGER t_raster BEFORE UPDATE OF images
        FOR EACH ROW EXECUTE FUNCTION lo_manage(raster);
    ```

lo_manage 触发器函数仅在插入或删除列数据的上下文中起作用，具体取决于您定义触发器的方式。对数据库执行 DROP 或 TRUNCATE 操作时，它不起作用。这意味着在删除之前应该从任何表中删除对象列，以防止创建孤立的对象。

例如，假设您要删除包含 images 表的数据库。您可以按如下方式删除该列。

```
postgres=> DELETE FROM images COLUMN raster
```

假设在该列上定义了用于处理删除 lo_manage 的函数，现在可以安全地删除该表。

使用 vacuumlo 删除孤立的大型对象

vacuumlo 实用程序识别孤立的大型对象并从数据库中删除它们。此实用程序自 PostgreSQL 9.1.24 以来一直可用。如果数据库用户定期使用大型对象，建议您偶尔运行 vacuumlo 来清理孤立的大型对象。

在安装 lo 扩展之前，您可以使用 vacuumlo 评估 Aurora PostgreSQL 数据库集群是否可以受益。要执行此操作，请使用 -n 选项（无操作）运行 vacuumlo 以显示要删除的内容，如下所示：

```
$ vacuumlo -v -n -h your-cluster-instance-1.666666666666.aws-region.rds.amazonaws.com -
p 5433 -U postgres docs-lab-spatial-db
Password:*****
Connected to database "docs-lab-spatial-db"
Test run: no large objects will be removed!
Would remove 0 large objects from database "docs-lab-spatial-db".
```

如输出所示，孤立的大型对象对于这个特定的数据库来说并不是问题。

有关此实用程序的更多信息，请参阅 PostgreSQL 文档中的 vacuumlo。

了解 vacuumlo 的工作方式

vacuumlo 命令从 PostgreSQL 数据库中删除孤立的大型对象（LO），而不会影响用户表或与用户表冲突。

该命令的工作原理如下：

1. vacuumlo 首先创建一个临时表，其中包含数据库中大型对象的所有对象 ID（OID）。
2. 然后，vacuumlo 会全面地扫描数据库中使用 oid 或 lo 数据类型的每一列。如果 vacuumlo 在这些列中找到匹配的 OID，则会从临时表中删除该 OID。vacuumlo 仅检查专门命名为 oid 或 lo 的列，而不检查基于这些类型的域。
3. 临时表中的其余条目表示孤立的大型对象，然后 vacuumlo 会安全地删除这些大型对象。

提高 vacuumlo 性能

您可以通过使用 -l 选项增加批处理大小来提高 vacuumlo 的性能。这样将允许 vacuumlo 一次处理更多大型对象。

如果您的系统有足够的内存，并且您可以将临时表完全容纳在内存中，则在数据库级别增加 temp_buffers 设置可提高性能。这样有助于表完全驻留在内存中，从而可提高总体性能。

接下来查询会估计临时表的大小：

```
SELECT
    pg_size_pretty(SUM(pg_column_size(oid))) estimated_lo_temp_table_size
FROM
    pg_largeobject_metadata;
```

大型对象的注意事项

接下来，您可以找到在处理大型对象时需要注意的一些重要注意事项：

- Vacuumlo 是唯一的解决方案，因为目前没有其他方法可以删除孤立的大型对象。
- 使用复制技术的诸如 pglogical、本机逻辑复制和 AWS DMS 等工具不支持复制大型对象。
- 在设计数据库架构时，请尽可能避免使用大型对象，并考虑改用 bytea 等替代数据类型。
- 定期运行 vacuumlo，至少每周一次，以防止孤立的大型对象出现问题。
- 在存储大型对象的表上使用带有 lo_manage 函数的触发器，以帮助防止创建孤立的大型对象。

使用 PostGIS 扩展管理空间数据

PostGIS 是一个 PostgreSQL 扩展，用于存储和管理空间信息。要了解有关 PostGIS 的更多信息，请参阅 PostGIS.net。

从版本 10.5 开始，PostgreSQL 支持 PostGIS 用于处理地图框矢量平铺数据的 libprotobuf 1.3.0 库。

设置 PostGIS 扩展需要 rds_superuser 权限。我们建议您创建一个用户（角色），以管理 PostGIS 和您的空间数据。PostGIS 扩展及其相关组件为 PostgreSQL 添加了数千个函数。如果这对您的使用案例有意义，请考虑在自己的架构中创建 PostGIS 扩展。以下示例说明了如何在其各自的数据库中安装扩展，但这并不是必需的。

主题

- 步骤 1：创建用户（角色）来管理 PostGIS 扩展
- 步骤 2：加载 PostGIS 扩展
- 步骤 3：移交扩展架构的所有权
- 步骤 4：移交 PostGIS 表的所有权
- 步骤 5：测试扩展
- 步骤 6：升级 PostGIS 扩展
- PostGIS 扩展版本
- 将 PostGIS 2 升级到 PostGIS 3

步骤 1：创建用户（角色）来管理 PostGIS 扩展

首先，以具有 rds_superuser 权限的用户身份连接到 RDS for PostgreSQL 数据库实例。如果您在设置实例时保留原定设置名称，则以 postgres 进行连接。

```
psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --username=postgres
 --password
```

创建单独的角色（用户）来管理 PostGIS 扩展。

```
postgres=>  CREATE ROLE gis_admin LOGIN PASSWORD 'change_me';
CREATE ROLE
```

向此角色授予 rds_superuser 权限，以允许角色安装扩展。

```
postgres=> GRANT rds_superuser TO gis_admin;
GRANT
```

创建一个要用于 PostGIS 构件的数据库。此为可选步骤。或者，您可以在用户数据库中为 PostGIS 扩展创建架构，但这也不是必需的。

```
postgres=> CREATE DATABASE lab_gis;
CREATE DATABASE
```

为 gis_admin 提供 lab_gis 数据库的所有权限。

```
postgres=> GRANT ALL PRIVILEGES ON DATABASE lab_gis TO gis_admin;
GRANT
```

退出会话并以 gis_admin 身份重新连接 RDS for PostgreSQL 数据库实例。

```
postgres=> psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --
username=gis_admin --password --dbname=lab_gis
Password for user gis_admin:...
lab_gis=>
```

按照后续步骤中的详细说明，继续设置扩展。

步骤 2：加载 PostGIS 扩展

PostGIS 扩展模块包括多个相关的扩展，它们协同工作以提供地理空间功能。您可能不需要在此步骤中创建的所有扩展，具体取决于您的使用案例。

使用 CREATE EXTENSION 语句加载 PostGIS 扩展。

```
CREATE EXTENSION postgis;
CREATE EXTENSION
CREATE EXTENSION postgis_raster;
CREATE EXTENSION
CREATE EXTENSION fuzzystrmatch;
CREATE EXTENSION
CREATE EXTENSION postgis_tiger_geocoder;
CREATE EXTENSION
CREATE EXTENSION postgis_topology;
CREATE EXTENSION
CREATE EXTENSION address_standardizer_data_us;
CREATE EXTENSION
```

您可以通过运行以下示例中显示的 SQL 查询来验证结果，该查询列出了扩展及其所有者。

```
SELECT n.nspname AS "Name",
  pg_catalog.pg_get_userbyid(n.nspowner) AS "Owner"
  FROM pg_catalog.pg_namespace n
  WHERE n.nspname !~ '^pg_' AND n.nspname <> 'information_schema'
  ORDER BY 1;
List of schemas
     Name     |   Owner
--------------+-----------
 public       | postgres
 tiger        | rdsadmin
 tiger_data   | rdsadmin
```

```
 topology     | rdsadmin
(4 rows)
```

步骤 3：移交扩展架构的所有权

使用 ALTER SCHEMA 语句将架构的所有权移交给 gis_admin 角色。

```
ALTER SCHEMA tiger OWNER TO gis_admin;
ALTER SCHEMA
ALTER SCHEMA tiger_data OWNER TO gis_admin;
ALTER SCHEMA
ALTER SCHEMA topology OWNER TO gis_admin;
ALTER SCHEMA
```

您可以通过运行以下 SQL 查询来确认所有权变更。或者，您也可以从 psql 命令行使用 \dn 元命令。

```
SELECT n.nspname AS "Name",
  pg_catalog.pg_get_userbyid(n.nspowner) AS "Owner"
  FROM pg_catalog.pg_namespace n
  WHERE n.nspname !~ '^pg_' AND n.nspname <> 'information_schema'
  ORDER BY 1;

       List of schemas
     Name     |     Owner
--------------+---------------
 public       | postgres
 tiger        | gis_admin
 tiger_data   | gis_admin
 topology     | gis_admin
(4 rows)
```

步骤 4：移交 PostGIS 表的所有权

> **Note**
>
> 请勿更改 PostGIS 函数的所有权。PostGIS 的正常运行和未来升级要求这些函数保留原始所有权。有关 PostGIS 权限的更多信息，请参阅 PostgreSQL Security。

使用以下函数将 PostGIS 表的所有权移交给 gis_admin 角色。从 psql 提示符处运行以下语句以创建此函数。

```
CREATE FUNCTION exec(text) returns text language plpgsql volatile AS $f$ BEGIN EXECUTE
 $1; RETURN $1; END; $f$;
CREATE FUNCTION
```

接下来，运行以下查询以运行 exec 函数，该函数进而将运行语句并更改权限。

```
SELECT exec('ALTER TABLE ' || quote_ident(s.nspname) || '.' || quote_ident(s.relname)
 || ' OWNER TO gis_admin;')
  FROM (
    SELECT nspname, relname
    FROM pg_class c JOIN pg_namespace n ON (c.relnamespace = n.oid)
    WHERE nspname in ('tiger','topology') AND
    relkind IN ('r','S','v') ORDER BY relkind = 'S')
s;
```

步骤 5：测试扩展

为避免需要指定架构名称，请使用以下命令将 tiger 架构添加到搜索路径中。

```
SET search_path=public,tiger;
SET
```

使用以下 SELECT 语句测试 tiger 架构。

```
SELECT address, streetname, streettypeabbrev, zip
 FROM normalize_address('1 Devonshire Place, Boston, MA 02109') AS na;
address | streetname | streettypeabbrev |  zip
---------+------------+------------------+-------
       1 | Devonshire | Pl               | 02109
(1 row)
```

要了解有关此扩展的更多信息，请参阅 PostGIS 文档中的 Tiger 地理编码器。

使用以下 SELECT 语句测试对 topology 架构的访问。这将调用 createtopology 函数，以使用指定的空间参考标识符（26986）和原定设置容差（0.5）注册新的拓扑对象（my_new_topo）。要了解更多信息，请参阅 PostGIS 文档中的 CreateTopology。

```
SELECT topology.createtopology('my_new_topo',26986,0.5);
 createtopology
----------------
              1
```

```
(1 row)
```

步骤 6：升级 PostGIS 扩展

PostgreSQL 的每个新发行版都支持与该发行版兼容的一个或多个 PostGIS 扩展版本。将 PostgreSQL 引擎升级到新版本不会自动升级 PostGIS 扩展。在升级 PostgreSQL 引擎之前，通常需要将 PostGIS 升级到当前 PostgreSQL 版本的最新可用版本。有关详细信息，请参阅PostGIS 扩展版本。

PostgreSQL 引擎升级后，您可以再次将 PostGIS 扩展升级到新升级的 PostgreSQL 引擎版本所支持的版本。有关升级 PostgreSQL 引擎的更多信息，请参阅测试将生产数据库集群升级到新的主要版本。

您可以随时检查 Aurora PostgreSQL 数据库集群上可用的 PostGIS 扩展版本更新。为此，请运行以下命令。PostGIS 2.5.0 及更高版本可以使用此功能。

```
SELECT postGIS_extensions_upgrade();
```

如果您的应用程序不支持最新的 PostGIS 版本，您可以安装主要版本中提供的旧版本 PostGIS，如下所示。

```
CREATE EXTENSION postgis VERSION "2.5.5";
```

如果要从旧版本升级到特定 PostGIS 版本，还可以使用以下命令。

```
ALTER EXTENSION postgis UPDATE TO "2.5.5";
```

根据要从中进行升级的版本，您可能需要再次使用此函数。第一次运行该函数的结果决定是否需要额外的升级功能。例如，从 PostGIS 2 升级到 PostGIS 3 就是这种情况。有关更多信息，请参阅 将 PostGIS 2 升级到 PostGIS 3。

如果您升级此扩展是为了准备进行 PostgreSQL 引擎的主要版本升级，则可以继续执行其他初步任务。有关更多信息，请参阅测试将生产数据库集群升级到新的主要版本。

PostGIS 扩展版本

我们建议您安装《Aurora PostgreSQL 版本注释》的 Aurora PostgreSQL 兼容版的扩展版本 中列出的所有扩展版本，如 PostGIS。要获取发行版中可用的版本列表，请使用以下命令。

```
SELECT * FROM pg_available_extension_versions WHERE name='postgis';
```

您可以在《Aurora PostgreSQL 的版本注释》的以下部分中找到版本信息：

- Aurora PostgreSQL 14 的扩展版本
- Aurora PostgreSQL 兼容版 13 的扩展版本
- Aurora PostgreSQL 兼容版 12 的扩展版本
- Aurora PostgreSQL 兼容版 11 的扩展版本
- Aurora PostgreSQL 兼容版 10 的扩展版本
- Aurora PostgreSQL 兼容版 9.6 的扩展版本

将 PostGIS 2 升级到 PostGIS 3

从版本 3.0 开始，PostGIS 光栅功能现在是一个单独的扩展，即 `postgis_raster`。此扩展有自己的安装和升级路径。这将从核心 `postgis` 扩展中删除光栅图像处理所需的数十个函数、数据类型和其他构件。这意味着，如果您的使用案例不需要光栅处理，则不需要安装 `postgis_raster` 扩展。

在以下升级示例中，第一个升级命令将光栅功能提取到 `postgis_raster` 扩展。然后，需要使用第二个升级命令将 `postgis_raster` 升级到新版本。

从 PostGIS 2 升级到 PostGIS 3

1. 确定 Aurora PostgreSQL 数据库集群 上 PostgreSQL 版本可用的 PostGIS 的默认版本。为此，请运行以下查询。

```
SELECT * FROM pg_available_extensions
    WHERE default_version > installed_version;
  name   | default_version | installed_version |                          comment
---------+-----------------+-------------------
+------------------------------------------------------------
 postgis | 3.1.4           | 2.3.7             | PostGIS geometry and geography
 spatial types and functions
(1 row)
```

2. 确定 Aurora PostgreSQL 数据库集群的写入器实例上每个数据库中安装的 PostGIS 版本。换句话说，按如下方式查询每个用户数据库。

```
SELECT
    e.extname AS "Name",
    e.extversion AS "Version",
    n.nspname AS "Schema",
    c.description AS "Description"
FROM
```

```
    pg_catalog.pg_extension e
    LEFT JOIN pg_catalog.pg_namespace n ON n.oid = e.extnamespace
    LEFT JOIN pg_catalog.pg_description c ON c.objoid = e.oid
    AND c.classoid = 'pg_catalog.pg_extension'::pg_catalog.regclass
WHERE
    e.extname LIKE '%postgis%'
ORDER BY
    1;
  Name   | Version | Schema |                            Description
---------+---------+--------
+------------------------------------------------------------------------
 postgis | 2.3.7   | public | PostGIS geometry, geography, and raster spatial types
 and functions
(1 row)
```

原定设置版本（PostGIS 3.1.4）和已安装版本（PostGIS 2.3.7）之间的不匹配意味着您需要升级 PostGIS 扩展。

```
ALTER EXTENSION postgis UPDATE;
ALTER EXTENSION
WARNING: unpackaging raster
WARNING: PostGIS Raster functionality has been unpackaged
```

3. 运行以下查询，以验证光栅功能现在是否已包含在自己的程序包中。

```
SELECT
    probin,
    count(*)
FROM
    pg_proc
WHERE
    probin LIKE '%postgis%'
GROUP BY
    probin;
          probin          | count
--------------------------+-------
 $libdir/rtpostgis-2.3    | 107
 $libdir/postgis-3        | 487
(2 rows)
```

输出将显示版本之间仍然存在差异。PostGIS 函数为版本 3（postgis-3），而光栅函数（rtpostgis）为第 2 版（rtpostgis-2.3）。要完成升级，请再次运行升级命令，如下所示。

```
postgres=> SELECT postgis_extensions_upgrade();
```

您可以放心地忽略警告消息。再次运行以下查询，以验证升级已完成。当 PostGIS 和所有相关扩展未标记为需要升级时，升级即告完成。

```
SELECT postgis_full_version();
```

4. 使用以下查询查看已完成的升级过程和单独打包的扩展，并验证其版本是否匹配。

```
SELECT
    e.extname AS "Name",
    e.extversion AS "Version",
    n.nspname AS "Schema",
    c.description AS "Description"
FROM
    pg_catalog.pg_extension e
    LEFT JOIN pg_catalog.pg_namespace n ON n.oid = e.extnamespace
    LEFT JOIN pg_catalog.pg_description c ON c.objoid = e.oid
        AND c.classoid = 'pg_catalog.pg_extension'::pg_catalog.regclass
WHERE
    e.extname LIKE '%postgis%'
ORDER BY
    1;
      Name      | Version | Schema |                            Description
----------------+---------+--------
+------------------------------------------------------------------------
 postgis        | 3.1.5   | public | PostGIS geometry, geography, and raster
 spatial types and functions
 postgis_raster | 3.1.5   | public | PostGIS raster types and functions
(2 rows)
```

输出显示，PostGIS 2 扩展已升级到 PostGIS 3，并且 `postgis` 和现在独立的 `postgis_raster` 扩展都为版本 3.1.5。

升级完成后，如果您不打算使用光栅功能，则可以按如下方式删除此扩展。

```
DROP EXTENSION postgis_raster;
```

使用 pg_partman 扩展管理 PostgreSQL 分区

PostgreSQL 表分区为数据输入和报告的高性能处理提供了框架。对于需要非常快速地输入大量数据的数据库，请使用分区。分区还可以更快地查询大型表。分区有助于在不影响数据库实例的情况下维护数据，因为它需要的 I/O 资源较少。

通过使用分区，您可以将数据拆分为自定义大小的块进行处理。例如，您可以将时间序列数据分区为范围，例如每小时、每日、每周、每月、每季度、每年、自定义或这些范围的任意组合。对于时间序列数据示例，如果您按小时对表进行分区，则每个分区会包含一小时的数据。如果您按天对时间序列表进行分区，则分区会保存一天的数据，以此类推。分区键控制分区的大小。

在分区表上使用 `INSERT` 或 `UPDATE` SQL 命令时，数据库引擎会将数据路由到相应的分区。存储数据的 PostgreSQL 表分区是主表的子表。

在数据库查询读取期间，PostgreSQL 优化程序会检查查询的 `WHERE` 子句，如果可能的话，将数据库扫描定向到仅相关分区。

从版本 10 开始，PostgreSQL 使用声明性分区来实现表分区，这也称为本机 PostgreSQL 分区。在 PostgreSQL 版本 10 之前，使用触发器来实现分区。

PostgreSQL 表分区提供了以下功能：

- 随时创建新分区。
- 可变分区范围。
- 使用数据定义语言 (DDL) 语句可分离和可重新连接的分区。

  例如，可分离的分区对于从主分区中删除历史数据但保留历史数据以供分析很有用。
- 新分区继承了父数据库表的属性，包括以下属性：
  - 索引
  - 主键，其必须包括分区键列
  - 外键
  - 检查约束
  - 参考
- 为完整表或每个特定分区创建索引。

您不能更改单个分区的架构。但是，您可以更改传播到分区的父表（例如添加新列）。

主题

- PostgreSQL pg_partman 扩展概述
- 启用 pg_partman 扩展
- 使用 create_parent 函数配置分区
- 使用 run_maintenance_proc 函数配置分区维护

PostgreSQL pg_partman 扩展概述

您可以使用 PostgreSQL pg_partman 扩展自动创建和维护表分区。有关更多一般信息，请参阅 pg_partman 文档中的 PG 分区管理器。

> **Note**
>
> Aurora PostgreSQL 版本 12.6 及更高版本支持该 pg_partman 扩展。

您可以使用以下设置来配置 pg_partman，而不必手动创建每个分区：

- 要分区的表
- 分区类型
- 分区键
- 分区粒度
- 分区预创建和管理选项

创建 PostgreSQL 分区表后，您可以通过调用 create_parent 函数向 pg_partman 注册该表。此举会根据传递给函数的参数创建必要的分区。

该 pg_partman 扩展还提供了 run_maintenance_proc 函数，您可以按计划调用该函数来自动管理分区。为确保根据需要创建正确的分区，请将此函数计划为定期运行（例如每小时）。您还可以确保自动删除分区。

启用 pg_partman 扩展

如果要管理分区的同一 PostgreSQL 数据库实例中有多个数据库，请为每个数据库分别启用 pg_partman 扩展。要为特定数据库启用 pg_partman 扩展，请创建分区维护架构，然后按如下所示创建 pg_partman 扩展。

```
CREATE SCHEMA partman;
```

```
CREATE EXTENSION pg_partman WITH SCHEMA partman;
```

> **Note**
>
> 要创建 pg_partman 扩展，请确保您具有 rds_superuser 权限。

如果您收到以下错误，请向该账户授予 rds_superuser 权限或使用您的超级用户帐户。

```
ERROR: permission denied to create extension "pg_partman"
HINT: Must be superuser to create this extension.
```

要授予 rds_superuser 权限，请连接您的超级用户账户并运行以下命令。

```
GRANT rds_superuser TO user-or-role;
```

对于显示使用 pg_partman 扩展的示例，我们使用以下示例数据库表和分区。此数据库使用基于时间戳的分区表。架构 data_mart 包含名为 events 的表，当中包含名为 created_at 的列。events 表中包含以下设置：

- 主键 event_id 和 created_at，其必须具有用于指导分区的列。
- 用于强制 ck_valid_operation 表列值的检查约束 operation。
- 两个外键，其中一个 (fk_orga_membership)) 指向外部表 organization，另一个 (fk_parent_event_id) 是自引用的外键。
- 两个索引，其中一个 (idx_org_id) 用于外键，另一个 (idx_event_type) 用于事件类型。

以下 DDL 语句创建这些对象，这些对象自动包含在每个分区中。

```
CREATE SCHEMA data_mart;
CREATE TABLE data_mart.organization ( org_id BIGSERIAL,
        org_name TEXT,
        CONSTRAINT pk_organization PRIMARY KEY (org_id)
    );

CREATE TABLE data_mart.events(
        event_id        BIGSERIAL,
        operation       CHAR(1),
```

```
        value           FLOAT(24),
        parent_event_id BIGINT,
        event_type      VARCHAR(25),
        org_id          BIGSERIAL,
        created_at      timestamp,
        CONSTRAINT pk_data_mart_event PRIMARY KEY (event_id, created_at),
        CONSTRAINT ck_valid_operation CHECK (operation = 'C' OR operation = 'D'),
        CONSTRAINT fk_orga_membership
            FOREIGN KEY(org_id)
            REFERENCES data_mart.organization (org_id),
        CONSTRAINT fk_parent_event_id
            FOREIGN KEY(parent_event_id, created_at)
            REFERENCES data_mart.events (event_id,created_at)
    ) PARTITION BY RANGE (created_at);

CREATE INDEX idx_org_id     ON  data_mart.events(org_id);
CREATE INDEX idx_event_type ON  data_mart.events(event_type);
```

使用 create_parent 函数配置分区

启用 pg_partman 扩展后，使用 create_parent 函数在分区维护架构中配置分区。以下示例使用在 events 中创建的 启用 pg_partman 扩展 表示例。按如下方式调用 create_parent 函数。

```
SELECT partman.create_parent(
 p_parent_table => 'data_mart.events',
 p_control      => 'created_at',
 p_type         => 'range',
 p_interval     => '1 day',
 p_premake      => 30);
```

参数如下所示：

- p_parent_table – 父分区表。此表必须已存在并完全限定（包括架构在内）。
- p_control – 分区所依据的列。数据类型必须是整数或基于时间的。
- p_type – 该类型是 'range' 或者 'list'。
- p_interval – 每个分区的时间间隔或整数范围。示例值包括 1 day、1 hour 等。
- p_premake – 为支持新插入而提前创建的分区数量。

有关 create_parent 函数的完整说明，请参阅 pg_partman 文档中的创建函数。

使用 run_maintenance_proc 函数配置分区维护

您可以运行分区维护操作来自动创建新分区、分离分区或删除旧分区。分区维护依赖于 pg_partman 扩展和 pg_cron 扩展的 run_maintenance_proc 函数，其将启动内部调度程序。调度程序 pg_cron 自动执行数据库中定义的 SQL 语句、函数和程序。

以下示例使用在 events 中创建的 启用 pg_partman 扩展 表示例将分区维护操作设置为自动运行。作为先决条件，请将 pg_cron 添加到数据库实例的参数组中的 shared_preload_libraries 参数。

```
CREATE EXTENSION pg_cron;

UPDATE partman.part_config
SET infinite_time_partitions = true,
    retention = '3 months',
    retention_keep_table=true
WHERE parent_table = 'data_mart.events';
SELECT cron.schedule('@hourly', $$CALL partman.run_maintenance_proc()$$);
```

以下是前述示例的分步说明：

1. 修改与数据库实例关联的参数组并将 pg_cron 添加到 shared_preload_libraries 参数值中。此更改需要重启数据库实例才能生效。有关更多信息，请参阅“在 Amazon Aurora 中修改数据库参数组中的参数”。
2. CREATE EXTENSION pg_cron; 使用具有 rds_superuser 权限的账户运行此命令。这将启用 pg_cron 扩展。有关更多信息，请参阅“使用 PostgreSQL pg_cron 扩展计划维护”。
3. 运行命令 UPDATE partman.part_config 以调整 data_mart.events 表的 pg_partman 设置。
4. 运行命令 SET... 配置 data_mart.events 表，其中包含以下子句：
   a. infinite_time_partitions = true, – 将表配置为能够在没有任何限制的情况下自动创建新分区。
   b. retention = '3 months', – 将表配置为最长保留三个月。
   c. retention_keep_table=true – 对表进行配置，以便在保留期到期时表不会自动删除。相反，早于保留期的分区只能从父表中分离。
5. 运行命令 SELECT cron.schedule... 创建一个 pg_cron 函数调用。此调用定义了计划程序运行 pg_partman 维护程序的频率，partman.run_maintenance_proc。对于此示例，该程序每小时运行一次。

有关 run_maintenance_proc 函数的完整说明，请参阅 pg_partman 文档中的维护函数。

使用 PostgreSQL pg_cron 扩展计划维护

您可以使用 PostgreSQL pg_cron 扩展来计划 PostgreSQL 数据库中的维护命令。有关扩展的更多信息，请参阅 pg_cron 文档中的什么是 pg_cron？

Aurora PostgreSQL 引擎版本 12.6 及更高版本支持 pg_cron 扩展

要了解有关使用 pg_cron 的更多信息，请参阅在 RDS for PostgreSQL 或 Aurora PostgreSQL 兼容版数据库上使用 pg_cron 计划任务。

主题

- 设置 pg_cron 扩展
- 授予数据库用户使用 pg_cron 的权限
- 计划 pg_cron 作业
- pg_cron 扩展的参考

设置 pg_cron 扩展

按如下方式设置 pg_cron 扩展：

1. 通过向 shared_preload_libraries 参数值添加 pg_cron，修改与 PostgreSQL 数据库实例关联的自定义参数组。

   重新启动 PostgreSQL 数据库实例，以使对参数组的更改生效。要了解有关使用参数组的更多信息，请参阅Amazon Aurora PostgreSQL 参数。
2. 重新启动 PostgreSQL 数据库实例后，使用具有 rds_superuser 权限的账户运行以下命令。例如，如果在创建 Aurora PostgreSQL 数据库实例时使用默认设置，请以用户 postgres 身份进行连接，然后创建扩展。

   ```
   CREATE EXTENSION pg_cron;
   ```

   pg_cron 调度程序是在名为 postgres 的默认 PostgreSQL 数据库中设置的。这些 pg_cron 对象是在此 postgres 数据库中创建的，所有调度操作都在此数据库中运行。
3. 您可以使用默认设置，也可以计划作业在 PostgreSQL 数据库实例的其他数据库中运行。要为 PostgreSQL 数据库实例中的其他数据库计划作业，请参阅 为原定设置数据库以外的数据库计划 cron 任务 中的示例。

授予数据库用户使用 pg_cron 的权限

安装 pg_cron 扩展需要 rds_superuser 权限。但是，可以（由 pg_cron 组/角色的成员）将使用 rds_superuser 的权限授予其他数据库用户，以便他们可以计划自己的任务。我们建议您仅在需要时才授予对 cron 架构的权限，前提是它可以改进生产环境中的操作。

要在 cron 架构中授予数据库用户权限，请运行以下命令：

```
postgres=> GRANT USAGE ON SCHEMA cron TO db-user;
```

这向 *db-user* 授予访问 cron 架构的权限，以便为他们有权限访问的对象计划 cron 任务。如果数据库用户没有权限，则在将错误消息发布到 postgresql.log 文件后，任务会失败，如下所示：

```
2020-12-08 16:41:00 UTC::@:[30647]:ERROR: permission denied for table table-name
2020-12-08 16:41:00 UTC::@:[27071]:LOG: background worker "pg_cron" (PID 30647) exited
 with exit code 1
```

换言之，请确保被授予对 cron 架构的权限的数据库用户也对他们计划安排的对象（表、架构等）拥有权限。

cron 任务的详细信息及其成功或失败情况也在 cron.job_run_details 表中捕获。有关更多信息，请参阅 用于计划任务和捕获状态的表 。

计划 pg_cron 作业

以下各节介绍了如何使用 pg_cron 作业安排各种管理任务。

> **Note**
>
> 创建 pg_cron 任务时，请检查 max_worker_processes 设置是否大于 cron.max_running_jobs 的数量。如果 pg_cron 任务耗尽后台工作进程，它将失败。原定设置的 pg_cron 任务数量为 5。有关更多信息，请参阅 用于管理 pg_cron 扩展的参数。

主题

- 对表执行清理操作
- 清除 pg_cron 历史记录表
- 仅将错误记录到 postgresql.log 文件中

- 为原定设置数据库以外的数据库计划 cron 任务

对表执行清理操作

Autovacuum 在大多数情况下处理清理维护。但是，您可能希望在选择的时间计划对特定表执行清理操作。

以下示例介绍了使用 cron.schedule 函数设置作业，以便每天 22:00 (GMT) 在特定表上使用 VACUUM FREEZE。

```
SELECT cron.schedule('manual vacuum', '0 22 * * *', 'VACUUM FREEZE pgbench_accounts');
 schedule
----------
1
(1 row)
```

运行上述示例之后，您可以按如下方式检查 cron.job_run_details 表中的历史记录。

```
postgres=> SELECT * FROM cron.job_run_details;
jobid  | runid | job_pid | database | username | command                        |
 status    | return_message | start_time                    | end_time
-------+-------+---------+----------+----------+--------------------------------
+-----------+----------------+-------------------------------
+-------------------------------
 1     | 1     | 3395    | postgres | adminuser| vacuum freeze pgbench_accounts
 | succeeded | VACUUM         | 2020-12-04 21:10:00.050386+00 | 2020-12-04
 21:10:00.072028+00
(1 row)
```

下面的内容说明如何查询 cron.job_run_details 表以查看失败的任务。

```
postgres=> SELECT * FROM cron.job_run_details WHERE status = 'failed';
jobid | runid | job_pid | database | username | command                        | status
 | return_message                                  | start_time                    |
 end_time
------+-------+---------+----------+----------+--------------------------------+--------
+-------------------------------------------------+-------------------------------
+-------------------------------
 5    | 4     | 30339   | postgres | adminuser| vacuum freeze pgbench_account | failed
 | ERROR: relation "pgbench_account" does not exist | 2020-12-04 21:48:00.015145+00 |
 2020-12-04 21:48:00.029567+00
```

```
(1 row)
```

有关更多信息，请参阅 用于计划任务和捕获状态的表 。

清除 pg_cron 历史记录表

cron.job_run_details 表包含 cron 作业的历史记录，随着时间的推移，这些历史记录可能会变得非常大。我们建议您计划清除此表的作业。例如，保留一周的条目可能足以进行故障排除。

以下示例使用 cron.schedule 函数计划每天午夜运行以清除 cron.job_run_details 表的作业。这项工作只保留了过去七天的历史记录。使用您的 rds_superuser 账户计划作业，如下所示。

```
SELECT cron.schedule('0 0 * * *', $$DELETE
    FROM cron.job_run_details
    WHERE end_time < now() - interval '7 days'$$);
```

有关更多信息，请参阅 用于计划任务和捕获状态的表 。

仅将错误记录到 postgresql.log 文件中

要防止向 cron.job_run_details 表中进行写入，请修改与 PostgreSQL 数据库实例关联的参数组，然后将 cron.log_run 参数设置为 off。pg_cron 扩展不再写入表，只会将错误捕获到 postgresql.log 文件中。有关更多信息，请参阅 在 Amazon Aurora 中修改数据库参数组中的参数。

使用以下命令检查 cron.log_run 参数的值。

```
postgres=> SHOW cron.log_run;
```

有关更多信息，请参阅 用于管理 pg_cron 扩展的参数。

为原定设置数据库以外的数据库计划 cron 任务

pg_cron 的元数据全部保存在名为 postgres 的 PostgreSQL 默认数据库中。由于后台工件用于运行维护 cron 作业，因此您可以在 PostgreSQL 数据库实例中的任何数据库中计划作业。

> **Note**
>
> 只有具有 rds_superuser 角色或 rds_superuser 权限的用户才能列出数据库中的所有 cron 作业。其他用户只能在 cron.job 表中查看其自己的作业。

1. 在 cron 数据库中，以与平常使用 cron.schedule 相同的方式计划作业。

```
postgres=> SELECT cron.schedule('database1 manual vacuum', '29 03 * * *', 'vacuum
 freeze test_table');
```

2. 作为具有 rds_superuser 角色的用户，请更新刚创建的作业的数据库列，使其在 PostgreSQL 数据库实例中的另一个数据库中运行。

```
postgres=> UPDATE cron.job SET database = 'database1' WHERE jobid = 106;
```

3. 通过查询 cron.job 表进行验证。

```
postgres=> SELECT * FROM cron.job;
jobid | schedule    | command                        | nodename  | nodeport |
 database | username  | active | jobname
------+-------------+--------------------------------+-----------+----------
+----------+-----------+--------+------------------------
106   | 29 03 * * * | vacuum freeze test_table       | localhost | 8192     |
 database1| adminuser | t      | database1 manual vacuum
  1   | 59 23 * * * | vacuum freeze pgbench_accounts | localhost | 8192     |
 postgres | adminuser | t      | manual vacuum
(2 rows)
```

> **Note**
>
> 在某些情况下，您可以添加打算在其他数据库上运行的 cron 作业。在这些情况下，在您更新正确的数据库列之前，该作业可能会尝试在默认数据库 (postgres) 中运行。如果用户名具有权限，则作业将在默认数据库中成功运行。

pg_cron 扩展的参考

您可以将以下参数、函数和表与 pg_cron 扩展搭配使用。有关更多信息，请参阅 pg_cron 文档中的什么是 pg_cron？。

主题

- 用于管理 pg_cron 扩展的参数
- 函数参考：cron.schedule
- 函数参考：cron.unschedule

- 用于计划任务和捕获状态的表

用于管理 pg_cron 扩展的参数

以下是用于控制 pg_cron 扩展行为的参数列表。

参数	描述
cron.database_name	保存 pg_cron 元数据的数据库。
cron.host	要连接到 PostgreSQL 的主机名。您无法修改此值。
cron.log_run	在 job_run_details 表中记录运行的每个任务。值为 on 或 off。有关更多信息，请参阅“用于计划任务和捕获状态的表”。
cron.log_statement	在运行所有 cron 语句之前将其记入日志。值为 on 或 off。
cron.max_running_jobs	可以同时运行的最大作业数。
cron.use_background_workers	使用后台工作程序而不是客户端会话。您无法修改此值。

使用以下 SQL 命令来显示这些参数及其值。

```
postgres=> SELECT name, setting, short_desc FROM pg_settings WHERE name LIKE 'cron.%'
 ORDER BY name;
```

函数参考：cron.schedule

此函数计划 cron 作业。作业最初是在默认 postgres 数据库中计划的。该函数返回一个表示作业标识符的 bigint 值。要计划作业在 PostgreSQL 数据库实例的其他数据库中运行，请参阅 为原定设置数据库以外的数据库计划 cron 任务 中的示例。

该函数有两种语法格式。

语法

```
cron.schedule (job_name,
    schedule,
    command
);

cron.schedule (schedule,
    command
);
```

参数

参数	描述
job_name	cron 作业的名字。
schedule	表示 cron 作业时间表的文本。格式是标准 cron 格式。
command	要运行的命令的文本。

示例

```
postgres=> SELECT cron.schedule ('test','0 10 * * *', 'VACUUM pgbench_history');
 schedule
----------
      145
(1 row)

postgres=> SELECT cron.schedule ('0 15 * * *', 'VACUUM pgbench_accounts');
 schedule
----------
      146
(1 row)
```

函数参考：cron.unschedule

此函数删除 cron 作业。您可以指定 job_name 或 job_id。策略可以确保您是删除作业计划的拥有者。该函数返回一个布尔值，指示成功或失败。

该函数使用以下语法格式。

语法

```
cron.unschedule (job_id);

cron.unschedule (job_name);
```

参数

参数	描述
job_id	计划 cron 作业时从 cron.schedule 函数返回的作业标识符。
job_name	使用该 cron.schedule 函数计划的 cron 作业的名称。

示例

```
postgres=> SELECT cron.unschedule(108);
 unschedule
------------
 t
(1 row)

postgres=> SELECT cron.unschedule('test');
 unschedule
------------
 t
(1 row)
```

用于计划任务和捕获状态的表

将以下各表用于计划 cron 作业和记录作业完成的方式。

表	描述
`cron.job`	包含有关每个计划作业的元数据。与此表的大多数交互应使用 `cron.schedule` 和 `cron.unschedule` 函数完成。 **Important** 我们不建议直接授予对此表的更新或插入权限。这样做将允许用户更新 `username` 列，从而以 `rds-superuser` 身份运行。
`cron.job_run_details`	包含过去运行的计划作业的历史信息。这对于调查运行的作业的状态、返回消息以及开始和结束时间非常有用。 **Note** 为了防止此表无限增长，请定期清除此表。有关示例，请参阅清除 pg_cron 历史记录表。

使用 pgAudit 记录数据库活动

金融机构、政府机构和许多行业需要保留审计日志以满足监管要求。通过将 PostgreSQL Audit 扩展（pgAudit）与 Aurora PostgreSQL 数据库集群结合使用，您可以捕获审计人员通常需要或满足监管要求的详细记录。例如，您可以设置 pgAudit 扩展来跟踪对特定数据库和表所做的更改，记录进行更改的用户以及许多其他详细信息。

pgAudit 扩展通过更详细地扩展日志消息，进一步构建原生 PostgreSQL 日志记录基础设施的功能。换句话说，您将使用与查看任何日志消息相同的方法来查看审计日志。有关 PostgreSQL 日志记录的更多信息，请参阅 Aurora PostgreSQL 数据库日志文件。

pgAudit 扩展会编辑日志中的敏感数据，例如明文密码。如果您的 Aurora PostgreSQL 数据库集群配置为记录数据操作语言（DML）语句（详情请见为您的 Aurora PostgreSQL 数据库集群开启查询日志记录），则可以使用 PostgreSQL Audit 扩展来避免明文密码问题。

您可以在数据库实例上配置具有高度明确性的审计。您可以审计所有数据库和所有用户。或者，您可以选择仅审计某些数据库、用户和其他对象。您也可以明确排除对某些用户和数据库进行审计。有关更多信息，请参阅 从审计日志记录中排除用户或数据库。

考虑到可以捕获的详细信息量，我们建议您在使用 pgAudit 时监控存储消耗。

所有可用的 Aurora PostgreSQL 版本都支持 pgAudit 扩展。有关 Aurora PostgreSQL 版本支持的 pgAudit 版本的列表，请参阅《Aurora PostgreSQL 版本注释》https://docs.aws.amazon.com/AmazonRDS/latest/AuroraPostgreSQLReleaseNotes/AuroraPostgreSQL.Extensions.html中的 Amazon Aurora PostgreSQL 的扩展版本。。

主题

- 设置 pgAudit 扩展
- 审计数据库对象
- 从审计日志记录中排除用户或数据库
- pgAudit 扩展的参考

设置 pgAudit 扩展

要在 Aurora PostgreSQL 数据库集群上设置 pgAudit 扩展，首先要将 pgAudit 添加到 Aurora PostgreSQL 数据库集群的自定义数据库集群参数组上的共享库中。有关创建自定义数据库集群参数组的信息，请参阅Amazon Aurora 的参数组。接下来，安装 pgAudit 扩展。最后，指定要审计的数据库和对象。本部分中的过程向您展示如何操作。您可以使用 AWS Management Console或 AWS CLI。

您必须拥有 `rds_superuser` 角色的权限才能执行所有这些任务。

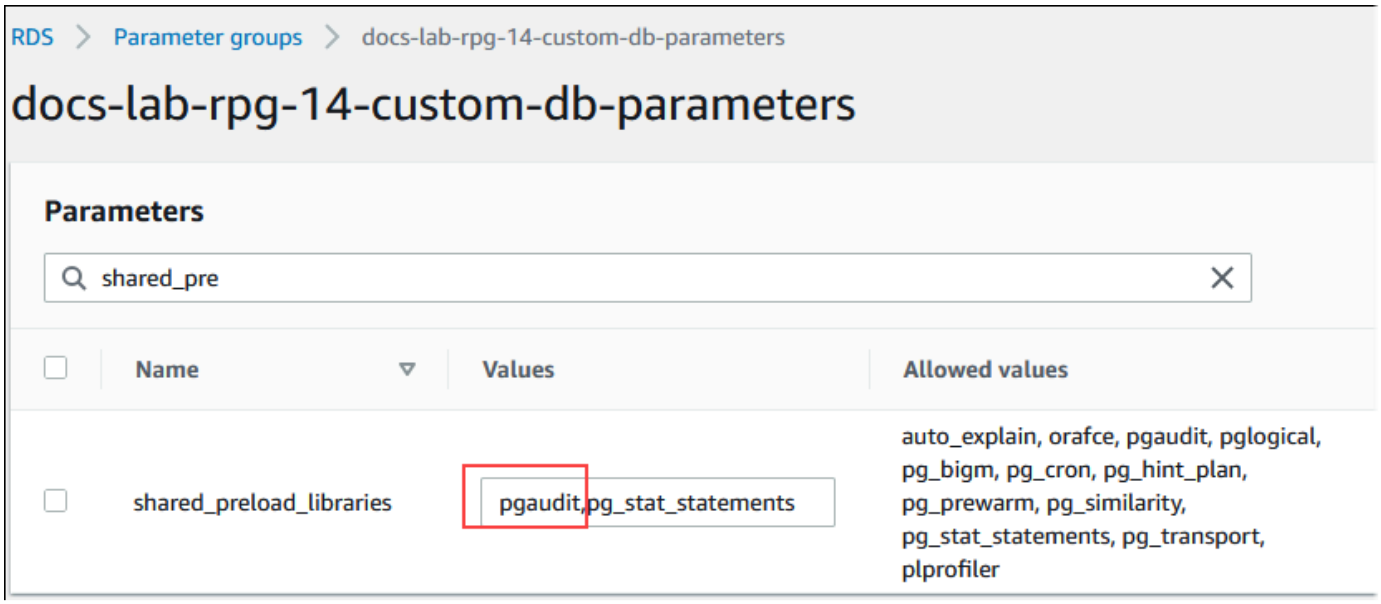
以下步骤假设您的 Aurora PostgreSQL 数据库集群 与自定义数据库集群 参数组相关联。

控制台

设置 pgAudit 扩展

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Aurora PostgreSQL 数据库集群的写入器实例 。
3. 打开 Aurora PostgreSQL 数据库集群写入器实例的配置选项卡。的 Configuration（配置）选项卡。在实例详细信息中，找到 Parameter group（参数组）链接。
4. 选择此链接以打开与您的 Aurora PostgreSQL 数据库集群 关联的自定义参数。
5. 在 Parameters（参数）搜索字段中，键入 `shared_pre` 以查找 `shared_preload_libraries` 参数。
6. 选择 Edit parameters（编辑参数）以访问属性值。

7. 将 pgaudit 添加到 Values（值）字段的列表中。使用逗号分隔值列表中的项目。



8. 重启 Aurora PostgreSQL 数据库集群的写入器实例，以使对 shared_preload_libraries 参数的更改生效。

9. 当实例可用时，请验证 pgAudit 是否已初始化。使用 psql 连接到 Aurora PostgreSQL 数据库集群的写入器实例，然后运行以下命令。

```
SHOW shared_preload_libraries;
shared_preload_libraries
--------------------------
rdsutils,pgaudit
(1 row)
```

10. 初始化 pgAudit 后，您现在可以创建扩展了。您需要在初始化库后创建扩展，因为 pgaudit 扩展会为审计数据定义语言（DDL）语句安装事件触发器。

```
CREATE EXTENSION pgaudit;
```

11. 关闭 psql 会话。

```
labdb=> \q
```

12. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。

13. 在列表中找到 pgaudit.log 参数并设置为适合您的使用案例的值。例如，将 pgaudit.log 参数设置为 write（如下图所示），可以捕获对日志的插入、更新、删除和其他一些类型的更改。



还可以为 `pgaudit.log` 参数选择以下值之一。

- none – 这是默认值。不记录任何数据库更改。
- all – 记录所有内容（read、write、function、role、ddl、misc）。
- ddl – 记录所有数据定义语言（DDL）语句（不包括在 `ROLE` 类中）。
- function – 记录函数调用和 `DO` 块。
- misc – 记录其他命令，例如 `DISCARD`、`FETCH`、`CHECKPOINT`、`VACUUM` 和 `SET`。
- read –当源为关系（例如表）或查询时记录 `SELECT` 和 `COPY`。
- role – 记录与角色和权限相关的语句，例如 `GRANT`、`REVOKE`、`CREATE ROLE`、`ALTER ROLE` 和 `DROP ROLE`。
- write – 当目标为关系（表）时，记录 `INSERT`、`UPDATE`、`DELETE`、`TRUNCATE` 和 `COPY`。

14. 选择 Save changes（保存更改）。
15. 通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
16. 从“数据库”列表中选择 Aurora PostgreSQL 数据库集群的写入器实例。

AWS CLI

设置 pgAudit

要使用 AWS CLI 设置 pgAudit，您可以调用 modify-db-parameter-group 操作来修改自定义参数组中的审计日志参数，如以下过程所示。

1. 使用以下 AWS CLI 命令向 `shared_preload_libraries` 参数中添加 `pgaudit`。

```
aws rds modify-db-parameter-group \
   --db-parameter-group-name custom-param-group-name \
   --parameters
 "ParameterName=shared_preload_libraries,ParameterValue=pgaudit,ApplyMethod=pending-
reboot" \
   --region aws-region
```

2. 使用以下 AWS CLI 命令重启 Aurora PostgreSQL 数据库集群的写入器实例，以便初始化 pgaudit 库。

```
aws rds reboot-db-instance \
    --db-instance-identifier writer-instance \
    --region aws-region
```

3. 当实例可用时，您可以验证 pgaudit 是否已初始化。使用 psql 连接到 Aurora PostgreSQL 数据库集群的写入器实例，然后运行以下命令。

```
SHOW shared_preload_libraries;
shared_preload_libraries
--------------------------
rdsutils,pgaudit
(1 row)
```

初始化 pgAudit 后，您现在可以创建扩展了。

```
CREATE EXTENSION pgaudit;
```

4. 关闭 psql 会话以便您可以使用 AWS CLI。

```
labdb=> \q
```

5. 使用以下 AWS CLI 命令指定要由会话审计日志记录所记录的语句类别。该示例将 pgaudit.log 参数设置为 write，用于捕获对日志的插入、更新和删除。

```
aws rds modify-db-parameter-group \
   --db-parameter-group-name custom-param-group-name \
   --parameters
 "ParameterName=pgaudit.log,ParameterValue=write,ApplyMethod=pending-reboot" \
   --region aws-region
```

还可以为 pgaudit.log 参数选择以下值之一。

- none – 这是默认值。不记录任何数据库更改。
- all – 记录所有内容（read、write、function、role、ddl、misc）。
- ddl – 记录所有数据定义语言（DDL）语句（不包括在 ROLE 类中）。
- function – 记录函数调用和 DO 块。
- misc – 记录其他命令，例如 DISCARD、FETCH、CHECKPOINT、VACUUM 和 SET。
- read –当源为关系（例如表）或查询时记录 SELECT 和 COPY。
- role – 记录与角色和权限相关的语句，例如 GRANT、REVOKE、CREATE ROLE、ALTER ROLE 和 DROP ROLE。
- write – 当目标为关系（表）时，记录 INSERT、UPDATE、DELETE、TRUNCATE 和 COPY。

使用以下 AWS CLI 命令重启 Aurora PostgreSQL 数据库集群的写入器实例。

```
aws rds reboot-db-instance \
    --db-instance-identifier writer-instance \
    --region aws-region
```

审计数据库对象

在 Aurora PostgreSQL 数据库集群 上设置 pgAudit 并根据您的要求进行配置后，将在 PostgreSQL 日志中捕获更多详细信息。例如，虽然默认 PostgreSQL 日志记录配置标识数据库表中发生更改的日期和时间，但使用 pgAudit 扩展后，日志条目可以包括模式、进行更改的用户和其他详细信息，具体取决于扩展参数的配置方式。您可以设置审计以通过以下方法跟踪更改。

- 对于每个会话，按用户进行跟踪。对于会话级别，您可以捕获完全限定的命令文本。
- 对于每个对象，按用户和数据库进行跟踪。

当您在系统上创建 rds_pgaudit 角色，然后将此角色添加到自定义参数组中的 pgaudit.role 参数时，将激活对象审计功能。默认情况下，pgaudit.role 参数处于未设置状态，唯一允许的值是 rds_pgaudit。以下步骤假设 pgaudit 已初始化，并且您已按照设置 pgAudit 扩展中的步骤创建了 pgaudit 扩展。

```
2022-10-07 23:36:51 UTC:52.95.4.10(14410):postgres@labdb:[1374]:LOG: statement: SELECT feedback, s.sentiment,s.confidence
FROM support,aws_comprehend.detect_sentiment(feedback, 'en') s
ORDER BY s.confidence DESC;
2022-10-07 23:36:51 UTC:52.95.4.10(14410):postgres@labdb:[1374]:LOG: AUDIT: SESSION,2,1,READ,SELECT,TABLE,public.support,"SELECT
feedback, s.sentiment,s.confidence
FROM support,aws_comprehend.detect_sentiment(feedback, 'en') s
ORDER BY s.confidence DESC;",<none>
2022-10-07 23:36:51 UTC:52.95.4.10(14410):postgres@labdb:[1374]:LOG: QUERY STATISTICS
2022-10-07 23:36:51 UTC:52.95.4.10(14410):postgres@labdb:[1374]:DETAIL: ! system usage stats:
! 0.009494 s user, 0.007442 s system, 0.141985 s elapsed
! [0.022327 s user, 0.007442 s system total]
```

如本示例所示，“LOG: AUDIT: SESSION”行提供了有关表及其架构的信息以及其他详细信息。

设置对象审计

1. 使用 psql 连接到 Aurora PostgreSQL 数据库集群的写入器实例。

   ```
   psql --host=your-instance-name.aws-region.rds.amazonaws.com --port=5432 --
   username=postgrespostgres --password --dbname=labdb
   ```

2. 使用以下命令创建名为 rds_pgaudit 的数据库角色。

   ```
   labdb=> CREATE ROLE rds_pgaudit;
   CREATE ROLE
   labdb=>
   ```

3. 关闭 psql 会话。

   ```
   labdb=> \q
   ```

   在接下来的几步中，使用 AWS CLI 修改自定义参数组中的审计日志参数。

4. 使用以下 AWS CLI 命令将 pgaudit.role 参数设置为 rds_pgaudit。默认情况下，此参数为空，rds_pgaudit 是唯一允许的值。

   ```
   aws rds modify-db-parameter-group \
      --db-parameter-group-name custom-param-group-name \
      --parameters
    "ParameterName=pgaudit.role,ParameterValue=rds_pgaudit,ApplyMethod=pending-reboot"
    \
      --region aws-region
   ```

5. 使用以下 AWS CLI 命令重启 Aurora PostgreSQL 数据库集群的写入器实例，以使对参数的更改生效。

```
aws rds reboot-db-instance \
    --db-instance-identifier writer-instance \
    --region aws-region
```

6. 运行以下命令确认 pgaudit.role 设置为 rds_pgaudit。

```
SHOW pgaudit.role;
pgaudit.role
-----------------
rds_pgaudit
```

要测试 pgAudit 日志记录，您可以运行几条要审计的示例命令。例如，您可以运行以下命令。

```
CREATE TABLE t1 (id int);
GRANT SELECT ON t1 TO rds_pgaudit;
SELECT * FROM t1;
id
----
(0 rows)
```

数据库日志应包含类似于以下内容的条目。

```
...
2017-06-12 19:09:49 UTC:...:rds_test@postgres:[11701]:LOG: AUDIT:
OBJECT,1,1,READ,SELECT,TABLE,public.t1,select * from t1;
...
```

有关查看日志的信息，请参阅监控 Amazon Aurora 日志文件。

要了解关于 pgAudit 扩展的更多信息，请参阅 GitHub 上的 pgAudit。

从审计日志记录中排除用户或数据库

如 Aurora PostgreSQL 数据库日志文件中所述，PostgreSQL 日志会消耗存储空间。使用 pgAudit 扩展会在不同程度上增加日志中收集的数据量，具体取决于您跟踪的更改。您可能不需要审计 Aurora PostgreSQL 数据库集群 中的每个用户或数据库。

为了最大限度地减少对存储的影响，避免不必要地捕获审计记录，您可以将用户和数据库排除在审计范围之外。您还可以在给定会话中更改日志记录。下面的示例向您演示如何操作。

> **Note**
>
> 会话级别的参数设置优先于 Aurora PostgreSQL 数据库集群的写入器实例的自定义数据库参数组中的设置。如果您不希望数据库用户绕过您的审计日志记录配置设置，请务必更改其权限。

假设您的 Aurora PostgreSQL 数据库集群 配置为审计所有用户和数据库的相同级别的活动。然后，您决定不想对用户 myuser 进行审计。您可以使用以下 SQL 命令对 myuser 关闭审计功能。

```
ALTER USER myuser SET pgaudit.log TO 'NONE';
```

然后，您可以使用以下查询来检查 pgaudit.log 的 user_specific_settings 列，以确认该参数已设置为 NONE。

```
SELECT
    usename AS user_name,
    useconfig AS user_specific_settings
FROM
    pg_user
WHERE
    usename = 'myuser';
```

您将看到如下输出。

```
 user_name | user_specific_settings
-----------+------------------------
 myuser    | {pgaudit.log=NONE}
(1 row)
```

在给定用户与数据库的会话期间，您可以使用以下命令对此用户关闭日志记录功能。

```
ALTER USER myuser IN DATABASE mydatabase SET pgaudit.log TO 'none';
```

使用以下查询，对于特定用户和数据库组合检查 pgaudit.log 的设置列。

```
SELECT
    usename AS "user_name",
    datname AS "database_name",
```

```
    pg_catalog.array_to_string(setconfig, E'\n') AS "settings"
FROM
    pg_catalog.pg_db_role_setting s
    LEFT JOIN pg_catalog.pg_database d ON d.oid = setdatabase
    LEFT JOIN pg_catalog.pg_user r ON r.usesysid = setrole
WHERE
    usename = 'myuser'
    AND datname = 'mydatabase'
ORDER BY
    1,
    2;
```

您将看到类似以下内容的输出。

```
  user_name | database_name |     settings
-----------+---------------+------------------
 myuser    | mydatabase    | pgaudit.log=none
(1 row)
```

对 myuser 关闭审计后，您决定不想跟踪对 mydatabase 的更改。您可以使用以下命令对该特定数据库关闭审计。

```
ALTER DATABASE mydatabase SET pgaudit.log to 'NONE';
```

然后，使用以下查询检查 database_specific_settings 列，以确认 pgaudit.log 已设置为 NONE。

```
SELECT
a.datname AS database_name,
b.setconfig AS database_specific_settings
FROM
pg_database a
FULL JOIN pg_db_role_setting b ON a.oid = b.setdatabase
WHERE
a.datname = 'mydatabase';
```

您将看到如下输出。

```
 database_name | database_specific_settings
---------------+----------------------------
 mydatabase    | {pgaudit.log=NONE}
```

```
(1 row)
```

要将 myuser 的设置恢复为默认设置，请使用以下命令：

```
ALTER USER myuser RESET pgaudit.log;
```

要将数据库的设置恢复为默认设置，请使用以下命令。

```
ALTER DATABASE mydatabase RESET pgaudit.log;
```

要将用户和数据库重置为默认设置，请使用以下命令。

```
ALTER USER myuser IN DATABASE mydatabase RESET pgaudit.log;
```

还可以通过将 pgaudit.log 设置为 pgaudit.log 参数的其他允许值之一，将特定事件捕获到日志中。有关更多信息，请参阅 pgaudit.log 参数允许的设置列表。

```
ALTER USER myuser SET pgaudit.log TO 'read';
ALTER DATABASE mydatabase SET pgaudit.log TO 'function';
ALTER USER myuser IN DATABASE mydatabase SET pgaudit.log TO 'read,function'
```

pgAudit 扩展的参考

您可以通过更改本节中列出的一个或多个参数来为审计日志指定所需的详细级别。

控制 pgAudit 行为

您可以通过更改下表中列出的一个或多个参数来控制审计日志记录。

参数	描述
pgaudit.log	指定会话审计日志记录将记录哪些语句类。允许的值包括 ddl、function、misc、read、role、write、none、all。有关更多信息，请参阅 pgaudit.log 参数允许的设置列表。
pgaudit.log_catalog	启用（设置为 1）时，如果语句中的所有关系都在 pg_catalog 中，则将语句添加到审计跟踪中。

参数	描述
pgaudit.log_level	指定要用于日志条目的日志级别。允许的值：debug5、debug4、debug3、debug2、debug1、info、notice、warning、log
pgaudit.log_parameter	启用（设置为 1）时，将在审计日志中捕获随语句传递的参数。
pgaudit.log_relation	启用（设置为 1）时，会话的审计日志为 SELECCT 或 DML 语句中引用的每个关系（TABLE、VIEW 等）创建单独的日志条目。
pgaudit.log_statement_once	指定日志记录在语句/子语句组合中的第一个日志条目中包含语句文本和参数，还是在每个条目中都包含。
pgaudit.role	指定用于对象审计日志记录的主角色。唯一允许的条目是 rds_pgaudit 。

pgaudit.log 参数允许的设置列表

值	描述
none	这是默认模式。不记录任何数据库更改。
all	记录所有内容（read、write、function、role、ddl、misc）。
ddl	记录所有数据定义语言（DDL）语句（不包括在 ROLE 类中）。
函数	记录函数调用和 DO 块。
misc	记录其他命令，例如 DISCARD、FETCH、CHECKPOINT 、VACUUM 和 SET。
read	当源为关系（例如表）或查询时记录 SELECT 和 COPY。
role	记录与角色和权限相关的语句，例如 GRANT、REVOKE、CREATE ROLE、ALTER ROLE 和 DROP ROLE。

值	描述
写入	当目标为关系（表）时，记录 INSERT、UPDATE、DELETE、TRUNCATE 和 COPY。

要使用会话审计记录多种事件类型，请使用逗号分隔的列表。要记录所有事件类型，请将 pgaudit.log 设置为 ALL。重启数据库实例以应用更改。

通过对象审计，您可以细化审计日志记录以使用特定的关系。例如，您可以指定要对一个或多个表上的 READ 操作进行审计日志记录。

使用 pglogical 跨实例同步数据

所有当前可用的 Aurora PostgreSQL 版本都支持 pglogical 扩展。pglogic 扩展早于 PostgreSQL 在版本 10 中引入的功能类似的逻辑复制特征。有关更多信息，请参阅使用 Aurora 进行 PostgreSQL 逻辑复制的概述。

pglogical 扩展支持在两个或更多 Aurora PostgreSQL 数据库集群 之间进行逻辑复制。它还支持在不同的 PostgreSQL 版本之间进行复制，以及在 RDS for PostgreSQL 数据库实例和 Aurora PostgreSQL 数据库集群上运行的数据库之间进行复制。pglogical 扩展使用发布-订阅模型将对表和其他对象（例如序列）的更改从发布者复制到订阅者。它依赖于复制插槽来确保更改从发布者节点同步到订阅者节点，定义如下。

- 发布者节点是作为要复制到其他节点的数据来源的 Aurora PostgreSQL 数据库集群。发布者节点定义要在发布集中复制的表。
- 订阅者节点是用于接收来自发布商的 WAL 更新的 Aurora PostgreSQL 数据库集群。订阅者创建订阅以连接到发布者并获取解码后的 WAL 数据。订阅者创建订阅时，将在发布者节点上创建复制插槽。

在下文中，您可以了解有关设置 pglogical 扩展的信息。

主题

- pglogical 扩展的要求和限制
- 设置 pglogical 扩展
- 为 Aurora PostgreSQL 数据库集群设置逻辑复制
- 在主要升级后重新建立逻辑复制
- 管理 Aurora PostgreSQL 的逻辑复制查槽

- pglogical 扩展的参数参考

pglogical 扩展的要求和限制

所有当前可用的 Aurora PostgreSQL 版本都支持 pglogical 扩展。

发布者节点和订阅者节点都必须设置为进行逻辑复制。

要从发布者复制到订阅用户的表必须具有相同的名称和相同的架构。这些表还必须包含相同的列，并且这些列必须使用相同的数据类型。发布者表和订阅者表必须具有相同的主键。我们建议您仅使用 PRIMARY KEY 作为唯一约束。

对于 CHECK 约束和 NOT NULL 约束，订阅者节点上的表可能比发布者节点上的表具有更宽松的约束。

pglogical 扩展提供了诸如双向复制之类的特征，PostgreSQL（版本 10 及更高版本）中内置的逻辑复制特征不支持这些特征。有关更多信息，请参阅使用 pglogic 进行 PostgreSQL 双向复制。

设置 pglogical 扩展

要在 Aurora PostgreSQL 数据库集群上设置 pglogical 扩展，首先要将 pglogical 添加到 Aurora PostgreSQL 数据库集群的自定义数据库集群参数组上的共享库中。您还需要将 rds.logical_replication 参数的值设置为 1，以开启逻辑解码。最后，在数据库中创建此扩展。您可以使用 AWS Management Console或 AWS CLI 执行这些任务。

您必须拥有 rds_superuser 角色的权限才能执行这些任务。

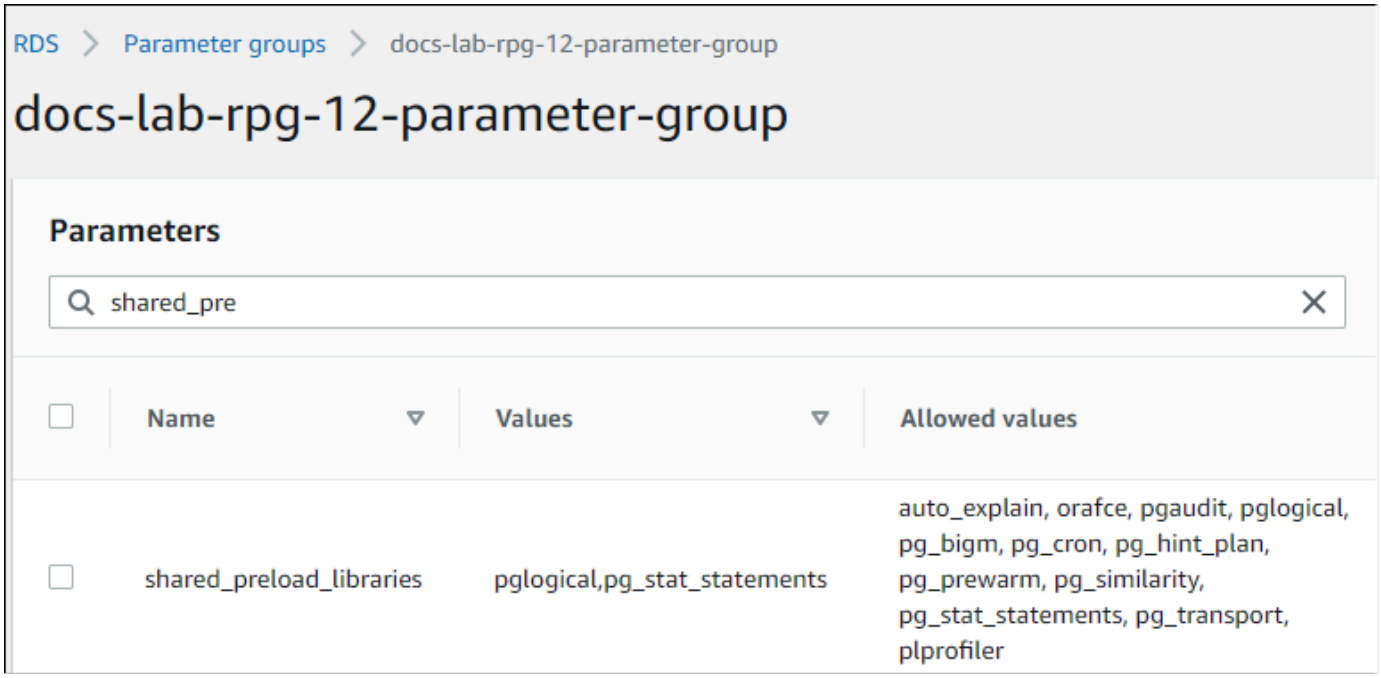
以下步骤假设您的 Aurora PostgreSQL 数据库集群 与自定义数据库集群 参数组相关联。有关创建自定义数据库集群参数组的信息，请参阅Amazon Aurora 的参数组。

控制台

设置 pglogical 扩展

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Aurora PostgreSQL 数据库集群的写入器实例 。
3. 打开 Aurora PostgreSQL 数据库集群写入器实例的配置选项卡。的 Configuration（配置）选项卡。在实例详细信息中，找到 Parameter group（参数组）链接。
4. 选择此链接以打开与您的 Aurora PostgreSQL 数据库集群 关联的自定义参数。

5. 在 Parameters（参数）搜索字段中，键入 `shared_pre` 以查找 `shared_preload_libraries` 参数。
6. 选择 Edit parameters（编辑参数）以访问属性值。
7. 将 `pglogical` 添加到 Values（值）字段的列表中。使用逗号分隔值列表中的项目。



8. 找到 `rds.logical_replication` 参数并将其设置为 `1`，以开启逻辑复制。
9. 重启 Aurora PostgreSQL 数据库集群的写入器实例，以使更改生效。
10. 当实例可用时，可以使用 `psql`（或 pgAdmin）连接到 Aurora PostgreSQL 数据库集群的写入器实例。

```
psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password --dbname=labdb
```

11. 要验证 pglogical 是否初始化，可以运行以下命令。

```
SHOW shared_preload_libraries;
shared_preload_libraries
--------------------------
rdsutils,pglogical
(1 row)
```

12. 验证启用逻辑解码的设置，如下所示。

```
SHOW wal_level;
```

```
wal_level
-----------
 logical
(1 row)
```

13. 创建扩展，如下所示。

```
CREATE EXTENSION pglogical;
EXTENSION CREATED
```

14. 选择 Save changes（保存更改）。
15. 通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
16. 从数据库列表中选择 Aurora PostgreSQL 数据库集群的写入器实例以将其选中，然后从 Actions（操作）菜单中选择 Reboot（重启）。

AWS CLI

设置 pglogical 扩展

要使用 AWS CLI 设置 pglogical，您可以调用 modify-db-parameter-group 操作来修改自定义参数组中的某些参数，如以下过程所示。

1. 使用以下 AWS CLI 命令向 `shared_preload_libraries` 参数中添加 `pglogical`。

```
aws rds modify-db-parameter-group \
   --db-parameter-group-name custom-param-group-name \
   --parameters
 "ParameterName=shared_preload_libraries,ParameterValue=pglogical,ApplyMethod=pending-
reboot" \
   --region aws-region
```

2. 使用以下 AWS CLI 命令将 `rds.logical_replication` 设置为 `1`，以针对 Aurora PostgreSQL 数据库集群的写入器实例 开启逻辑解码功能。

```
aws rds modify-db-parameter-group \
   --db-parameter-group-name custom-param-group-name \
   --parameters
 "ParameterName=rds.logical_replication,ParameterValue=1,ApplyMethod=pending-
reboot" \
   --region aws-region
```

3. 使用以下 AWS CLI 命令重启 Aurora PostgreSQL 数据库集群的写入器实例，以便初始化 pglogical 库。

```
aws rds reboot-db-instance \
    --db-instance-identifier writer-instance \
    --region aws-region
```

4. 当实例可用时，使用 psql 连接到 Aurora PostgreSQL 数据库集群的写入器实例。

```
psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password --dbname=labdb
```

5. 创建扩展，如下所示。

```
CREATE EXTENSION pglogical;
EXTENSION CREATED
```

6. 使用以下 AWS CLI 命令重启 Aurora PostgreSQL 数据库集群的写入器实例。

```
aws rds reboot-db-instance \
    --db-instance-identifier writer-instance \
    --region aws-region
```

为 Aurora PostgreSQL 数据库集群设置逻辑复制

以下过程说明如何在两个 Aurora PostgreSQL 数据库集群之间启动逻辑复制。这些步骤假设来源（发布者）和目标（订阅者）都如设置 pglogical 扩展中所述设置了 pglogical 扩展。

> **Note**
>
> 订阅用户节点的 node_name 不能以 rds 开头。

创建发布者节点并定义要复制的表

这些步骤假设您的 Aurora PostgreSQL 数据库集群有一个写入器实例以及一个数据库，其中包含一个或多个您要复制到另一个节点的表。您需要在订阅者上根据发布者重新创建表结构，因此，如果需要，首先获取表结构。为此，您可以使用 psql 元命令 \d *tablename*，然后在订阅者实例上创建相同的表。以下过程在发布者（来源）上创建示例表以用于演示目的。

1. 使用 psql 连接到具有要用作订阅者来源的表的实例。

```
psql --host=source-instance.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password --dbname=labdb
```

如果没有现有表要复制，可以按如下方式创建示例表。

a. 使用以下 SQL 语句创建一个示例表。

```
CREATE TABLE docs_lab_table (a int PRIMARY KEY);
```

b. 使用以下 SQL 语句用生成的数据填充表。

```
INSERT INTO docs_lab_table VALUES (generate_series(1,5000));
INSERT 0 5000
```

c. 使用以下 SQL 语句验证表中是否存在数据。

```
SELECT count(*) FROM docs_lab_table;
```

2. 将这一 Aurora PostgreSQL 数据库集群标识为发布者节点，如下所示。

```
SELECT pglogical.create_node(
    node_name := 'docs_lab_provider',
    dsn := 'host=source-instance.aws-region.rds.amazonaws.com port=5432
 dbname=labdb');
 create_node
-------------
   3410995529
(1 row)
```

3. 将要复制的表添加到默认的复制集。有关复制集的更多信息，请参阅 pglogical 文档中的复制集。

```
SELECT pglogical.replication_set_add_table('default', 'docs_lab_table', 'true',
 NULL, NULL);
 replication_set_add_table
  ---------------------------
  t
  (1 row)
```

发布者节点设置已完成。现在，您可以设置订阅者节点以接收来自发布者的更新。

设置订阅者节点并创建订阅以接收更新

这些步骤假设已使用 pglogical 扩展设置了 Aurora PostgreSQL 数据库集群。有关更多信息，请参阅 设置 pglogical 扩展。

1. 使用 psql 连接到要从发布者接收更新的实例。

```
psql --host=target-instance.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password --dbname=labdb
```

2. 在订阅者 Aurora PostgreSQL 数据库集群上，创建与发布者上存在的相同表。在本例中，该表为 docs_lab_table。您可以按如下所示创建表。

```
CREATE TABLE docs_lab_table (a int PRIMARY KEY);
```

3. 验证此表为空。

```
SELECT count(*) FROM docs_lab_table;
 count
-------
  0
(1 row)
```

4. 将这一 Aurora PostgreSQL 数据库集群标识为订阅者节点，如下所示。

```
SELECT pglogical.create_node(
    node_name := 'docs_lab_target',
    dsn := 'host=target-instance.aws-region.rds.amazonaws.com port=5432
 sslmode=require dbname=labdb user=postgres password=********');
 create_node
-------------
   2182738256
(1 row)
```

5. 创建订阅。

```
SELECT pglogical.create_subscription(
   subscription_name := 'docs_lab_subscription',
   provider_dsn := 'host=source-instance.aws-region.rds.amazonaws.com port=5432
 sslmode=require dbname=labdb user=postgres password=*******',
```

```
   replication_sets := ARRAY['default'],
   synchronize_data := true,
   forward_origins := '{}' );
 create_subscription
---------------------
1038357190
(1 row)
```

完成此步骤后，将在订阅者上的表中创建发布者上表中的数据。您可以使用以下 SQL 查询来验证是否已发生这种情况。

```
SELECT count(*) FROM docs_lab_table;
 count
-------
  5000
(1 row)
```

此后，对发布者上的表所做的更改将复制到订阅者上的表中。

在主要升级后重新建立逻辑复制

对于设置为逻辑复制的发布者节点的 Aurora PostgreSQL 数据库集群，在可以对其执行主要版本升级之前，您必须删除所有复制插槽，即使是不活动的复制插槽也是如此。我们建议您暂时从发布者节点转移数据库事务，删除复制插槽，升级 Aurora PostgreSQL 数据库集群，然后重新建立并重新启动复制。

复制插槽仅托管在发布者节点上。逻辑复制场景中的 Aurora PostgreSQL 订阅者节点没有要删除的插槽。Aurora PostgreSQL 主要版本升级过程支持将订阅者升级到独立于发布者节点的 PostgreSQL 的新主要版本。但是，升级过程确实会中断复制过程并干扰发布者节点和订阅者节点之间的 WAL 数据同步。在升级发布者、订阅者或两者之后，您需要在发布者和订阅者之间重新建立逻辑复制。以下过程说明了如何确定复制已中断以及如何解决此问题。

确定逻辑复制已中断

您可以通过查询发布者节点或订阅者节点来确定复制过程是否已中断，如下所示。

检查发布者节点

- 使用 psql 连接到发布者节点，然后查询 pg_replication_slots 函数。注意活动列中的值。通常，这将返回 t（true），表明复制处于活动状态。如果查询返回 f（false），则表明向订阅者的复制已停止。

```
SELECT slot_name,plugin,slot_type,active FROM pg_replication_slots;
                    slot_name              |      plugin      | slot_type | active
-------------------------------------------+------------------+-----------+--------
 pgl_labdb_docs_labcb4fa94_docs_lab3de412c | pglogical_output | logical   | f
(1 row)
```

检查订阅者节点

在订阅者节点上，您可以通过三种不同的方式检查复制的状态。

- 浏览订阅者节点上的 PostgreSQL 日志，以查找失败消息。该日志使用包含退出代码 1 的消息来标识故障，如下所示。

```
2022-07-06 16:17:03 UTC::@:[7361]:LOG: background worker "pglogical apply
 16404:2880255011" (PID 14610) exited with exit code 1
2022-07-06 16:19:44 UTC::@:[7361]:LOG: background worker "pglogical apply
 16404:2880255011" (PID 21783) exited with exit code 1
```

- 查询 pg_replication_origin 函数。使用 psql 连接到订阅者节点上的数据库并查询 pg_replication_origin 函数，如下所示。

```
SELECT * FROM pg_replication_origin;
 roident | roname
---------+--------
(0 rows)
```

结果集为空表示复制已中断。正常情况下，您将看到如下输出。

```
  roident |                       roname
 ---------+---------------------------------------------------
        1 | pgl_labdb_docs_labcb4fa94_docs_lab3de412c
 (1 row)
```

- 查询 pglogical.show_subscription_status 函数，如以下示例所示。

```
SELECT subscription_name,status,slot_name FROM pglogical.show_subscription_status();
     subscription_name | status |              slot_name
-------+----------------+--------+-----------------------------------
 docs_lab_subscription | down   | pgl_labdb_docs_labcb4fa94_docs_lab3de412c
(1 row)
```

此输出显示复制已中断。它的状态为 down。通常，输出将状态显示为 replicating。

如果您的逻辑复制过程已中断，则可以按照以下步骤重新建立复制。

在发布者节点和订阅者节点之间重新建立逻辑复制

要重新建立复制，请先断开订阅者与发布者节点的连接，然后重新建立订阅，如这些步骤所述。

1. 使用 psql 连接到订阅者节点，如下所示。

```
psql --host=222222222222.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password --dbname=labdb
```

2. 通过使用 pglogical.alter_subscription_disable 函数停用订阅。

```
SELECT pglogical.alter_subscription_disable('docs_lab_subscription',true);
 alter_subscription_disable
----------------------------
 t
(1 row)
```

3. 通过查询 pg_replication_origin 获取发布者节点的标识符，如下所示。

```
SELECT * FROM pg_replication_origin;
 roident |               roname
---------+-----------------------------------
       1 | pgl_labdb_docs_labcb4fa94_docs_lab3de412c
(1 row)
```

4. 将上一步的响应与 pg_replication_origin_create 命令一起使用，以分配重新建立时订阅可以使用的标识符。

```
SELECT pg_replication_origin_create('pgl_labdb_docs_labcb4fa94_docs_lab3de412c');
  pg_replication_origin_create
```

```
------------------------------
                             1
(1 row)
```

5. 通过传递其状态为 true 的名称来打开订阅，如下面的示例所示。

```
SELECT pglogical.alter_subscription_enable('docs_lab_subscription',true);
  alter_subscription_enable
---------------------------
 t
(1 row)
```

检查节点的状态。其状态应为 replicating，如本例所示。

```
SELECT subscription_name,status,slot_name
  FROM pglogical.show_subscription_status();
            subscription_name |   status    |              slot_name
------------------------------+-------------+-------------------------------------
 docs_lab_subscription         | replicating |
 pgl_labdb_docs_lab98f517b_docs_lab3de412c
(1 row)
```

检查发布者节点上订阅者的复制插槽的状态。插槽的 active 列应返回 t（true），表示已重新建立复制。

```
SELECT slot_name,plugin,slot_type,active
  FROM pg_replication_slots;
                   slot_name              |      plugin      | slot_type | active
------------------------------------------+------------------+-----------+--------
 pgl_labdb_docs_lab98f517b_docs_lab3de412c | pglogical_output | logical   | t
(1 row)
```

管理 Aurora PostgreSQL 的逻辑复制查槽

对于在逻辑复制场景中充当发布者节点的 Aurora PostgreSQL 数据库集群的编写器实例，在对其执行主要版本升级之前，必须删除该实例上的复制插槽。主要版本升级预检查过程会通知您，在插槽被删除之前，升级无法继续。

要识别使用 pglogical 扩展创建的复制插槽，请登录到每个数据库并获取节点的名称。当您查询订阅者节点时，您会在输出中得到发布者节点和订阅者节点，如本示例所示。

```
SELECT * FROM pglogical.node;
node_id   |     node_name
------------+------------------
 2182738256 | docs_lab_target
 3410995529 | docs_lab_provider
(2 rows)
```

您可以通过以下查询获取有关订阅的详细信息。

```
SELECT sub_name,sub_slot_name,sub_target
  FROM pglogical.subscription;
 sub_name |         sub_slot_name          | sub_target
----------+--------------------------------+------------
  docs_lab_subscription     | pgl_labdb_docs_labcb4fa94_docs_lab3de412c | 2182738256
(1 row)
```

现在您可以删除订阅，如下所示。

```
SELECT pglogical.drop_subscription(subscription_name := 'docs_lab_subscription');
 drop_subscription
-------------------
                 1
(1 row)
```

删除订阅后，您可以删除该节点。

```
SELECT pglogical.drop_node(node_name := 'docs-lab-subscriber');
 drop_node
-----------
 t
(1 row)
```

您可以验证该节点是否不再存在，如下所示。

```
SELECT * FROM pglogical.node;
 node_id | node_name
---------+-----------
(0 rows)
```

pglogical 扩展的参数参考

在表中，您可以找到与 pglogical 扩展关联的参数。pglogical.conflict_log_level 和 pglogical.conflict_resolution 等参数用于处理更新冲突。当对订阅来自发布者的更改的相同表进行本地更改时，可能会出现冲突。在不同情况下也可能发生冲突，例如双向复制或当多个订阅者从同一个发布者进行复制时。有关更多信息，请参阅使用 pglogical 进行 PostgreSQL 双向复制。

参数	描述
pglogical.batch_inserts	在可能时执行批量插入。默认情况下未设置。更改为“1”将打开，更改为“0”将关闭。
pglogical.conflict_log_level	设置用于记录已解决的冲突的日志级别。支持的字符串值为 debug5、debug4、debug3、debug2、debug1、info、notice、warning、error、log、fatal、panic。
pglogical.conflict_resolution	设置在冲突可以解决时用来解决冲突的方法。支持的字符串值为 error、apply_remote、keep_local、last_update_wins、first_update_wins。
pglogical.extra_connection_options	要添加到所有对等节点连接的连接选项。
pglogical.synchronous_commit	pglogical 专用的同步提交值
pglogical.use_spi	使用 SPI（服务器编程接口）而非低级 API 来应用更改。设置为“1”将打开，设置为“0”将关闭。有关 SPI 的更多信息，请参阅 PostgreSQL 文档中的服务器编程接口。

使用 Amazon Aurora PostgreSQL 支持的外部数据包装器

外部数据包装器 (FDW) 是一种特定类型的扩展，提供对外部数据的访问。例如，oracle_fdw 扩展允许您的 Aurora PostgreSQL 数据库实例 使用 Oracle 数据库。

在下文中，您可以了解有关几种支持的 PostgreSQL 外部数据包装器的信息。

主题

- 使用 log_fdw 扩展通过 SQL 访问数据库日志

- 使用 postgres_fdw 扩展访问外部数据
- 使用 mysql_fdw 扩展处理 MySQL 数据库
- 通过使用 oracle_fdw 扩展来使用 Oracle 数据库
- 通过使用 tds_fdw 扩展来使用 SQL Server 数据库

使用 log_fdw 扩展通过 SQL 访问数据库日志

Aurora PostgreSQL 数据库集群支持 log_fdw 扩展，您可以使用该扩展通过 SQL 界面访问数据库引擎日志。此 log_fdw 扩展提供了两个新函数，便于创建数据库日志的外部表：

- list_postgres_log_files – 列出数据库日志目录中的文件，以及文件大小 (以字节为单位)。
- create_foreign_table_for_log_file(table_name text, server_name text, log_file_name text) – 针对当前数据库中的指定文件构建外部表。

log_fdw 创建的所有函数均归 rds_superuser 所有。rds_superuser 角色的成员可以将这些函数的访问权限授予其他数据库用户。

默认情况下，日志文件由 Amazon Aurora 以 stderr（标准错误）格式生成，如 log_destination 参数中指定。此参数只有两个选项，即，stderr 和 csvlog（逗号分隔值，CSV）。如果您为参数添加 csvlog 选项，Amazon Aurora 会同时生成 stderr 和 csvlog 日志。这可能会影响数据库集群的存储容量，因此您需要了解影响日志处理的其它参数。有关更多信息，请参阅 设置日志目标（stderr、csvlog）。

生成 csvlog 日志的一个优势是 log_fdw 扩展允许您构建将数据整齐地拆分为多个列的外部表。为此，您的实例需要与自定义数据库参数组关联，以便您可以更改 log_destination 的设置。有关如何执行此操作的更多信息，请参阅Amazon Aurora 的参数组。

以下示例假设 log_destination 参数包含 cvslog。

使用 log_fdw 扩展

1. 安装 log_fdw 扩展。

```
postgres=> CREATE EXTENSION log_fdw;
CREATE EXTENSION
```

2. 创建日志服务器，作为外部数据包装器。

```
postgres=> CREATE SERVER log_server FOREIGN DATA WRAPPER log_fdw;
CREATE SERVER
```

3. 选择日志文件列表中的所有文件。

```
postgres=> SELECT * FROM list_postgres_log_files() ORDER BY 1;
```

示例响应如下所示。

```
          file_name           | file_size_bytes
------------------------------+-----------------
 postgresql.log.2023-08-09-22.csv |            1111
 postgresql.log.2023-08-09-23.csv |            1172
 postgresql.log.2023-08-10-00.csv |            1744
 postgresql.log.2023-08-10-01.csv |            1102
(4 rows)
```

4. 为所选文件创建包含单个“log_entry”列的表。

```
postgres=> SELECT create_foreign_table_for_log_file('my_postgres_error_log',
     'log_server', 'postgresql.log.2023-08-09-22.csv');
```

除了告知现在存在表格外，响应不提供详细信息。

```
-----------------------------------
(1 row)
```

5. 选择日志文件的示例。以下代码检索日志时间和错误消息描述。

```
postgres=> SELECT log_time, message FROM my_postgres_error_log ORDER BY 1;
```

示例响应如下所示。

```
             log_time             |                                 message
----------------------------------
+--------------------------------------------------------------------------------
Tue Aug 09 15:45:18.172 2023 PDT | ending log output to stderr
Tue Aug 09 15:45:18.175 2023 PDT | database system was interrupted; last known up
 at 2023-08-09 22:43:34 UTC
Tue Aug 09 15:45:18.223 2023 PDT | checkpoint record is at 0/90002E0
```

```
Tue Aug 09 15:45:18.223 2023 PDT | redo record is at 0/90002A8; shutdown FALSE
Tue Aug 09 15:45:18.223 2023 PDT | next transaction ID: 0/1879; next OID: 24578
Tue Aug 09 15:45:18.223 2023 PDT | next MultiXactId: 1; next MultiXactOffset: 0
Tue Aug 09 15:45:18.223 2023 PDT | oldest unfrozen transaction ID: 1822, in
 database 1
(7 rows)
```

使用 postgres_fdw 扩展访问外部数据

您可以使用 postgres_fdw 扩展访问远程数据库服务器上表中的数据。如果您从 PostgreSQL 数据库实例设置远程连接，则访问还可用于您的只读副本。

使用 postgres_fdw 访问远程数据库服务器

1. 安装 postgres_fdw 扩展。

```
CREATE EXTENSION postgres_fdw;
```

2. 使用 CREATE SERVER 创建外部数据服务器。

```
CREATE SERVER foreign_server
FOREIGN DATA WRAPPER postgres_fdw
OPTIONS (host 'xxx.xx.xxx.xx', port '5432', dbname 'foreign_db');
```

3. 创建用户映射，用于标识将在远程服务器上使用的角色。

> **Important**
>
> 要掩蔽密码以使其不出现在日志中，请在会话级别设置 `log_statement=none`。在参数级别进行设置不会掩蔽密码。

```
CREATE USER MAPPING FOR local_user
SERVER foreign_server
OPTIONS (user 'foreign_user', password 'password');
```

4. 创建一个表，该表映射到远程服务器上的表。

```
CREATE FOREIGN TABLE foreign_table (
        id integer NOT NULL,
```

```
        data text)
SERVER foreign_server
OPTIONS (schema_name 'some_schema', table_name 'some_table');
```

使用 mysql_fdw 扩展处理 MySQL 数据库

要从 Aurora PostgreSQL 数据库集群访问 MySQL 兼容数据库，您可以安装并使用 `mysql_fdw` 扩展。这款外部数据包装器允许您使用 RDS for MySQL、Aurora MySQL、MariaDB 和其他 MySQL 兼容数据库。从 Aurora PostgreSQL 数据库集群到 MySQL 数据库的连接会尽可能加密，具体取决于客户端和服务器配置。但是，如果您愿意，可以强制加密。有关更多信息，请参阅 将传输中加密与扩展配合使用。

`mysql_fdw` 扩展在 Amazon Aurora PostgreSQL 版本 15.4、14.9、13.12、12.16、 以及更高版本中受支持。它支持从 RDS for PostgreSQL 数据库到 MySQL 兼容数据库实例上的表的选择、插入、更新和删除。

主题

- 将 Aurora PostgreSQL 数据库设置为使用 mysql_fdw 扩展
- 示例：从 Aurora PostgreSQL 使用 Aurora MySQL 数据库
- 将传输中加密与扩展配合使用

将 Aurora PostgreSQL 数据库设置为使用 mysql_fdw 扩展

在您的 Aurora PostgreSQL 数据库集群上设置 `mysql_fdw` 扩展涉及在您的数据库集群中加载扩展，然后创建到 MySQL 数据库实例的连接点。对于该任务，您需要了解有关 MySQL 数据库实例的以下详细信息：

- 主机名或终端节点。对于Aurora MySQL 数据库集群，您可以使用控制台查找终端节点。选择 Connectivity & security（连接和安全）选项卡，然后查看 Endpoint and port（终端节点和端口）部分。
- 端口号。MySQL 的默认端口是 3306。
- 数据库的名称。数据库标识符。

您还需要为 MySQL 端口 3306 提供对安全组或访问控制列表 (ACL) 的访问权限。Aurora PostgreSQL 数据库集群和 Aurora MySQL 数据库集群均需要访问端口 3306。如果访问权限配置不正确，当尝试连接到 MySQL 兼容表时，您会看到一条与以下内容类似的错误消息：

```
ERROR: failed to connect to MySQL: Can't connect to MySQL server on 'hostname.aws-
region.rds.amazonaws.com:3306' (110)
```

在以下过程中，您（作为 rds_superuser 账户）创建外部服务器。然后，您将访问外部服务器的权限授予特定用户。然后，这些用户创建其自身到相应 MySQL 用户账户的映射以使用 MySQL 数据库实例。

使用 mysql_fdw 访问 MySQL 数据库服务器

1. 使用具有 rds_superuser 角色的账户连接到您的 PostgreSQL 数据库实例。如果在创建 Aurora PostgreSQL 数据库集群时接受默认值，则用户名为 postgres，您可以使用 psql 命令行工具进行连接，如下所示：

   ```
   psql --host=your-DB-instance.aws-region.rds.amazonaws.com --port=5432 --
   username=postgres --password
   ```

2. 按如下方式安装 mysql_fdw 扩展：

   ```
   postgres=> CREATE EXTENSION mysql_fdw;
   CREATE EXTENSION
   ```

在 Aurora PostgreSQL 数据库集群上安装扩展后，您可以设置提供与 MySQL 数据库连接的外部服务器。

创建外部服务器

在 Aurora PostgreSQL 数据库集群上执行这些任务。这些步骤假定您以具有 rds_superuser 特权的用户身份连接，例如 postgres。

1. 在 Aurora PostgreSQL 数据库集群中创建外部服务器：

   ```
   postgres=> CREATE SERVER mysql-db FOREIGN DATA WRAPPER mysql_fdw OPTIONS (host 'db-
   name.111122223333.aws-region.rds.amazonaws.com', port '3306');
   CREATE SERVER
   ```

2. 向适当的用户授予访问外部服务器的权限。这些用户应该是非管理员用户，即，没有 rds_superuser 角色的用户。

   ```
   postgres=> GRANT USAGE ON FOREIGN SERVER mysql-db to user1;
   ```

```
GRANT
```

PostgreSQL 用户通过外部服务器创建和管理其自身与 MySQL 数据库的连接。

示例：从 Aurora PostgreSQL 使用 Aurora MySQL 数据库

假设您在 Aurora PostgreSQL 数据库实例上有一个简单的表。您的 Aurora PostgreSQL 用户想要查询该表中的（SELECT）、INSERT、UPDATE 和 DELETE 项目。假设 mysql_fdw 扩展是在您的 RDS for PostgreSQL 数据库实例上创建的，如前面的过程中所述。以具有 rds_superuser 权限的用户身份连接到 RDS for PostgreSQL 数据库实例后，您可以继续执行以下步骤。

1. 在 Aurora PostgreSQL 数据库实例上，创建一个外部服务器：

```
test=> CREATE SERVER mysqldb FOREIGN DATA WRAPPER mysql_fdw OPTIONS (host 'your-
DB.aws-region.rds.amazonaws.com', port '3306');
CREATE SERVER
```

2. 将使用权授予没有 rds_superuser 权限的用户，例如 user1：

```
test=> GRANT USAGE ON FOREIGN SERVER mysqldb TO user1;
GRANT
```

3. 作为 *user1* 连接，然后创建一个到 MySQL 用户的映射：

```
test=> CREATE USER MAPPING FOR user1 SERVER mysqldb OPTIONS (username 'myuser',
 password 'mypassword');
CREATE USER MAPPING
```

4. 创建链接到 MySQL 表的外部表：

```
test=> CREATE FOREIGN TABLE mytab (a int, b text) SERVER mysqldb OPTIONS (dbname
 'test', table_name '');
CREATE FOREIGN TABLE
```

5. 针对外表运行简单查询：

```
test=> SELECT * FROM mytab;
a |   b
---+-------
1 | apple
(1 row)
```

6. 您可以从 MySQL 表中添加、更改和删除数据。例如：

```
test=> INSERT INTO mytab values (2, 'mango');
INSERT 0 1
```

再次运行 SELECT 查询以查看结果：

```
test=> SELECT * FROM mytab ORDER BY 1;
 a |   b
---+-------
1 | apple
2 | mango
(2 rows)
```

将传输中加密与扩展配合使用

默认情况下，从 Aurora PostgreSQL 到 MySQL 的连接使用传输中加密（TLS/SSL）。但是，当客户端和服务器配置不同时，连接会回退为非加密状态。您可以通过在 RDS for MySQL 用户账户上指定 REQUIRE SSL 选项来对所有传出连接强制加密。这种方法也适用于 MariaDB 和 Aurora MySQL 用户账户。

对于配置为 REQUIRE SSL 的 MySQL 用户帐户，如果无法建立安全连接，则连接尝试将失败。

要对现有 MySQL 数据库用户帐户强制加密，可以使用 ALTER USER 命令。根据 MySQL 版本的不同，语法有所不同，如下表所示。有关更多信息，请参阅《MySQL 参考手册》中的 ALTER USER。

MySQL 5.7、MySQL 8.0	MySQL 5.6
`ALTER USER 'user'@'%' REQUIRE SSL;`	`GRANT USAGE ON *.* to 'user'@'%' REQUIRE SSL;`

有关 mysql_fdw 扩展的更多信息，请参阅 mysql_fdw 文档。

通过使用 oracle_fdw 扩展来使用 Oracle 数据库

要从 Aurora PostgreSQL 数据库集群 访问 Oracle 数据库，您可以安装并使用 oracle_fdw 扩展。此扩展是 Oracle 数据库的外部数据包装器。要了解有关此扩展的更多信息，请参阅 oracle_fdw 文档。

Aurora PostgreSQL 12.7（Amazon Aurora PostgreSQL 版本 4.2）及更高版本支持 `oracle_fdw` 扩展。

主题

- 启用 oracle_fdw 扩展
- 示例：使用链接到 Amazon RDS for Oracle Database 的外部服务器
- 在传输过程中使用加密
- 了解 pg_user_mappings 视图和权限

启用 oracle_fdw 扩展

要使用 oracle_fdw 扩展，请执行以下步骤。

启用 oracle_fdw 扩展

- 使用具有 `rds_superuser` 权限的账户运行以下命令。

```
CREATE EXTENSION oracle_fdw;
```

示例：使用链接到 Amazon RDS for Oracle Database 的外部服务器

以下示例展示如何使用链接到 Amazon RDS for Oracle Database 的外部服务器。

创建链接到 RDS for Oracle 数据库的外部服务器

1. 请注意 RDS for Oracle 数据库实例上的以下内容：
   - Endpoint
   - 端口
   - 数据库名称
2. 创建外部服务器。

```
test=> CREATE SERVER oradb FOREIGN DATA WRAPPER oracle_fdw OPTIONS (dbserver
 '//endpoint:port/DB_name');
CREATE SERVER
```

3. 将使用权授予没有 `rds_superuser` 权限的用户，例如 `user1`。

```
test=> GRANT USAGE ON FOREIGN SERVER oradb TO user1;
GRANT
```

4. 作为 user1 连接并创建到 Oracle 用户的映射。

```
test=> CREATE USER MAPPING FOR user1 SERVER oradb OPTIONS (user 'oracleuser',
 password 'mypassword');
CREATE USER MAPPING
```

5. 创建链接到 Oracle 表的外部表。

```
test=> CREATE FOREIGN TABLE mytab (a int) SERVER oradb OPTIONS (table 'MYTABLE');
CREATE FOREIGN TABLE
```

6. 查询外部表。

```
test=>  SELECT * FROM mytab;
a
---
1
(1 row)
```

如果查询报告以下错误，请检查您的安全组和访问控制列表（ACL）以确保两个实例可以通信。

```
ERROR: connection for foreign table "mytab" cannot be established
DETAIL: ORA-12170: TNS:Connect timeout occurred
```

在传输过程中使用加密

PostgreSQL-to-Oracle 传输中加密基于客户端和服务器配置参数的组合。有关使用 Oracle 21c 的示例，请参阅 Oracle 文档中的关于协商加密和完整性的值。用于 Amazon RDS 上 oracle_fdw 的客户端配置有 ACCEPTED，这意味着加密取决于 Oracle 数据库服务器配置，并且使用 Oracle 安全库（libnnz）进行加密。

如果您的数据库位于 RDS for Oracle 上，请参阅 Oracle 本机网络加密来配置加密。

了解 pg_user_mappings 视图和权限

PostgreSQL 目录 pg_user_mapping 存储 Aurora PostgreSQL 用户到外部数据（远程）服务器上用户的映射。虽然对目录的访问受到限制，但是您可以使用 pg_user_mappings 视图来查看映射。在

下面，您可以找到一个示例，该示例演示如何将权限应用于示例 Oracle 数据库，但此信息更普遍地适用于任何外部数据包装器。

在以下输出中，您可以找到映射到三个不同示例用户的角色和权限。用户 rdssu1 和 rdssu2 是 rds_superuser 角色的成员，而 user1 不是。此示例使用 psql 元命令 \du 列出现有角色。

```
test=>  \du
                                                                List of roles
    Role name    |                          Attributes                          |
                  Member of
-----------------+-------------------------------------------------------------
+-----------------------------------------------------------
 rdssu1          |                                                              |
 {rds_superuser}
 rdssu2          |                                                              |
 {rds_superuser}
 user1           |                                                              | {}
```

所有用户，包括具有 rds_superuser 权限的用户，都可以查看 pg_user_mappings 表中他们自己的用户映射 (umoptions)。如以下示例所示，当 rdssu1 尝试获取所有用户映射时，即使存在 rdssu1rds_superuser 权限，也会出现错误：

```
test=> SELECT * FROM pg_user_mapping;
ERROR: permission denied for table pg_user_mapping
```

下面是一些示例。

```
test=> SET SESSION AUTHORIZATION rdssu1;
SET
test=> SELECT * FROM pg_user_mappings;
 umid  | srvid | srvname | umuser | usename    |            umoptions
-------+-------+---------+--------+------------+----------------------------------
 16414 | 16411 | oradb   |  16412 | user1      |
 16423 | 16411 | oradb   |  16421 | rdssu1     | {user=oracleuser,password=mypwd}
 16424 | 16411 | oradb   |  16422 | rdssu2     |
 (3 rows)

test=> SET SESSION AUTHORIZATION rdssu2;
SET
test=> SELECT * FROM pg_user_mappings;
 umid  | srvid | srvname | umuser | usename    |            umoptions
```

```
-------+-------+---------+--------+------------+-----------------------------------
 16414 | 16411 | oradb   |  16412 | user1      |
 16423 | 16411 | oradb   |  16421 | rdssu1     |
 16424 | 16411 | oradb   |  16422 | rdssu2     | {user=oracleuser,password=mypwd}
 (3 rows)

test=> SET SESSION AUTHORIZATION user1;
SET
test=> SELECT * FROM pg_user_mappings;
 umid  | srvid | srvname | umuser | usename    |           umoptions
-------+-------+---------+--------+------------+-------------------------------
 16414 | 16411 | oradb   |  16412 | user1      | {user=oracleuser,password=mypwd}
 16423 | 16411 | oradb   |  16421 | rdssu1     |
 16424 | 16411 | oradb   |  16422 | rdssu2     |
 (3 rows)
```

由于 information_schema._pg_user_mappings 和 pg_catalog.pg_user_mappings 的实施差异，手动创建的 rds_superuser 需要额外的权限才能在 pg_catalog.pg_user_mappings 中查看密码。

rds_superuser 无需额外权限即可在 information_schema._pg_user_mappings 中查看密码。

没有 rds_superuser 角色的用户只能在以下条件下在 pg_user_mappings 中查看密码：

- 当前用户是被映射的用户，拥有服务器或对其具有 USAGE 权限。
- 当前用户是服务器所有者，此映射用于 PUBLIC。

通过使用 tds_fdw 扩展来使用 SQL Server 数据库

您可以使用 PostgreSQL tds_fdw 扩展来访问支持表格数据流 (TDS) 协议的数据库，例如 Sybase 和 Microsoft SQL Server 数据库。此外部数据包装器可让您从 Aurora PostgreSQL 数据库集群连接到使用 TDS 协议的数据库，包括 Amazon RDS for Microsoft SQL Server。有关更多信息，请参阅 GitHub 上的 tds-fdw/tds_fdw 文档。

Amazon Aurora PostgreSQL 版本 13.6 及更高版本支持该 tds_fdw 扩展。

将 Aurora PostgreSQL 数据库设置为使用 tds_fdw 扩展

在以下过程中，您可以找到设置 tds_fdw 并将其与 Aurora PostgreSQL 数据库集群结合使用的示例。在可以使用 tds_fdw 连接到 SQL Server 数据库之前，您需要获取实例的以下详细信息：

- 主机名或终端节点。对于 RDS for SQL Server 数据库实例，您可以使用控制台查找终端节点。选择 Connectivity & security（连接和安全）选项卡，然后查看 Endpoint and port（终端节点和端口）部分。
- 端口号。Microsoft SQL Server 的默认端口是 1433。
- 数据库的名称。数据库标识符。

您还需要为 SQL Server 端口 1433 提供对安全组或访问控制列表 (ACL) 的访问权限。Aurora PostgreSQL 数据库集群和 RDS for SQL Server 数据库实例都需要访问端口 1433。如果访问权限配置不正确，当您尝试查询 Microsoft SQL Server 时，会看到以下错误消息：

```
ERROR: DB-Library error: DB #: 20009, DB Msg: Unable to connect:
Adaptive Server is unavailable or does not exist (mssql2019.aws-
region.rds.amazonaws.com), OS #: 0, OS Msg: Success, Level: 9
```

使用 tds_fdw 连接到 SQL Server 数据库

1. 使用具有 `rds_superuser` 角色的账户连接到您的 Aurora PostgreSQL 数据库集群的主要实例：

   ```
   psql --host=your-cluster-name-instance-1.aws-region.rds.amazonaws.com --port=5432
    --username=test --password
   ```

2. 安装 `tds_fdw` 扩展：

   ```
   test=> CREATE EXTENSION tds_fdw;
   CREATE EXTENSION
   ```

在 Aurora PostgreSQL 数据库集群上安装扩展后，应设置外部服务器。

创建外部服务器

使用具有 `rds_superuser` 权限的账户在 Aurora PostgreSQL 数据库集群上执行这些任务。

1. 在 Aurora PostgreSQL 数据库集群中创建外部服务器：

   ```
   test=> CREATE SERVER sqlserverdb FOREIGN DATA WRAPPER tds_fdw OPTIONS
    (servername 'mssql2019.aws-region.rds.amazonaws.com', port '1433', database
    'tds_fdw_testing');
   CREATE SERVER
   ```

要访问 SQLServer 端的非 ASCII 数据，请在 Aurora PostgreSQL 数据库集群中使用 character_set 选项创建服务器链接：

```
test=> CREATE SERVER sqlserverdb FOREIGN DATA WRAPPER tds_fdw OPTIONS (servername
 'mssql2019.aws-region.rds.amazonaws.com', port '1433', database 'tds_fdw_testing',
 character_set 'UTF-8');
CREATE SERVER
```

2. 向没有 rds_superuser 角色权限的用户授予权限，例如 user1：

```
test=> GRANT USAGE ON FOREIGN SERVER sqlserverdb TO user1;
```

3. 以 user1 身份进行连接，然后创建到 SQL Server 用户的映射：

```
test=> CREATE USER MAPPING FOR user1 SERVER sqlserverdb OPTIONS (username
 'sqlserveruser', password 'password');
CREATE USER MAPPING
```

4. 创建链接到 SQL Server 表的外部表：

```
test=> CREATE FOREIGN TABLE mytab (a int) SERVER sqlserverdb OPTIONS (table
 'MYTABLE');
CREATE FOREIGN TABLE
```

5. 查询外部表：

```
test=> SELECT * FROM mytab;
 a
---
 1
(1 row)
```

使用传输中的加密进行连接

Aurora PostgreSQL 到 SQL Server 的连接使用传输中加密 (TLS/SSL)，具体取决于 SQL Server 数据库配置。如果 SQL Server 未配置为加密，则向 SQL Server 数据库发出请求的 RDS for PostgreSQL 客户端将回退到未加密状态。

您可以通过设置 `rds.force_ssl` 参数对与 RDS for SQL Server 数据库实例的连接进行加密。要了解操作方法，请参阅强制与数据库实例的连接使用 SSL。有关 RDS for SQL Server 的 SSL/TLS 配置的更多信息，请参阅将 SSL 与 Microsoft SQL Server 数据库实例结合使用。

使用适用于 PostgreSQL 的可信语言扩展

适用于 PostgreSQL 的可信语言扩展是一个用于构建 PostgreSQL 扩展的开源开发套件。它允许您构建高性能 PostgreSQL 扩展，并在您的 Aurora PostgreSQL 数据库集群上安全地运行它们。通过使用适用于 PostgreSQL 的可信语言扩展（TLE），您可以创建 PostgreSQL 扩展，这些扩展遵循已记载的用于扩展 PostgreSQL 功能的方法。有关更多信息，请参阅 PostgreSQL 文档中的将相关对象打包为扩展。

TLE 的一个关键好处是，您可以在不提供对 PostgreSQL 实例底层文件系统的访问权限的环境中使用它。以前，安装新扩展需要访问文件系统。TLE 消除了这一约束。它提供了一个开发环境，用于为任何 PostgreSQL 数据库创建新扩展，包括在 Aurora PostgreSQL 数据库集群上运行的扩展。

TLE 旨在防止访问您使用 TLE 创建的扩展的不安全资源。它的运行时环境限制了任何扩展缺陷对单个数据库连接的影响。TLE 还让数据库管理员可以细粒度控制谁可以安装扩展，并为运行扩展提供了权限模型。

Aurora PostgreSQL 版本 14.5 及更高版本支持 TLE。

可信语言扩展开发环境和运行时打包为 `pg_tle` PostgreSQL 扩展，版本 1.0.1。它支持在 JavaScript、Perl、Tcl、PL/pgSQL 和 SQL 中创建扩展。您可以在 Aurora PostgreSQL 数据库集群中安装 `pg_tle` 扩展，方式与安装其他 PostgreSQL 扩展一样。设置 `pg_tle` 后，开发人员可以使用它来创建新的 PostgreSQL 扩展，称为 TLE 扩展。

在以下主题中，您可以了解有关如何设置可信语言扩展以及如何开始创建自己的 TLE 扩展的信息。

主题

- 术语
- 使用适用于 PostgreSQL 的可信语言扩展的要求
- 在 Aurora PostgreSQL 数据库集群中设置可信语言扩展
- 适用于 PostgreSQL 的可信语言扩展概述
- 为 Aurora PostgreSQL 创建 TLE 扩展
- 从数据库中删除 TLE 扩展
- 卸载适用于 PostgreSQL 的可信语言扩展
- 在您的 TLE 扩展中使用 PostgreSQL 挂钩
- 适用于 PostgreSQL 的可信语言扩展的函数参考
- 适用于 PostgreSQL 的可信语言扩展的挂钩参考

术语

为了帮助您更好地了解可信语言扩展，请查看以下词汇表，了解本主题中使用的术语。

适用于 PostgreSQL 的可信语言扩展

适用于 PostgreSQL 的可信语言扩展是打包为 `pg_tle` 扩展的开源开发套件的正式名称。它可以在任何 PostgreSQL 系统上使用。有关更多信息，请参阅 GitHub 上的 aws/pg_tle。

可信语言扩展

可信语言扩展是适用于 PostgreSQL 的可信语言扩展的简称。本文档中也使用了这个缩写名称及其缩写（TLE）。

可信语言

可信语言是一种具有特定安全属性的编程或脚本语言。例如，可信语言通常限制对文件系统的访问，并限制对指定网络属性的使用。TLE 开发套件旨在支持可信语言。PostgreSQL 支持几种不同的用于创建可信扩展或不可信扩展的语言。有关示例，请参阅 PostgreSQL 文档中的可信和不可信 PL/Perl。当您使用可信语言扩展创建扩展时，该扩展本质上使用可信语言机制。

TLE 扩展

TLE 扩展是使用可信语言扩展（TLE）开发套件创建的 PostgreSQL 扩展。

使用适用于 PostgreSQL 的可信语言扩展的要求

以下是设置和使用 TLE 开发套件的要求。

- Aurora PostgreSQL 版本 – 仅在 Aurora PostgreSQL 版本 14.5 及更高的版本上支持可信语言扩展。
  - 如果您需要升级 Aurora PostgreSQL 数据库集群，请参阅升级 Amazon Aurora PostgreSQL 数据库集群。
  - 如果您还没有运行 PostgreSQL 的 Aurora 数据库集群，则可以创建一个。有关更多信息，请参阅创建 Aurora PostgreSQL 数据库集群并连接到该集群。
- 需要 **`rds_superuser`** 权限 - 要设置和配置 `pg_tle` 扩展，您的数据库用户角色必须具有 `rds_superuser` 角色的权限。默认情况下，此角色被授予 `postgres` 用户，此用户创建 Aurora PostgreSQL 数据库集群 。
- 需要自定义数据库参数组 – 您的 Aurora PostgreSQL 数据库集群必须使用自定义数据库参数组进行配置。为 Aurora PostgreSQL 数据库集群的写入器实例使用自定义数据库参数组。

- 如果未使用自定义数据库参数组配置您的 Aurora PostgreSQL 数据库集群，则应创建一个参数组并将其与 Aurora PostgreSQL 数据库集群的写入器实例相关联。有关步骤的简短摘要，请参阅 创建和应用自定义数据库参数组。
- 如果已经使用自定义数据库参数组配置了 Aurora PostgreSQL 数据库集群，则可以设置可信语言扩展。有关详细信息，请参阅在 Aurora PostgreSQL 数据库集群中设置可信语言扩展。

创建和应用自定义数据库参数组

使用以下步骤创建自定义数据库参数组，并将 Aurora PostgreSQL 数据库集群配置为使用该参数组。

控制台

创建自定义数据库参数组并将其与您的 Aurora PostgreSQL 数据库集群一起使用

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 从 Amazon RDS 菜单中选择 Parameter groups（参数组）。
3. 选择创建参数组。
4. 在 Parameter group details（参数组详细信息）页面中，输入以下信息。
   - 对于Parameter group family（参数组系列），选择 aurora-postgresql14 。
   - 对于 Type（类型），请选择 DB Parameter Group（数据库参数组）。
   - 对于 Group name（组名称），在操作上下文中为参数组指定一个有意义的名称。
   - 对于 Description（描述），输入有用的描述，以便团队中的其他人可以轻松找到它。
5. 选择创建。您的自定义数据库参数组是在您的 AWS 区域中创建的。现在，您可以按照以下步骤修改 Aurora PostgreSQL 数据库集群以使用它。
6. 从 Amazon RDS 菜单中选择 Databases（数据库）。
7. 从列出的列表中选择要与 TLE 结合使用的 Aurora PostgreSQL 数据库集群，然后选择 Modify（修改）。
8. 在 Modify DB cluster settings（修改数据库集群设置）页面中，找到 Database options（数据库选项），然后使用选择器选择您的自定义数据库参数组。
9. 选择 Continue（继续）以保存更改。
10. 选择 Apply immediately（立即应用），这样您就可以继续将 Aurora PostgreSQL 数据库集群设置为使用 TLE。

要继续为系统设置可信语言扩展，请参阅 在 Aurora PostgreSQL 数据库集群中设置可信语言扩展。

有关使用数据库集群和数据库参数组的更多信息，请参阅 Amazon Aurora 数据库集群的数据库集群参数组。

AWS CLI

在使用 CLI 命令时，您可以通过将您的 AWS CLI 配置为使用默认 AWS 区域来避免指定 `--region` 参数。有关更多信息，请参阅《AWS Command Line Interface 用户指南》中的配置基础知识。

创建自定义数据库参数组并将其与您的 Aurora PostgreSQL 数据库集群一起使用

1. 使用 create-db-parameter-group AWS CLI 命令为您的 AWS 区域创建一个基于 aurora-postgresql14 的自定义数据库参数组。请注意，在此步骤中，您将创建数据库参数组，以应用于 Aurora PostgreSQL 数据库集群的写入器实例。

   对于 Linux、macOS 或 Unix：

   ```
   aws rds create-db-parameter-group \
     --region aws-region \
     --db-parameter-group-name custom-params-for-pg-tle \
     --db-parameter-group-family aurora-postgresql14 \
     --description "My custom DB parameter group for Trusted Language Extensions"
   ```

   对于 Windows：

   ```
   aws rds create-db-parameter-group ^
     --region aws-region ^
     --db-parameter-group-name custom-params-for-pg-tle ^
     --db-parameter-group-family aurora-postgresql14 ^
     --description "My custom DB parameter group for Trusted Language Extensions"
   ```

   您的自定义数据库参数组在 AWS 区域中可用，因此您可以修改 Aurora PostgreSQL 数据库集群的写入器实例以使用它。
2. 使用 modify-db-instance AWS CLI 命令将自定义数据库参数组应用于 Aurora PostgreSQL 数据库集群的写入器实例。此命令立即重启活动实例。

   对于 Linux、macOS 或 Unix：

   ```
   aws rds modify-db-instance \
     --region aws-region \
   ```

```
  --db-instance-identifier your-writer-instance-name \
  --db-parameter-group-name custom-params-for-pg-tle \
  --apply-immediately
```

对于 Windows：

```
aws rds modify-db-instance ^
  --region aws-region ^
  --db-instance-identifier your-writer-instance-name ^
  --db-parameter-group-name custom-params-for-pg-tle ^
  --apply-immediately
```

要继续为系统设置可信语言扩展，请参阅 在 Aurora PostgreSQL 数据库集群中设置可信语言扩展。

有关更多信息，请参阅 Amazon Aurora 数据库实例的数据库参数组 。

在 Aurora PostgreSQL 数据库集群中设置可信语言扩展

以下步骤假设您的 Aurora PostgreSQL 数据库集群与自定义数据库集群参数组相关联。您可以使用 AWS Management Console或 AWS CLI 来执行这些步骤。

当您在 Aurora PostgreSQL 数据库集群中设置可信语言扩展时，您可以将其安装在特定的数据库中，供对该数据库拥有权限的数据库用户使用。

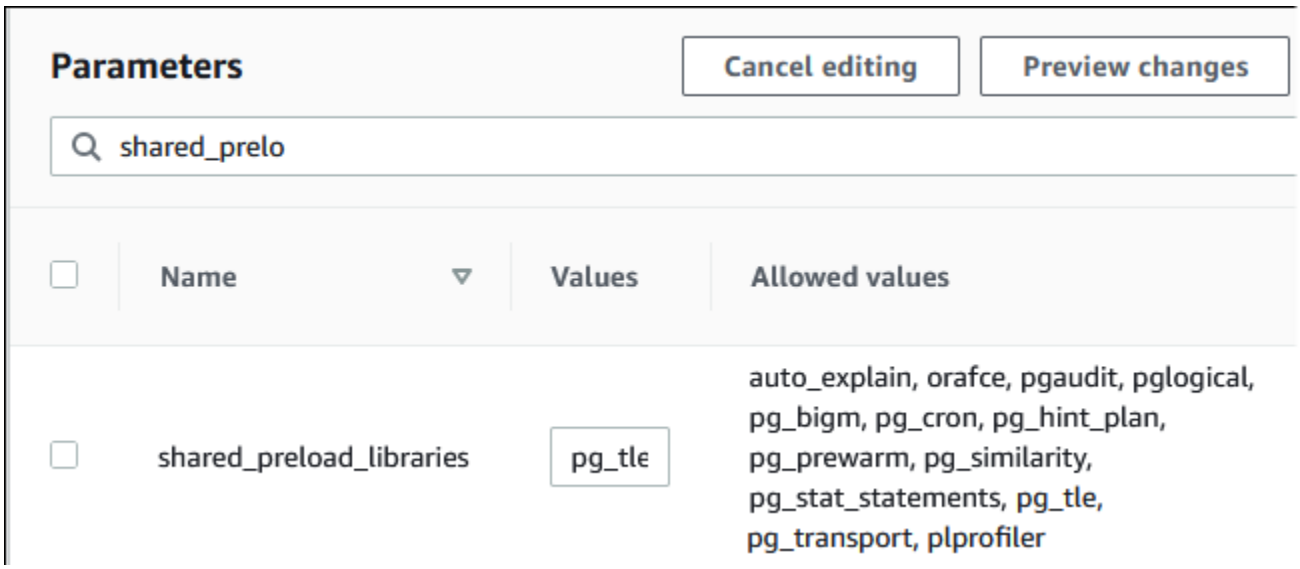
控制台

设置可信语言扩展

使用作为 `rds_superuser` 组（角色）成员的账户执行以下步骤。

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Aurora PostgreSQL 数据库集群的写入器实例 。
3. 打开 Aurora PostgreSQL 数据库集群写入器实例的配置选项卡。的 Configuration（配置）选项卡。在实例详细信息中，找到 Parameter group（参数组）链接。
4. 选择此链接以打开与您的 Aurora PostgreSQL 数据库集群 关联的自定义参数。
5. 在 Parameters（参数）搜索字段中，键入 `shared_pre` 以查找 `shared_preload_libraries` 参数。

6. 选择 Edit parameters（编辑参数）以访问属性值。
7. 将 pg_tle 添加到 Values（值）字段的列表中。使用逗号分隔值列表中的项目。



8. 重启 Aurora PostgreSQL 数据库集群的写入器实例，以使对 shared_preload_libraries 参数的更改生效。
9. 当实例可用时，验证 pg_tle 是否已初始化。使用 psql 连接到 Aurora PostgreSQL 数据库集群的写入器实例，然后运行以下命令。

```
SHOW shared_preload_libraries;
shared_preload_libraries
--------------------------
rdsutils,pg_tle
(1 row)
```

10. 初始化 pg_tle 扩展后，您现在可以创建此扩展了。

```
CREATE EXTENSION pg_tle;
```

可以使用以下 psql 元命令验证扩展是否已安装。

```
labdb=> \dx
                          List of installed extensions
  Name   | Version |   Schema   |                Description
---------+---------+------------+-------------------------------------------
 pg_tle  | 1.0.1   | pgtle      | Trusted-Language Extensions for PostgreSQL
 plpgsql | 1.0     | pg_catalog | PL/pgSQL procedural language
```

11. 在设置 Aurora PostgreSQL 数据库集群时，将 pgtle_admin 角色授予为其创建的主用户名。如果您接受了默认值，则它就是 postgres。

```
labdb=> GRANT pgtle_admin TO postgres;
GRANT ROLE
```

    您可以使用 psql 元命令来验证授权是否已完成，如以下示例中所示。输出中仅显示 pgtle_admin 和 postgres 角色。有关更多信息，请参阅了解 PostgreSQL 角色和权限。

```
labdb=> \du
                          List of roles
    Role name    |           Attributes            |               Member of
-----------------+---------------------------------
+-----------------------------------
pgtle_admin     | Cannot login                     | {}
postgres        | Create role, Create DB          +| {rds_superuser,pgtle_admin}
                | Password valid until infinity    |...
```

12. 使用 \q 元命令关闭 psql 会话。

```
\q
```

要开始创建 TLE 扩展，请参阅 示例：使用 SQL 创建可信语言扩展。

AWS CLI

在使用 CLI 命令时，您可以通过将您的 AWS CLI 配置为使用默认 AWS 区域来避免指定 --region 参数。有关更多信息，请参阅《AWS Command Line Interface 用户指南》中的配置基础知识。

设置可信语言扩展

1. 使用 modify-db-parameter-group AWS CLI 命令将 pg_tle 添加到 shared_preload_libraries 参数。

```
aws rds modify-db-parameter-group \
   --db-parameter-group-name custom-param-group-name \
   --parameters
 "ParameterName=shared_preload_libraries,ParameterValue=pg_tle,ApplyMethod=pending-
reboot" \
   --region aws-region
```

2. 使用 reboot-db-instance AWS CLI 命令重启 Aurora PostgreSQL 数据库集群的写入器实例并初始化 pg_tle 库。

```
aws rds reboot-db-instance \
    --db-instance-identifier writer-instance \
    --region aws-region
```

3. 当实例可用时，您可以验证 pg_tle 是否已初始化。使用 psql 连接到 Aurora PostgreSQL 数据库集群的写入器实例，然后运行以下命令。

```
SHOW shared_preload_libraries;
shared_preload_libraries
--------------------------
rdsutils,pg_tle
(1 row)
```

初始化 pg_tle 后，您现在可以创建扩展了。

```
CREATE EXTENSION pg_tle;
```

4. 在设置 Aurora PostgreSQL 数据库集群时，将 pgtle_admin 角色授予为其创建的主用户名。如果您接受了默认值，则它就是 postgres。

```
GRANT pgtle_admin TO postgres;
GRANT ROLE
```

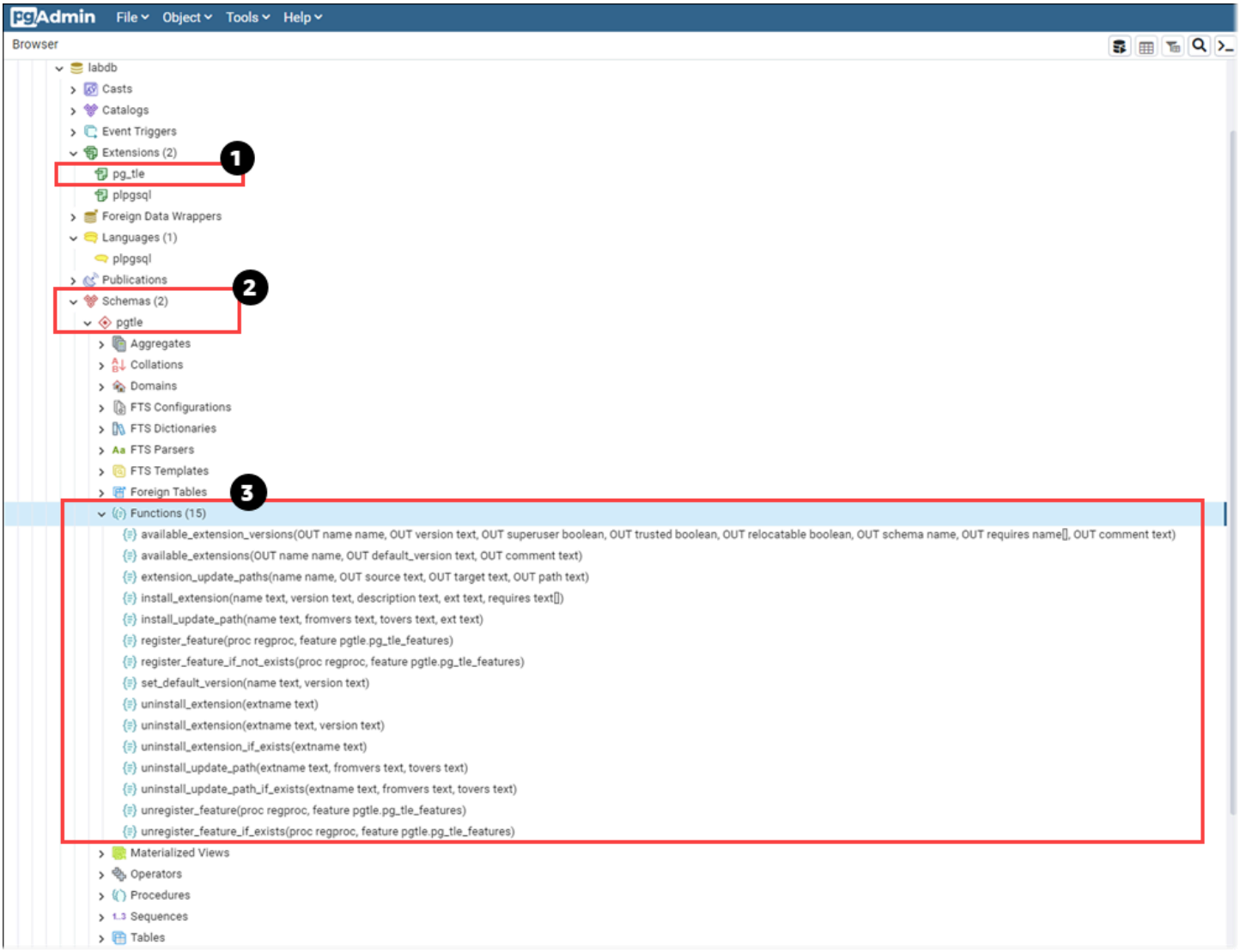
5. 按如下方式关闭 psql 会话。

```
labdb=> \q
```

要开始创建 TLE 扩展，请参阅 示例：使用 SQL 创建可信语言扩展。

适用于 PostgreSQL 的可信语言扩展概述

适用于 PostgreSQL 的可信语言扩展是一个 PostgreSQL 扩展，您可以像设置其他 PostgreSQL 扩展一样，将其安装在 Aurora PostgreSQL 数据库集群中。在下图的 pgAdmin 客户端工具的示例数据库中，您可以查看构成 pg_tle 扩展的部分组件。



您可以查看以下详细信息。

1. 适用于 PostgreSQL 的可信语言扩展（TLE）开发套件打包为 pg_tle 扩展。因此，pg_tle 添加到安装它的数据库的可用扩展中。
2. TLE 有它自己的模式，即 pgtle。此模式包含辅助函数（3 个），用于安装和管理您创建的扩展。
3. TLE 提供了十多个辅助函数，用于安装、注册和管理您的扩展。要了解有关这些函数的更多信息，请参阅 适用于 PostgreSQL 的可信语言扩展的函数参考。

pg_tle 扩展的其他组件包含以下各项：

- **pgtle_admin** 角色 – pgtle_admin 角色是在安装 pg_tle 扩展时创建的。此角色获得了相关权限，应视为具有此类权限。我们强烈建议您在向数据库用户授予 pgtle_admin 角色时遵循最低权

限原则。换句话说，仅向允许创建、安装和管理新 TLE 扩展（例如 `postgres`）的数据库用户授予 `pgtle_admin` 角色。

- **`pgtle.feature_info` 表** – `pgtle.feature_info` 表是一个受保护的表，其中包含有关您的 TLE、挂钩及其使用的自定义存储过程和函数的信息。如果您有 `pgtle_admin` 权限，则可以使用以下可信语言扩展函数在表中添加和更新该信息。
  - pgtle.register_feature
  - pgtle.register_feature_if_not_exists
  - pgtle.unregister_feature
  - pgtle.unregister_feature_if_exists

为 Aurora PostgreSQL 创建 TLE 扩展

您可以将使用 TLE 创建的任何扩展安装在安装了 `pg_tle` 扩展的任何 Aurora PostgreSQL 数据库集群中。`pg_tle` 扩展的范围限于安装该扩展的 PostgreSQL 数据库。使用 TLE 创建的扩展的范围限于同一个数据库。

使用各种 `pgtle` 函数安装构成 TLE 扩展的代码。以下可信语言扩展函数全部需要 `pgtle_admin` 角色。

- pgtle.install_extension
- pgtle.install_update_path
- pgtle.register_feature
- pgtle.register_feature_if_not_exists
- pgtle.set_default_version
- pgtle.uninstall_extension(name)
- pgtle.uninstall_extension(name, version)
- pgtle.uninstall_extension_if_exists
- pgtle.uninstall_update_path
- pgtle.uninstall_update_path_if_exists
- pgtle.unregister_feature
- pgtle.unregister_feature_if_exists

示例：使用 SQL 创建可信语言扩展

以下示例说明如何创建名为 pg_distance 的 TLE 扩展，该扩展包含一些 SQL 函数，用于使用不同的公式计算距离。在列表中，您可以找到用于计算曼哈顿距离的函数和用于计算欧几里得距离的函数。有关这些公式之间差异的更多信息，请参阅维基百科中的 Taxicab 几何和欧几里得几何。

如果您按照在 Aurora PostgreSQL 数据库集群中设置可信语言扩展中详述设置了 pg_tle 扩展，则可以在自己的 Aurora PostgreSQL 数据库集群中使用此示例。

> **Note**
>
> 您需要拥有 pgtle_admin 角色的权限才能执行此过程。

创建示例 TLE 扩展

以下步骤使用名为 labdb 的示例数据库。该数据库归 postgres 主用户所有。postgres 角色还具有 pgtle_admin 角色的权限。

1. 使用 psql 连接到 Aurora PostgreSQL 数据库集群的写入器实例 。

```
psql --host=db-instance-123456789012.aws-region.rds.amazonaws.com
--port=5432 --username=postgres --password --dbname=labdb
```

2. 通过复制以下代码并将其粘贴到 psql 会话控制台中来创建名为 pg_distance 的 TLE 扩展。

```
SELECT pgtle.install_extension
(
 'pg_distance',
 '0.1',
  'Distance functions for two points',
$_pg_tle_$
    CREATE FUNCTION dist(x1 float8, y1 float8, x2 float8, y2 float8, norm int)
    RETURNS float8
    AS $$
      SELECT (abs(x2 - x1) ^ norm + abs(y2 - y1) ^ norm) ^ (1::float8 / norm);
    $$ LANGUAGE SQL;

    CREATE FUNCTION manhattan_dist(x1 float8, y1 float8, x2 float8, y2 float8)
    RETURNS float8
    AS $$
```

```
      SELECT dist(x1, y1, x2, y2, 1);
    $$ LANGUAGE SQL;

    CREATE FUNCTION euclidean_dist(x1 float8, y1 float8, x2 float8, y2 float8)
    RETURNS float8
    AS $$
      SELECT dist(x1, y1, x2, y2, 2);
    $$ LANGUAGE SQL;
$_pg_tle_$
);
```

您将看到如下输出。

```
install_extension
---------------
 t
(1 row)
```

构成 pg_distance 扩展的构件现在已安装在您的数据库中。这些构件包括扩展的控制文件和代码，这些项目需要存在，这样才能使用 CREATE EXTENSION 命令创建扩展。换句话说，您仍然需要创建扩展以使其函数可供数据库用户使用。

3. 要创建扩展，请像使用任何其他扩展一样使用 CREATE EXTENSION 命令。与其他扩展一样，数据库用户需要在数据库中拥有 CREATE 权限。

```
CREATE EXTENSION pg_distance;
```

4. 要测试 pg_distance TLE 扩展，可以使用它来计算四个点之间的曼哈顿距离。

```
labdb=> SELECT manhattan_dist(1, 1, 5, 5);
8
```

要计算同一组点之间的欧几里得距离，可以使用以下方法。

```
labdb=> SELECT euclidean_dist(1, 1, 5, 5);
5.656854249492381
```

pg_distance 扩展将函数加载到数据库中，并使对数据库具有权限的任何用户都可以使用它们。

修改您的 TLE 扩展

要提高此 TLE 扩展中打包的函数的查询性能，请在其规范中添加以下两个 PostgreSQL 属性。

- IMMUTABLE – IMMUTABLE 属性可确保查询优化程序可以使用优化措施来缩短查询响应时间。有关更多信息，请参阅 PostgreSQL 文档中的函数波动性类别。
- PARALLEL SAFE – PARALLEL SAFE 属性是允许 PostgreSQL 在并行模式下运行该函数的另一个属性。有关更多信息，请参阅 PostgreSQL 文档中的 CREATE FUNCTION。

在以下示例中，您可以看到如何使用 pgtle.install_update_path 函数向每个函数添加这些属性，以创建 pg_distance TLE 扩展的版本 0.2。有关此函数的更多信息，请参阅 pgtle.install_update_path。您需要拥有 pgtle_admin 角色才能执行此任务。

更新现有 TLE 扩展并指定默认版本

1. 使用 psql 或其他客户端工具（例如 pgAdmin）连接到 Aurora PostgreSQL 数据库集群的写入器实例。

```
psql --host=db-instance-123456789012.aws-region.rds.amazonaws.com
--port=5432 --username=postgres --password --dbname=labdb
```

2. 通过复制以下代码并将其粘贴到 psql 会话控制台中来修改现有 TLE 扩展。

```
SELECT pgtle.install_update_path
(
 'pg_distance',
 '0.1',
 '0.2',
$_pg_tle_$
    CREATE OR REPLACE FUNCTION dist(x1 float8, y1 float8, x2 float8, y2 float8,
 norm int)
    RETURNS float8
    AS $$
      SELECT (abs(x2 - x1) ^ norm + abs(y2 - y1) ^ norm) ^ (1::float8 / norm);
    $$ LANGUAGE SQL IMMUTABLE PARALLEL SAFE;

    CREATE OR REPLACE FUNCTION manhattan_dist(x1 float8, y1 float8, x2 float8, y2
 float8)
    RETURNS float8
    AS $$
      SELECT dist(x1, y1, x2, y2, 1);
```

```
    $$ LANGUAGE SQL IMMUTABLE PARALLEL SAFE;

    CREATE OR REPLACE FUNCTION euclidean_dist(x1 float8, y1 float8, x2 float8, y2
 float8)
    RETURNS float8
    AS $$
      SELECT dist(x1, y1, x2, y2, 2);
    $$ LANGUAGE SQL IMMUTABLE PARALLEL SAFE;
$_pg_tle_$
);
```

您看到的响应与以下内容类似。

```
install_update_path
--------------------
 t
(1 row)
```

您可以将此版本的扩展设置为默认版本，这样，数据库用户在其数据库中创建或更新扩展时就不必指定版本。

3. 要将 TLE 扩展的修改版本（版本 0.2）指定为默认版本，请使用以下示例所示的 `pgtle.set_default_version` 函数。

```
SELECT pgtle.set_default_version('pg_distance', '0.2');
```

有关此函数的更多信息，请参阅 pgtle.set_default_version。

4. 代码准备就绪后，您可以使用 `ALTER EXTENSION ... UPDATE` 命令以常规方式更新已安装的 TLE 扩展，如此处所示：

```
ALTER EXTENSION pg_distance UPDATE;
```

从数据库中删除 TLE 扩展

您可以使用 `DROP EXTENSION` 命令删除 TLE 扩展，方法与处理其他 PostgreSQL 扩展的方法相同。删除扩展并不会删除构成扩展的安装文件，这允许用户重新创建扩展。要删除扩展及其安装文件，请执行以下两步过程。

删除 TLE 扩展并删除其安装文件

1. 使用 psql 或其他客户端工具连接到 Aurora PostgreSQL 数据库集群的写入器实例。

```
psql --host=cluster-instance-1.111122223333.aws-region.rds.amazonaws.com --
port=5432 --username=postgres --password --dbname=dbname
```

2. 像删除任何 PostgreSQL 扩展一样删除此扩展。

```
DROP EXTENSION your-TLE-extension
```

例如，如果您按照示例：使用 SQL 创建可信语言扩展中的详细说明创建 pg_distance 扩展，则可以按如下方式删除此扩展。

```
DROP EXTENSION pg_distance;
```

您将看到确认扩展已删除的输出，如下所示。

```
DROP EXTENSION
```

此时，该扩展在数据库中不再处于活动状态。但是，它的安装文件和控制文件在数据库中仍然可用，因此数据库用户可以根据需要再次创建扩展。

- 如果您想让扩展文件保持完好，以便数据库用户可以创建您的 TLE 扩展，则可以在此处停止。
- 如果要删除构成扩展的所有文件，请继续执行下一步。

3. 要删除扩展的所有安装文件，请使用 pgtle.uninstall_extension 函数。此函数删除扩展的所有代码和控制文件。

```
SELECT pgtle.uninstall_extension('your-tle-extension-name');
```

例如，要删除所有 pg_distance 安装文件，请使用以下命令。

```
SELECT pgtle.uninstall_extension('pg_distance');
 uninstall_extension
---------------------
 t
(1 row)
```

卸载适用于 PostgreSQL 的可信语言扩展

如果您不想再使用 TLE 创建自己的 TLE 扩展，则可以删除 pg_tle 扩展并删除所有构件。此操作包括删除数据库中的所有 TLE 扩展和删除 pgtle 模式。

从数据库中删除 **pg_tle** 扩展及其模式

1. 使用 psql 或其他客户端工具连接到 Aurora PostgreSQL 数据库集群的写入器实例。

   ```
   psql --host=cluster-instance-1.111122223333.aws-region.rds.amazonaws.com --
   port=5432 --username=postgres --password --dbname=dbname
   ```

2. 从数据库中删除 pg_tle 扩展。如果数据库中仍在运行您自己的 TLE 扩展，则还需要删除这些扩展。为此，您可以使用 CASCADE 关键字，如以下所示。

   ```
   DROP EXTENSION pg_tle CASCADE;
   ```

   如果 pg_tle 扩展在数据库中仍未处于活动状态，则无需使用 CASCADE 关键字。

3. 删除 pgtle 模式。此操作将从数据库中删除所有管理函数。

   ```
   DROP SCHEMA pgtle CASCADE;
   ```

   该过程完成后，该命令将返回以下内容。

   ```
   DROP SCHEMA
   ```

   pg_tle 扩展、其模式和函数以及所有构件均已删除。要使用 TLE 创建新扩展，请再次完成设置过程。有关更多信息，请参阅 在 Aurora PostgreSQL 数据库集群中设置可信语言扩展。

在您的 TLE 扩展中使用 PostgreSQL 挂钩

挂钩是 PostgreSQL 中可用的一种回调机制，它允许开发人员在常规数据库操作期间调用自定义函数或其他例程。TLE 开发套件支持 PostgreSQL 挂钩，因此您可以在运行时将自定义函数与 PostgreSQL 行为集成在一起。例如，您可以使用挂钩将身份验证过程与您自己的自定义代码关联起来，或者根据您的特定需求修改查询规划和执行流程。

您的 TLE 扩展可以使用挂钩。如果挂钩在作用域方面是全局的，则它适用于所有数据库。因此，如果您的 TLE 扩展使用全局挂钩，则需要在用户可以访问的所有数据库中创建 TLE 扩展。

当您使用 pg_tle 扩展构建自己的可信语言扩展时，您可以使用 SQL API 中的可用挂钩来构建扩展的功能。您应该向 pg_tle 注册任何挂钩。对于某些挂钩，您可能还需要设置各种配置参数。例如，可以将 passcode 检查挂钩设置为 on、off 或 require。有关可用 pg_tle 挂钩的特定要求的更多信息，请参阅 适用于 PostgreSQL 的可信语言扩展的挂钩参考。

示例：创建使用 PostgreSQL 挂钩的扩展

本节讨论的示例使用 PostgreSQL 挂钩检查在特定 SQL 操作期间提供的密码，并防止数据库用户将其密码设置为 password_check.bad_passwords 表中包含的任何密码。该表包含十大最常用但易于破解的密码选择。

要在 Aurora PostgreSQL 数据库集群中设置此示例，您必须已经安装了可信语言扩展。有关更多信息，请参阅 在 Aurora PostgreSQL 数据库集群中设置可信语言扩展。

设置密码检查挂钩示例

1. 使用 psql 连接到 Aurora PostgreSQL 数据库集群的写入器实例 。

```
psql --host=db-instance-123456789012.aws-region.rds.amazonaws.com
--port=5432 --username=postgres --password --dbname=labdb
```

2. 从密码检查挂钩代码列表中复制代码并将其粘贴到数据库中。

```
SELECT pgtle.install_extension (
  'my_password_check_rules',
  '1.0',
  'Do not let users use the 10 most commonly used passwords',
$_pgtle_$
  CREATE SCHEMA password_check;
  REVOKE ALL ON SCHEMA password_check FROM PUBLIC;
  GRANT USAGE ON SCHEMA password_check TO PUBLIC;

  CREATE TABLE password_check.bad_passwords (plaintext) AS
  VALUES
    ('123456'),
    ('password'),
    ('12345678'),
    ('qwerty'),
    ('123456789'),
    ('12345'),
    ('1234'),
    ('111111'),
```

```
    ('1234567'),
    ('dragon');
  CREATE UNIQUE INDEX ON password_check.bad_passwords (plaintext);

  CREATE FUNCTION password_check.passcheck_hook(username text, password text,
 password_type pgtle.password_types, valid_until timestamptz, valid_null boolean)
  RETURNS void AS $$
    DECLARE
      invalid bool := false;
    BEGIN
      IF password_type = 'PASSWORD_TYPE_MD5' THEN
        SELECT EXISTS(
          SELECT 1
          FROM password_check.bad_passwords bp
          WHERE ('md5' || md5(bp.plaintext || username)) = password
        ) INTO invalid;
        IF invalid THEN
          RAISE EXCEPTION 'Cannot use passwords from the common password
 dictionary';
        END IF;
      ELSIF password_type = 'PASSWORD_TYPE_PLAINTEXT' THEN
        SELECT EXISTS(
          SELECT 1
          FROM password_check.bad_passwords bp
          WHERE bp.plaintext = password
        ) INTO invalid;
        IF invalid THEN
          RAISE EXCEPTION 'Cannot use passwords from the common password
 dictionary';
        END IF;
      END IF;
    END
  $$ LANGUAGE plpgsql SECURITY DEFINER;

  GRANT EXECUTE ON FUNCTION password_check.passcheck_hook TO PUBLIC;

  SELECT pgtle.register_feature('password_check.passcheck_hook', 'passcheck');
$_pgtle_$
);
```

将扩展加载到数据库后，您会看到如下输出。

```
 install_extension
```

```
-------------------
 t
(1 row)
```

3. 当仍然连接到数据库时，现在可以创建扩展了。

```
CREATE EXTENSION my_password_check_rules;
```

4. 您可以使用以下 psql 元命令确认已在数据库中创建扩展。

```
\dx
                        List of installed extensions
          Name           | Version |   Schema   |
 Description
------------------------+---------+------------
+-------------------------------------------------------------
 my_password_check_rules | 1.0     | public     | Prevent use of any of the top-ten
 most common bad passwords
 pg_tle                  | 1.0.1   | pgtle      | Trusted-Language Extensions for
 PostgreSQL
 plpgsql                 | 1.0     | pg_catalog | PL/pgSQL procedural language
(3 rows)
```

5. 打开另一个终端会话来使用 AWS CLI。您需要修改您的自定义数据库参数组才能开启密码检查挂钩。为此，请使用 modify-db-parameter-group CLI 命令，如以下示例中所示。

```
aws rds modify-db-parameter-group \
    --region aws-region \
    --db-parameter-group-name your-custom-parameter-group \
    --parameters
 "ParameterName=pgtle.enable_password_check,ParameterValue=on,ApplyMethod=immediate"
```

对参数组设置进行的更改可能需要几分钟才能生效。但是，此参数是动态的，因此您无需重新启动 Aurora PostgreSQL 数据库集群的写入器实例，即可使该设置生效。

6. 打开 psql 会话并查询数据库，以验证密码检查挂钩已开启。

```
labdb=> SHOW pgtle.enable_password_check;
pgtle.enable_password_check
----------------------------
on
(1 row)
```

密码检查挂钩现处于活动状态。您可以通过创建新角色并使用其中一个错误密码来对其进行测试，如以下示例中所示。

```
CREATE ROLE test_role PASSWORD 'password';
ERROR:  Cannot use passwords from the common password dictionary
CONTEXT:  PL/pgSQL function
 password_check.passcheck_hook(text,text,pgtle.password_types,timestamp with time
 zone,boolean) line 21 at RAISE
SQL statement "SELECT password_check.passcheck_hook(
    $1::pg_catalog.text,
    $2::pg_catalog.text,
    $3::pgtle.password_types,
    $4::pg_catalog.timestamptz,
    $5::pg_catalog.bool)"
```

对输出设置了格式以便于阅读。

以下示例显示，pgsql 交互式元命令 \password 行为也受到密码检查挂钩的影响。

```
postgres=> SET password_encryption TO 'md5';
SET
postgres=> \password
Enter new password for user "postgres":*****
Enter it again:*****
ERROR:  Cannot use passwords from the common password dictionary
CONTEXT:  PL/pgSQL function
 password_check.passcheck_hook(text,text,pgtle.password_types,timestamp with time
 zone,boolean) line 12 at RAISE
SQL statement "SELECT password_check.passcheck_hook($1::pg_catalog.text,
 $2::pg_catalog.text, $3::pgtle.password_types, $4::pg_catalog.timestamptz,
 $5::pg_catalog.bool)"
```

如果需要，可以删除此 TLE 扩展并卸载其源文件。有关更多信息，请参阅 从数据库中删除 TLE 扩展。

密码检查挂钩代码列表

此处显示的示例代码定义了 my_password_check_rules TLE 扩展的规范。当您复制此代码并将其粘贴到数据库中时，my_password_check_rules 扩展的代码将加载到数据库中，并注册 password_check 挂钩以供扩展使用。

```
SELECT pgtle.install_extension (
```

```
  'my_password_check_rules',
  '1.0',
  'Do not let users use the 10 most commonly used passwords',
$_pgtle_$
  CREATE SCHEMA password_check;
  REVOKE ALL ON SCHEMA password_check FROM PUBLIC;
  GRANT USAGE ON SCHEMA password_check TO PUBLIC;

  CREATE TABLE password_check.bad_passwords (plaintext) AS
  VALUES
    ('123456'),
    ('password'),
    ('12345678'),
    ('qwerty'),
    ('123456789'),
    ('12345'),
    ('1234'),
    ('111111'),
    ('1234567'),
    ('dragon');
  CREATE UNIQUE INDEX ON password_check.bad_passwords (plaintext);

  CREATE FUNCTION password_check.passcheck_hook(username text, password text,
 password_type pgtle.password_types, valid_until timestamptz, valid_null boolean)
  RETURNS void AS $$
    DECLARE
      invalid bool := false;
    BEGIN
      IF password_type = 'PASSWORD_TYPE_MD5' THEN
        SELECT EXISTS(
          SELECT 1
          FROM password_check.bad_passwords bp
          WHERE ('md5' || md5(bp.plaintext || username)) = password
        ) INTO invalid;
        IF invalid THEN
          RAISE EXCEPTION 'Cannot use passwords from the common password dictionary';
        END IF;
      ELSIF password_type = 'PASSWORD_TYPE_PLAINTEXT' THEN
        SELECT EXISTS(
          SELECT 1
          FROM password_check.bad_passwords bp
          WHERE bp.plaintext = password
        ) INTO invalid;
        IF invalid THEN
```

```
          RAISE EXCEPTION 'Cannot use passwords from the common password dictionary';
        END IF;
      END IF;
    END
  $$ LANGUAGE plpgsql SECURITY DEFINER;

  GRANT EXECUTE ON FUNCTION password_check.passcheck_hook TO PUBLIC;

  SELECT pgtle.register_feature('password_check.passcheck_hook', 'passcheck');
$_pgtle_$
);
```

适用于 PostgreSQL 的可信语言扩展的函数参考

查看以下有关适用于 PostgreSQL 的可信语言扩展中提供的函数的参考文档。使用这些函数安装、注册、更新和管理您的 TLE 扩展，即您使用可信语言扩展开发套件开发的 PostgreSQL 扩展。

函数

- pgtle.available_extensions
- pgtle.available_extension_versions
- pgtle.extension_update_paths
- pgtle.install_extension
- pgtle.install_update_path
- pgtle.register_feature
- pgtle.register_feature_if_not_exists
- pgtle.set_default_version
- pgtle.uninstall_extension(name)
- pgtle.uninstall_extension(name, version)
- pgtle.uninstall_extension_if_exists
- pgtle.uninstall_update_path
- pgtle.uninstall_update_path_if_exists
- pgtle.unregister_feature
- pgtle.unregister_feature_if_exists

pgtle.available_extensions

pgtle.available_extensions 函数是一个集合返回函数。它返回数据库中所有可用的 TLE 扩展。返回的每一行都包含有关单个 TLE 扩展的信息。

函数原型

```
pgtle.available_extensions()
```

角色

无。

参数

无。

输出

- name – TLE 扩展的名称。
- default_version – 在未指定版本的情况下调用 CREATE EXTENSION 时要使用的 TLE 扩展的版本。
- description – 有关 TLE 扩展的更详细描述。

用法示例

```
SELECT * FROM pgtle.available_extensions();
```

pgtle.available_extension_versions

available_extension_versions 函数是一个集合返回函数。它返回所有可用 TLE 扩展及其版本的列表。每行都包含有关给定 TLE 扩展的特定版本的信息，包括它是否需要特定角色。

函数原型

```
pgtle.available_extension_versions()
```

角色

无。

参数

无。

输出

- name – TLE 扩展的名称。
- version – TLE 扩展的版本。
- superuser – 对于您的 TLE 扩展，此值始终为 false。创建 TLE 扩展或更新 TLE 扩展所需的权限与在给定数据库中创建其他对象所需的权限相同。
- trusted – 对于您的 TLE 扩展，此值始终为 false。
- relocatable – 对于您的 TLE 扩展，此值始终为 false。
- schema – 指定安装 TLE 扩展的模式的名称。
- requires – 包含此 TLE 扩展所需的其他扩展的名称的数组。
- description – TLE 扩展的详细描述。

有关输出值的更多信息，请参阅 PostgreSQL 文档中的将相关对象打包为扩展 > 扩展文件。

用法示例

```
SELECT * FROM pgtle.available_extension_versions();
```

pgtle.extension_update_paths

extension_update_paths 函数是一个集合返回函数。它返回 TLE 扩展的所有可能更新路径的列表。每行都包含该 TLE 扩展的可用升级或降级。

函数原型

```
pgtle.extension_update_paths(name)
```

角色

无。

参数

name – 从中获取升级路径的 TLE 扩展的名称。

输出

- source – 更新的源版本。
- target – 更新的目标版本。
- path – 用于将 TLE 扩展从 source 版本更新到 target 版本的升级路径，例如 0.1--0.2。

用法示例

```
SELECT * FROM pgtle.extension_update_paths('your-TLE');
```

pgtle.install_extension

install_extension 函数允许您在数据库中安装构成 TLE 扩展的构件，之后可以使用 CREATE EXTENSION 命令创建 TLE 扩展。

函数原型

```
pgtle.install_extension(name text, version text, description text, ext text, requires
 text[] DEFAULT NULL::text[])
```

角色

无。

参数

- name – TLE 扩展的名称。调用 CREATE EXTENSION 时使用此值。
- version – TLE 扩展的版本。
- description – 有关 TLE 扩展的详细描述。此描述显示在 pgtle.available_extensions() 的 comment 字段中。
- ext – TLE 扩展的内容。此值包含诸如函数之类的对象。
- requires – 一个可选参数，用于指定此 TLE 扩展的依赖项。pg_tle 扩展会自动添加为依赖项。

其中许多参数与扩展控制文件中包含的参数相同，用于在 PostgreSQL 实例的文件系统上安装 PostgreSQL 扩展。有关更多信息，请参阅 PostgreSQL 文档的将相关对象打包为扩展中的扩展文件。

输出

此函数在成功时返回 OK，在出现错误时返回 NULL。

- OK – TLE 扩展已成功安装在数据库中。
- NULL – TLE 扩展未成功安装在数据库中。

用法示例

```
SELECT pgtle.install_extension(
 'pg_tle_test',
 '0.1',
 'My first pg_tle extension',
$_pgtle_$
  CREATE FUNCTION my_test()
  RETURNS INT
  AS $$
    SELECT 42;
  $$ LANGUAGE SQL IMMUTABLE;
$_pgtle_$
);
```

pgtle.install_update_path

`install_update_path` 函数提供 TLE 扩展的两个不同版本之间的更新路径。此功能允许您的 TLE 扩展的用户使用 `ALTER EXTENSION ... UPDATE` 语法更新其版本。

函数原型

```
pgtle.install_update_path(name text, fromvers text, tovers text, ext text)
```

角色

`pgtle_admin`

参数

- `name` – TLE 扩展的名称。调用 `CREATE EXTENSION` 时使用此值。
- `fromvers` – 用于升级的 TLE 扩展的源版本。
- `tovers` – 用于升级的 TLE 扩展的目标版本。
- `ext` – 更新的内容。此值包含诸如函数之类的对象。

输出

无。

用法示例

```
SELECT pgtle.install_update_path('pg_tle_test', '0.1', '0.2',
  $_pgtle_$
    CREATE OR REPLACE FUNCTION my_test()
    RETURNS INT
    AS $$
      SELECT 21;
    $$ LANGUAGE SQL IMMUTABLE;
  $_pgtle_$
);
```

pgtle.register_feature

`register_feature` 函数向 `pgtle.feature_info` 表中添加了指定的内部 PostgreSQL 功能。PostgreSQL 挂钩是 PostgreSQL 内部功能的一个示例。可信语言扩展开发套件支持使用 PostgreSQL 挂钩。目前，此函数支持以下功能。

- `passcheck` – 将密码检查挂钩注册到自定义 PostgreSQL 的密码检查行为的过程或函数。

函数原型

```
pgtle.register_feature(proc regproc, feature pg_tle_feature)
```

角色

`pgtle_admin`

参数

- `proc` – 用于该功能的存储过程或函数的名称。
- `feature` – 要注册到该函数的 `pg_tle` 功能（例如 `passcheck`）的名称。

输出

无。

用法示例

```
SELECT pgtle.register_feature('pw_hook', 'passcheck');
```

pgtle.register_feature_if_not_exists

`pgtle.register_feature_if_not_exists` 函数将指定的 PostgreSQL 功能添加到 `pgtle.feature_info` 表中，并标识使用该功能的 TLE 扩展或其他过程或函数。有关挂钩和可信语言扩展的更多信息，请参阅 在您的 TLE 扩展中使用 PostgreSQL 挂钩。

函数原型

```
pgtle.register_feature_if_not_exists(proc regproc, feature pg_tle_feature)
```

角色

`pgtle_admin`

参数

- `proc` – 包含用作 TLE 扩展的功能的逻辑（代码）的存储过程或函数的名称。例如，`pw_hook` 代码。
- `feature` – 要为 TLE 函数注册的 PostgreSQL 功能的名称。目前，唯一可用的功能是 `passcheck` 挂钩。有关更多信息，请参阅 密码检查挂钩（passcheck）。

输出

为指定的扩展注册该功能后返回 `true`。如果该功能已经注册，则返回 `false`。

用法示例

```
SELECT pgtle.register_feature_if_not_exists('pw_hook', 'passcheck');
```

pgtle.set_default_version

`set_default_version` 函数可让您指定 TLE 扩展的 `default_version`。您可以使用此函数定义升级路径，并将该版本指定为 TLE 扩展的缺省版本。当数据库用户在 `CREATE EXTENSION` 和 `ALTER`

EXTENSION ... UPDATE 命令中指定您的 TLE 扩展时，将在数据库中为该用户创建该版本的 TLE 扩展。

成功时，此函数返回 true。如果在 name 参数中指定的 TLE 扩展不存在，此函数将返回错误。同样，如果 TLE 扩展的 version 不存在，它会返回错误。

函数原型

```
pgtle.set_default_version(name text, version text)
```

角色

pgtle_admin

参数

- name – TLE 扩展的名称。调用 CREATE EXTENSION 时使用此值。
- version – 用于设置缺省设置的 TLE 扩展版本。

输出

- true – 成功设置缺省版本时，此函数返回 true。
- ERROR – 如果具有指定名称或版本的 TLE 扩展不存在，则返回错误消息。

用法示例

```
SELECT * FROM pgtle.set_default_version('my-extension', '1.1');
```

pgtle.uninstall_extension(name)

uninstall_extension 函数从数据库中删除 TLE 扩展的所有版本。此函数可防止 CREATE EXTENSION 的将来调用安装 TLE 扩展。如果数据库中不存在 TLE 扩展，则会引发错误。

uninstall_extension 函数不会删除数据库中当前处于活动状态的 TLE 扩展。要删除当前处于活动状态的 TLE 扩展，您需要显式调用 DROP EXTENSION 以将其删除。

函数原型

```
pgtle.uninstall_extension(extname text)
```

角色

pgtle_admin

参数

- extname – 要卸载的 TLE 扩展的名称。此名称与用于 CREATE EXTENSION 的名称相同，旨在加载 TLE 扩展以便在给定数据库中使用。

输出

无。

用法示例

```
SELECT * FROM pgtle.uninstall_extension('pg_tle_test');
```

pgtle.uninstall_extension(name, version)

uninstall_extension(name, version) 函数从数据库中删除指定版本的 TLE 扩展。此功能可防止 CREATE EXTENSION 和 ALTER EXTENSION 将 TLE 扩展安装或更新到指定版本。此函数还删除指定版本的 TLE 扩展的所有更新路径。如果 TLE 扩展当前在数据库中处于活动状态，则此函数不会将其卸载。必须显式调用 DROP EXTENSION 才能删除 TLE 扩展。要卸载 TLE 扩展的所有版本，请参阅 pgtle.uninstall_extension(name)。

函数原型

```
pgtle.uninstall_extension(extname text, version text)
```

角色

pgtle_admin

参数

- extname – TLE 扩展的名称。调用 CREATE EXTENSION 时使用此值。
- version – 要从数据库中卸载的 TLE 扩展的版本。

输出

无。

用法示例

```
SELECT * FROM pgtle.uninstall_extension('pg_tle_test', '0.2');
```

pgtle.uninstall_extension_if_exists

uninstall_extension_if_exists 函数从给定的数据库中删除 TLE 扩展的所有版本。如果 TLE 扩展不存在，则该函数会静默返回（不会引发任何错误消息）。如果指定的扩展当前在数据库中处于活动状态，则此函数不会将其删除。必须先显式调用 DROP EXTENSION 以删除 TLE 扩展，然后才能使用此函数卸载其构件。

函数原型

```
pgtle.uninstall_extension_if_exists(extname text)
```

角色

pgtle_admin

参数

- extname – TLE 扩展的名称。调用 CREATE EXTENSION 时使用此值。

输出

在卸载指定扩展后，uninstall_extension_if_exists 函数返回 true。如果指定的扩展不存在，则此函数将返回 false。

- true – 卸载 TLE 扩展后返回 true。
- false – 当数据库中不存在 TLE 扩展时返回 false。

用法示例

```
SELECT * FROM pgtle.uninstall_extension_if_exists('pg_tle_test');
```

pgtle.uninstall_update_path

uninstall_update_path 函数从 TLE 扩展中删除特定的更新路径。这可以防止 ALTER EXTENSION ... UPDATE TO 将其用作更新路径。

如果 TLE 扩展当前正由此更新路径上的其中一个版本使用，则该扩展保留在数据库中。

如果指定的更新路径不存在，则此函数会引发错误。

函数原型

```
pgtle.uninstall_update_path(extname text, fromvers text, tovers text)
```

角色

pgtle_admin

参数

- extname – TLE 扩展的名称。调用 CREATE EXTENSION 时使用此值。
- fromvers – 更新路径上使用的 TLE 扩展的源版本。
- tovers – 更新路径上使用的 TLE 扩展的目标版本。

输出

无。

用法示例

```
SELECT * FROM pgtle.uninstall_update_path('pg_tle_test', '0.1', '0.2');
```

pgtle.uninstall_update_path_if_exists

uninstall_update_path_if_exists 函数与 uninstall_update_path 类似，它从 TLE 扩展中删除指定的更新路径。但是，如果更新路径不存在，则此函数不会引发错误消息。相反，该函数返回 false。

函数原型

```
pgtle.uninstall_update_path_if_exists(extname text, fromvers text, tovers text)
```

角色

pgtle_admin

参数

- extname – TLE 扩展的名称。调用 CREATE EXTENSION 时使用此值。
- fromvers – 更新路径上使用的 TLE 扩展的源版本。
- tovers – 更新路径上使用的 TLE 扩展的目标版本。

输出

- true – 该函数已成功更新 TLE 扩展的路径。
- false – 该函数无法更新 TLE 扩展的路径。

用法示例

```
SELECT * FROM pgtle.uninstall_update_path_if_exists('pg_tle_test', '0.1', '0.2');
```

pgtle.unregister_feature

unregister_feature 函数提供了一种方法，用于删除注册为使用 pg_tle 功能（如挂钩）的函数。有关注册功能的信息，请参阅 pgtle.register_feature。

函数原型

```
pgtle.unregister_feature(proc regproc, feature pg_tle_features)
```

角色

pgtle_admin

参数

- proc – 要向 pg_tle 功能注册的存储函数的名称。
- feature – 要向函数注册的 pg_tle 功能的名称。例如，passcheck 是一项可以注册以供您开发的可信语言扩展使用的功能。有关更多信息，请参阅 密码检查挂钩（passcheck）。

输出

无。

用法示例

```
SELECT * FROM pgtle.unregister_feature('pw_hook', 'passcheck');
```

pgtle.unregister_feature_if_exists

unregister_feature 函数提供了一种方法，用于删除注册为使用 pg_tle 功能（如挂钩）的函数。有关更多信息，请参阅 在您的 TLE 扩展中使用 PostgreSQL 挂钩。成功取消注册该功能后，返回 true。如果未注册该功能，则返回 false。

有关为 TLE 扩展注册 pg_tle 功能的信息，请参阅 pgtle.register_feature。

函数原型

```
pgtle.unregister_feature_if_exists('proc regproc', 'feature pg_tle_features')
```

角色

pgtle_admin

参数

- proc – 为包含 pg_tle 功能而注册的存储函数的名称。
- feature – 向可信语言扩展注册的 pg_tle 功能的名称。

输出

返回 true 或 false，如下所示。

- true – 该函数已成功将该功能从扩展中取消注册。
- false – 该函数无法从 TLE 扩展中取消注册该功能。

用法示例

```
SELECT * FROM pgtle.unregister_feature_if_exists('pw_hook', 'passcheck');
```

适用于 PostgreSQL 的可信语言扩展的挂钩参考

适用于 PostgreSQL 的可信语言扩展支持 PostgreSQL 挂钩。挂钩是一种内部回调机制，可供开发人员扩展 PostgreSQL 的核心功能。通过使用挂钩，开发人员可以实现自己的函数或过程以在各种数据

库操作中使用，从而以某种方式修改 PostgreSQL 的行为。例如，您可以使用 passcheck 挂钩自定义 PostgreSQL 如何处理在为用户（角色）创建或更改密码时提供的密码。

查看以下文档，了解可用于 TLE 扩展的 passcheck 钩子。要了解有关可用钩子（包括客户端身份验证钩子）的更多信息，请参阅 Trusted Language Extensions hooks。

密码检查挂钩（passcheck）

passcheck 挂钩用于在以下 SQL 命令和 psql 元命令的密码检查过程中自定义 PostgreSQL 行为。

- CREATE ROLE *username* ...PASSWORD – 有关更多信息，请参阅 PostgreSQL 文档中的 CREATE ROLE。
- ALTER ROLE *username*...PASSWORD – 有关更多信息，请参阅 PostgreSQL 文档中的 ALTER ROLE。
- \password *username* – 此交互式 psql 元命令在透明地使用 ALTER ROLE ... PASSWORD 语法之前，通过对密码进行哈希处理来安全地更改指定用户的密码。元命令是 ALTER ROLE ... PASSWORD 命令的安全包装器，因此挂钩适用于 psql 元命令的行为。

有关示例，请参阅密码检查挂钩代码列表。

目录

- 函数原型
- 参数
- 配置
- 使用说明

函数原型

```
passcheck_hook(username text, password text, password_type pgtle.password_types,
 valid_until timestamptz, valid_null boolean)
```

参数

passcheck 挂钩函数采用以下参数。

- username – 设置密码的角色（用户名）的名称（文本）。

- password – 纯文本或哈希处理的密码。输入的密码应与在 password_type 中指定的类型相匹配。
- password_type – 指定密码的 pgtle.password_type 格式。此格式可能是以下选项之一。
  - PASSWORD_TYPE_PLAINTEXT – 纯文本密码。
  - PASSWORD_TYPE_MD5 – 已使用 MD5（消息摘要 5）算法进行哈希处理的密码。
  - PASSWORD_TYPE_SCRAM_SHA_256 – 已使用 SCRAM-SHA-256 算法进行哈希处理的密码。
- valid_until – 指定密码变为失效的时间。此参数是可选的。如果使用此参数，请将时间指定为 timestamptz 值。
- valid_null – 如果此布尔值设置为 true，则 valid_until 选项设置为 NULL。

配置

函数 pgtle.enable_password_check 控制 passcheck 挂钩是否处于活动状态。passcheck 挂钩有三种可能的设置。

- off – 关闭 passcheck 密码检查挂钩。这是默认值。
- on – 打开 passcode 密码检查挂钩，以便对照表检查密码。
- require – 需要定义密码检查挂钩。

使用说明

要打开或关闭 passcheck 挂钩，您需要修改 Aurora PostgreSQL 数据库集群的写入器实例的自定义数据库参数组。

对于 Linux、macOS 或 Unix：

```
aws rds modify-db-parameter-group \
    --region aws-region \
    --db-parameter-group-name your-custom-parameter-group \
    --parameters
 "ParameterName=pgtle.enable_password_check,ParameterValue=on,ApplyMethod=immediate"
```

对于 Windows：

```
aws rds modify-db-parameter-group ^
    --region aws-region ^
    --db-parameter-group-name your-custom-parameter-group ^
```

```
    --parameters
 "ParameterName=pgtle.enable_password_check,ParameterValue=on,ApplyMethod=immediate"
```

Amazon Aurora PostgreSQL 参考

在以下主题中，可以找到有关 Amazon Aurora PostgreSQL 中的排序规则、函数、参数和等待事件的信息。

主题

- 用于 EBCDIC 和其他大型机迁移的 Aurora PostgreSQL 排序规则
- Aurora PostgreSQL 中支持的排序规则
- Aurora PostgreSQL 函数参考
- Amazon Aurora PostgreSQL 参数
- Amazon Aurora PostgreSQL 等待事件

用于 EBCDIC 和其他大型机迁移的 Aurora PostgreSQL 排序规则

将大型机应用程序迁移到新平台（如 AWS）可以完美地保留应用程序行为。要使应用程序在新平台上的行为与在大型机上的行为完全相同，需要使用相同的排序规则对迁移的数据进行整理。例如，许多 Db2 迁移解决方案将空值转移到 u0180（Unicode 位置 0180），因此这些排序规则首先对 u0180 进行排序。这是一个示例，说明排序规则与其大型机来源有何不同，以及为什么需要选择能够更好地映射到原始 EBCDIC 排序规则的排序规则。

Aurora PostgreSQL 14.3 及更高版本提供了许多 ICU 和 EBCDIC 排序规则，以支持使用 AWS Mainframe Modernization 服务向 AWS 执行此类迁移。要了解有关此服务的更多信息，请参阅什么是 AWS Mainframe Modernization？

在下表中，您可以找到 Aurora PostgreSQL 提供的排序规则。这些排序规则遵循 EBCDIC 规则，并确保大型机应用程序在 AWS 上的运行方式与在大型机环境中相同。排序规则名称包括相关的代码页（cpnnnn），以便您可以为大型机来源选择适当的排序规则。例如，使用 `en-US-cp037-x-icu` 针对 EBCDIC 数据（源自使用代码页 037 的大型机应用程序的）实现排序规则行为。

EBCDIC 排序规则	AWS Blu Age 排序规则	AWS Micro Focus 排序规则
da-DK-cp1142-x-icu	da-DK-cp1142b-x-icu	da-DK-cp1142m-x-icu
da-DK-cp277-x-icu	da-DK-cp277b-x-icu	–
de-DE-cp1141-x-icu	de-DE-cp1141b-x-icu	de-DE-cp1141m-x-icu

EBCDIC 排序规则	AWS Blu Age 排序规则	AWS Micro Focus 排序规则
de-DE-cp273-x-icu	de-DE-cp273b-x-icu	–
en-GB-cp1146-x-icu	en-GB-cp1146b-x-icu	en-GB-cp1146m-x-icu
en-GB-cp285-x-icu	en-GB-cp285b-x-icu	–
en-US-cp037-x-icu	en-US-cp037b-x-icu	–
en-US-cp1140-x-icu	en-US-cp1140b-x-icu	en-US-cp1140m-x-icu
es-ES-cp1145-x-icu	es-ES-cp1145b-x-icu	es-ES-cp1145m-x-icu
es-ES-cp284-x-icu	es-ES-cp284b-x-icu	–
fi-FI-cp1143-x-icu	fi-FI-cp1143b-x-icu	fi-FI-cp1143m-x-icu
fi-FI-cp278-x-icu	fi-FI-cp278b-x-icu	–
fr-FR-cp1147-x-icu	fr-FR-cp1147b-x-icu	fr-FR-cp1147m-x-icu
fr-FR-cp297-x-icu	fr-FR-cp297b-x-icu	–
it-IT-cp1144-x-icu	it-IT-cp1144b-x-icu	it-IT-cp1144m-x-icu
it-IT-cp280-x-icu	it-IT-cp280b-x-icu	–
nl-BE-cp1148-x-icu	nl-BE-cp1148b-x-icu	nl-BE-cp1148m-x-icu
nl-BE-cp500-x-icu	nl-BE-cp500b-x-icu	–

要了解有关 AWS Blu Age 的更多信息，请参阅《AWS Mainframe Modernization 用户指南》中的教程：AWS Blu Age 的托管式运行时。

有关使用 AWS Micro Focus 的更多信息，请参阅《AWS Mainframe Modernization 用户指南》中的教程：Micro Focus 的托管式运行时。

有关在 PostgreSQL 中管理排序规则的更多信息，请参阅 PostgreSQL 文档中的排序规则支持。

Aurora PostgreSQL 中支持的排序规则

排序规则是一组规则，用于确定如何对存储在数据库中的字符串进行排序和比较。排序规则在计算机系统中起着重要作用，并作为操作系统的一部分包含在其中。当向语言中添加新字符或排序规则发生变化时，排序规则会随着时间的推移而变化。

排序规则库为排序规则定义特定的规则和算法。PostgreSQL 中最常用的排序规则库是 GNU C（Glibc）和 Unicode 国际化组件（ICU）。原定设置情况下，Aurora PostgreSQL 使用 Glibc 排序规则，其中包括适用于多字节字符序列的 Unicode 字符排序顺序。

当您创建新的 Aurora PostgreSQL 数据库集群时，它将检查操作系统中是否有可用的排序规则。`CREATE DATABASE` 命令的 PostgreSQL 参数 `LC_COLLATE` 和 `LC_CTYPE` 用于指定排序规则，该排序规则是该数据库中的原定设置排序规则。或者，您也可以在 `CREATE DATABASE` 中使用 `LOCALE` 参数来设置这些参数。这决定了数据库中字符串的原定设置排序规则以及将字符分类为字母、数字或符号的规则。您也可以选择用于列、索引或查询的排序规则。

Aurora PostgreSQL 依赖于操作系统中的 Glibc 库来提供排序规则支持。Aurora PostgreSQL 实例定期使用最新版本的操作系统进行更新。这些更新有时包含较新版本的 Glibc 库。较新版本的 Glibc 很少会更改某些字符的排序顺序或排序规则，这可能会导致数据的排序方式不同或生成无效的索引条目。如果您在更新期间发现用于排序规则的排序顺序有问题，则可能需要重建索引。

为了减少 Glibc 更新可能产生的影响，Aurora PostgreSQL 现在包含一个独立的原定设置排序规则库。这个排序规则库在 Aurora PostgreSQL 14.6、13.9、12.13、11.18 和更高的次要版本中可用。它与 Glibc 2.26-59.amzn2 兼容，并提供排序顺序稳定性以防止错误的查询结果。

Aurora PostgreSQL 函数参考

下面，您可以找到适用于运行 Aurora PostgreSQL 兼容版数据库引擎的 Aurora 数据库集群的 Aurora PostgreSQL 函数的列表。这些 Aurora PostgreSQL 函数是标准 PostgreSQL 函数的补充。有关标准 PostgreSQL 函数的更多信息，请参阅 PostgreSQL–函数和运算符。

概览

您可以为运行 Aurora PostgreSQL 的 Amazon RDS 数据库实例使用以下函数：

- aurora_db_instance_identifier
- aurora_ccm_status
- aurora_global_db_instance_status
- aurora_global_db_status

- aurora_list_builtins
- aurora_replica_status
- aurora_stat_activity
- aurora_stat_backend_wait
- aurora_stat_bgwriter
- aurora_stat_database
- aurora_stat_dml_activity
- aurora_stat_get_db_commit_latency
- aurora_stat_logical_wal_cache
- aurora_stat_memctx_usage
- aurora_stat_optimized_reads_cache
- aurora_stat_plans
- aurora_stat_reset_wal_cache
- aurora_stat_statements
- aurora_stat_system_waits
- aurora_stat_wait_event
- aurora_stat_wait_type
- aurora_version
- aurora_volume_logical_start_lsn
- aurora_wait_report

aurora_db_instance_identifier

报告您所连接的数据库实例的名称。

语法

```
aurora_db_instance_identifier()
```

参数

无

返回类型

VARCHAR 字符串

使用说明

此函数显示数据库客户端或应用程序连接的 Aurora PostgreSQL 兼容版集群的数据库实例名称。

此函数从 Aurora PostgreSQL 版本 13.7、12.11、11.16、10.21 以及所有其他更高版本开始提供。

示例

以下示例显示了调用 `aurora_db_instance_identifier` 函数的结果。

```
=> SELECT aurora_db_instance_identifier();
aurora_db_instance_identifier
------------------------------
 test-my-instance-name
```

您可以将此函数的结果与 `aurora_replica_status` 函数相结合，以获得有关连接的数据库实例的详细信息。aurora_replica_status 本身并不能显示您正在使用哪个数据库实例。下面的示例演示如何操作。

```
=> SELECT *
    FROM aurora_replica_status() rt,
         aurora_db_instance_identifier() di
    WHERE rt.server_id = di;
-[ RECORD 1 ]----------------------+----------------------
server_id                         | test-my-instance-name
session_id                        | MASTER_SESSION_ID
durable_lsn                       | 88492069
highest_lsn_rcvd                  |
current_read_lsn                  |
cur_replay_latency_in_usec        |
active_txns                       |
is_current                        | t
last_transport_error              | 0
last_error_timestamp              |
last_update_timestamp             | 2022-06-03 11:18:25+00
feedback_xmin                     |
feedback_epoch                    |
replica_lag_in_msec               |
```

```
log_stream_speed_in_kib_per_second | 0
log_buffer_sequence_number         | 0
oldest_read_view_trx_id            |
oldest_read_view_lsn               |
pending_read_ios                   | 819
```

aurora_ccm_status

显示集群缓存管理器的状态。

语法

```
aurora_ccm_status()
```

参数

无。

返回类型

包含以下列的 SETOF 记录：

- buffers_sent_last_minute – 过去一分钟内发送到指定读取器的缓冲区数量。
- buffers_found_last_minute – 在过去一分钟内识别的频繁访问缓冲区的数量。
- buffers_sent_last_scan – 在上次完整扫描缓冲区缓存期间，发送到指定读取器的缓冲区数量。
- buffers_found_last_scan – 在上次完整扫描缓冲区缓存期间，发送的经常访问缓冲区数量。已在指定读取器上缓存的缓冲区不会发送。
- buffers_sent_current_scan – 在当前的扫描期间发送的缓冲区数量。
- buffers_found_current_scan – 在当前扫描中识别的经常访问缓冲区的数量。
- current_scan_progress – 在当前的扫描期间，到现在已访问的缓冲区数量。

使用说明

您可以使用此功能来检查和监控集群缓存管理 (CCM) 功能。仅当 CCM 在 Aurora PostgreSQL 数据库集群上处于活动状态时，此函数才有效。要使用此函数，您需要连接到 Aurora PostgreSQL 数据库集群上的写入数据库实例。

为 Aurora PostgreSQL 数据库集群启用 CCM，方法是在集群的自定义数据库集群参数组中，将 apg_ccm_enabled 设置为 1。要了解如何操作，请参阅配置集群缓存管理。

当集群具有如下配置的 Aurora 读取器实例时，集群缓存管理在 Aurora PostgreSQL 数据库集群上处于活动状态：

- Aurora 读取器实例使用与集群的写入器实例相同的数据库实例类类型和大小。
- Aurora 读取器实例配置为集群的 Tier-0。如果集群有多个读取器，则这是唯一的 Tier-0 读取器。

将多个读取器设置为 Tier-0 将会禁用 CCM。CCM 禁用时，调用此函数将返回以下错误消息：

```
ERROR: Cluster Cache Manager is disabled
```

您也可以使用 PostgreSQL pg_buffercache 扩展来分析缓冲区缓存。有关更多信息，请参阅 PostgreSQL 文档中的 pg_buffercache。

有关更多信息，请参阅 Aurora PostgreSQL 集群缓存管理简介。

示例

以下示例显示了调用 aurora_ccm_status 函数的结果。此第一个示例显示了 CCM 统计数据。

```
=> SELECT * FROM aurora_ccm_status();
 buffers_sent_last_minute | buffers_found_last_minute | buffers_sent_last_scan |
 buffers_found_last_scan | buffers_sent_current_scan | buffers_found_current_scan |
 current_scan_progress
--------------------------+---------------------------+------------------------
+-------------------------+---------------------------+---------------------------
+------------------------
                   2242000 |                   2242003 |               17920442 |
         17923410 |                  14098000 |                  14100964 |
   15877443
```

有关更完整的详细信息，您可以使用扩展显示，如下所示：

```
\x
Expanded display is on.
SELECT *  FROM aurora_ccm_status();
[ RECORD 1 ]----------------------+---------
buffers_sent_last_minute          | 2242000
```

```
buffers_found_last_minute          | 2242003
buffers_sent_last_scan             | 17920442
buffers_found_last_scan            | 17923410
buffers_sent_current_scan          | 14098000
buffers_found_current_scan         | 14100964
current_scan_progress              | 15877443
```

此示例说明如何检查温速率和温百分比。

```
=> SELECT buffers_sent_last_minute * 8/60 AS warm_rate_kbps,
100 * (1.0-buffers_sent_last_scan/buffers_found_last_scan) AS warm_percent
FROM aurora_ccm_status ();
 warm_rate_kbps | warm_percent
----------------+--------------
 16523 |        100.0
```

aurora_global_db_instance_status

显示所有 Aurora 实例的状态，包括 Aurora 全局数据库集群中的副本。

语法

```
aurora_global_db_instance_status()
```

参数

无

返回类型

包含以下列的 SETOF 记录：

- server_id – 数据库实例的标识符。
- session_id – 当前会话的唯一标识符。MASTER_SESSION_ID 的值标识写入器（主要）数据库实例。
- aws_region – 此全球数据库实例在其中运行的 AWS 区域。有关区域列表，请参阅区域可用性。
- durable_lsn – 在存储中变得持久的日志序列号（LSN）。日志序列号 (LSN) 是标识数据库事务日志中的记录的唯一序列号。对 LSN 进行排序，以便较大的 LSN 表示较晚的事务。

- highest_lsn_rcvd – 数据库实例从写入器数据库实例收到的最高 LSN。
- feedback_epoch – 数据库实例在生成热备用信息时使用的纪元。热备用服务器是在主数据库处于恢复或备用模式时支持连接和查询的数据库实例。热备用服务器信息包括纪元（时间点）和有关用作热备用服务器的数据库实例的其他详细信息。有关更多信息，请参阅 PostgreSQL 文档中的热备用。
- feedback_xmin – 数据库实例使用的最小（最早）活动事务 ID。
- oldest_read_view_lsn – 数据库实例用于从存储中读取的最早 LSN。
- visibility_lag_in_msec – 此数据库实例滞后于写入器数据库实例的时间（以毫秒为单位）。

使用说明

此函数显示 Aurora 数据库集群的复制统计数据。对于集群中的每个 Aurora PostgreSQL 数据库实例，该函数显示一行数据，其中包括全局数据库配置中的任何跨区域副本。

您可以从 Aurora PostgreSQL 数据库集群或 Aurora PostgreSQL 全局数据库中的任何实例运行此函数。该函数返回有关所有副本实例滞后的详细信息。

要了解有关使用此函数（aurora_global_db_instance_status）或使用 aurora_global_db_status 监控滞后的更多信息，请参阅 监控基于 Aurora PostgreSQL 的全局数据库。

有关 Aurora 全局数据库的信息，请参阅Amazon Aurora Global Database 概览。

要开始使用 Aurora 全局数据库，请参阅 开始使用 Amazon Aurora Global Database 或参阅 Amazon Aurora 常见问题。

示例

此示例显示跨区域实例统计数据。

```
=> SELECT *
  FROM aurora_global_db_instance_status();
               server_id                |              session_id              |
  aws_region  | durable_lsn | highest_lsn_rcvd | feedback_epoch | feedback_xmin |
 oldest_read_view_lsn | visibility_lag_in_msec
--------------------------------------+--------------------------------------
+-------------+-------------+------------------+----------------+---------------
+----------------------+------------------------
 db-119-001-instance-01                   | MASTER_SESSION_ID                    | eu-
west-1    |  2534560273 |           [NULL] |         [NULL] |        [NULL] |
     [NULL] |                 [NULL]
```

```
 db-119-001-instance-02                    | 4ecff34d-d57c-409c-ba28-278b31d6fc40 | eu-
west-1    |  2534560266 |       2534560273 |              0 |      19669196 |
 2534560266 |                       6
 db-119-001-instance-03                    | 3e8a20fc-be86-43d5-95e5-bdf19d27ad6b | eu-
west-1    |  2534560266 |       2534560273 |              0 |      19669196 |
 2534560266 |                       6
 db-119-001-instance-04                    | fc1b0023-e8b4-4361-bede-2a7e926cead6 | eu-
west-1    |  2534560266 |       2534560273 |              0 |      19669196 |
 2534560254 |                      23
 db-119-001-instance-05                    | 30319b74-3f08-4e13-9728-e02aa1aa8649 | eu-
west-1    |  2534560266 |       2534560273 |              0 |      19669196 |
 2534560254 |                      23
 db-119-001-global-instance-1              | a331ffbb-d982-49ba-8973-527c96329c60 | eu-
central-1 |  2534560254 |       2534560266 |              0 |      19669196 |
 2534560247 |                     996
 db-119-001-global-instance-1              | e0955367-7082-43c4-b4db-70674064a9da | eu-
west-2    |  2534560254 |       2534560266 |              0 |      19669196 |
 2534560247 |                      14
 db-119-001-global-instance-1-eu-west-2a   | 1248dc12-d3a4-46f5-a9e2-85850491a897 | eu-
west-2    |  2534560254 |       2534560266 |              0 |      19669196 |
 2534560247 |                       0
```

此示例说明如何检查全局副本滞后（以毫秒为单位）。

```
=> SELECT CASE
          WHEN 'MASTER_SESSION_ID' = session_id THEN 'Primary'
         ELSE 'Secondary'
      END AS global_role,
      aws_region,
      server_id,
      visibility_lag_in_msec
  FROM aurora_global_db_instance_status()
  ORDER BY 1, 2, 3;
   global_role |  aws_region  |               server_id                |
 visibility_lag_in_msec
-------------+--------------+----------------------------------------
+------------------------
 Primary     | eu-west-1    | db-119-001-instance-01                 |
 [NULL]
 Secondary   | eu-central-1 | db-119-001-global-instance-1           |
     13
 Secondary   | eu-west-1    | db-119-001-instance-02                 |
     10
```

```
 Secondary   | eu-west-1    | db-119-001-instance-03                  |
      9
 Secondary   | eu-west-1    | db-119-001-instance-04                  |
      2
 Secondary   | eu-west-1    | db-119-001-instance-05                  |
     18
 Secondary   | eu-west-2    | db-119-001-global-instance-1            |
     14
 Secondary   | eu-west-2    | db-119-001-global-instance-1-eu-west-2a |
     13
```

此示例说明如何从全局数据库配置中检查每个 AWS 区域 的最小、最大和平均滞后。

```
=> SELECT 'Secondary' global_role,
       aws_region,
       min(visibility_lag_in_msec) min_lag_in_msec,
       max(visibility_lag_in_msec) max_lag_in_msec,
       round(avg(visibility_lag_in_msec),0) avg_lag_in_msec
  FROM aurora_global_db_instance_status()
 WHERE aws_region NOT IN (SELECT   aws_region
                            FROM aurora_global_db_instance_status()
                           WHERE session_id='MASTER_SESSION_ID')
                        GROUP BY aws_region
UNION ALL
SELECT  'Primary' global_role,
        aws_region,
        NULL,
        NULL,
        NULL
    FROM aurora_global_db_instance_status()
   WHERE session_id='MASTER_SESSION_ID'
ORDER BY 1, 5;
 global_role |  aws_region  | min_lag_in_msec | max_lag_in_msec | avg_lag_in_msec
-------------+--------------+-----------------+-----------------+-----------------
 Primary     | eu-west-1    |          [NULL] |          [NULL] |          [NULL]
 Secondary   | eu-central-1 |             133 |             133 |             133
 Secondary   | eu-west-2    |               0 |             495 |             248
```

aurora_global_db_status

显示有关 Aurora 全局数据库滞后各方面的信息，特别是底层 Aurora 存储的滞后（所谓持久性滞后）以及恢复点目标（RPO）之间的滞后。

语法

```
aurora_global_db_status()
```

参数

无。

返回类型

包含以下列的 SETOF 记录：

- aws_region – 此数据库集群所在的 AWS 区域。有关按引擎排列的完整 AWS 区域列表，请参阅区域及可用区。
- highest_lsn_written – 此数据库集群上当前存在的最高日志序列号（LSN）。日志序列号 (LSN) 是标识数据库事务日志中的记录的唯一序列号。对 LSN 进行排序，以便较大的 LSN 表示较晚的事务。
- durability_lag_in_msec – 辅助数据库集群上的 highest_lsn_written 与主数据库集群上的 highest_lsn_written 之间的时间戳值差异。值为 -1 表示 Aurora 全局数据库的主数据库集群。
- rpo_lag_in_msec – 恢复点目标（RPO）滞后。RPO 滞后是最近的用户事务在存储在 Aurora 全局数据库的主数据库集群上之后，执行 COMMIT 操作以便存储在辅助数据库集群上所需的时间。值为 -1 表示主数据库集群（因此滞后无关紧要）。

  简而言之，该指标计算 Aurora 全局数据库中每个 Aurora PostgreSQL 数据库集群的恢复点目标，即如果发生中断，可能会丢失多少数据。与滞后一样，RPO 是按时间计量的。
- last_lag_calculation_time – 指定上次为 durability_lag_in_msec 和 rpo_lag_in_msec 计算值的时间戳。时间值（如 1970-01-01 00:00:00+00）表示这是主数据库集群。
- feedback_epoch – 辅助数据库集群在生成热备用信息时使用的纪元。热备用服务器是在主数据库处于恢复或备用模式时支持连接和查询的数据库实例。热备用服务器信息包括纪元（时间点）和有关用作热备用服务器的数据库实例的其他详细信息。有关更多信息，请参阅 PostgreSQL 文档中的热备用。
- feedback_xmin – 辅助数据库集群使用的最小（最早）活动事务 ID。

使用说明

Aurora PostgreSQL 的当前所有可用版本都支持此功能。此函数显示 Aurora 全局数据库的复制统计数据。它为 Aurora PostgreSQL 全局数据库中的每个数据库集群显示一行。您可以从 Aurora PostgreSQL 全局数据库中的任何实例运行此函数。

要评估 Aurora 全局数据库复制滞后（即可见的数据滞后），请参阅 aurora_global_db_instance_status。

要了解有关使用 `aurora_global_db_status` 和 `aurora_global_db_instance_status` 监控 Aurora 全局数据库滞后的更多信息，请参阅 监控基于 Aurora PostgreSQL 的全局数据库。有关 Aurora 全局数据库的信息，请参阅Amazon Aurora Global Database 概览。

示例

此示例说明如何显示跨区域存储统计数据。

```
=> SELECT CASE
          WHEN '-1' = durability_lag_in_msec THEN 'Primary'
          ELSE 'Secondary'
       END AS global_role,
       *
  FROM aurora_global_db_status();
 global_role | aws_region | highest_lsn_written | durability_lag_in_msec |
 rpo_lag_in_msec | last_lag_calculation_time  | feedback_epoch | feedback_xmin
-------------+------------+---------------------+------------------------
+-----------------+----------------------------+----------------+---------------
 Primary     | eu-west-1  |           131031557 |                     -1 |
 -1 | 1970-01-01 00:00:00+00     |             0 |              0
 Secondary   | eu-west-2  |           131031554 |                    410 |
  0 | 2021-06-01 18:59:36.124+00 |             0 |         12640
 Secondary   | eu-west-3  |           131031554 |                    410 |
  0 | 2021-06-01 18:59:36.124+00 |             0 |         12640
```

aurora_list_builtins

列出所有可用的 Aurora PostgreSQL 内置函数，以及简要说明和函数详细信息。

语法

```
aurora_list_builtins()
```

参数

无

返回类型

SETOF 记录

示例

以下示例显示了调用 aurora_list_builtins 函数的结果。

```
=> SELECT *
FROM aurora_list_builtins();

               Name                | Result data type |                   Argument data
 types                     | Type | Volatility |  Parallel  | Security |
             Description
-----------------------------------+------------------
+-----------------------------------------------------------+------+------------
+------------+----------
+-------------------------------------------------------------------------
 aurora_version                    | text             |
                           | func | stable     | safe       | invoker  | Amazon Aurora
 PostgreSQL-Compatible Edition version string
 aurora_stat_wait_type             | SETOF record     | OUT type_id smallint, OUT
 type_name text                 | func | volatile   | restricted | invoker  | Lists all
 supported wait types
 aurora_stat_wait_event            | SETOF record     | OUT type_id smallint, OUT
 event_id integer, OUT event_na.| func | volatile   | restricted | invoker  | Lists all
 supported wait events
                                   |                  |.me text
                           |      |            |            |          |
 aurora_list_builtins              | SETOF record     | OUT "Name" text, OUT "Result
 data type" text, OUT "Argum.| func | stable     | safe       | invoker  | Lists all
 Aurora built-in functions
                                   |                  |.ent data types" text, OUT
 "Type" text, OUT "Volatility" .|      |            |            |          |
                                   |                  |.text, OUT "Parallel" text, OUT
 "Security" text, OUT "Des.|      |            |            |          |
                                   |                  |.cription" text
                           |      |            |            |          |
 .
 .
```

```
.
aurora_stat_file                  | SETOF record     | OUT filename text, OUT
allocated_bytes bigint, OUT used_.| func | stable     | safe       | invoker  | Lists
all files present in Aurora storage
                                  |                  |.bytes bigint
                          |      |           |           |          |
aurora_stat_get_db_commit_latency | bigint           | oid
                           | func | stable     | restricted | invoker  | Per DB commit
latency in microsecs
```

aurora_replica_status

显示所有 Aurora PostgreSQL 读取器节点的状态。

语法

```
aurora_replica_status()
```

参数

无

返回类型

包含以下列的 SETOF 记录：

- `server_id` – 数据库实例 ID（标识符）。
- `session_id` – 当前会话的唯一标识符，针对主实例和读取器实例返回，如下所示：
  - 对于主实例，`session_id` 始终为 `MASTER_SESSION_ID’。
  - 对于读取器实例，`session_id` 始终为读取器实例的 UUID（通用唯一标识符）。
- `durable_lsn` – 保存在存储中的日志序列号 (LSN)。
  - 对于主卷，该值是当前生效的主卷持久性 LSN (VDL)。
  - 对于任何辅助卷，该值是已成功应用辅助卷的主卷的 VDL。

> **Note**
>
> 日志序列号 (LSN) 是标识数据库事务日志中的记录的唯一序列号。对 LSN 进行排序，以便较大的 LSN 表示序列中较晚发生的事务。

- highest_lsn_rcvd – 数据库实例从写入器数据库实例收到的最高（最新）LSN。
- current_read_lsn – 应用于所有读取器的最新快照的 LSN。
- cur_replay_latency_in_usec – 在辅助卷上重播日志所需的微秒数。
- active_txns – 当前处于活动状态的事务数量。
- is_current – 未使用。
- last_transport_error – 上次复制错误代码。
- last_error_timestamp – 上次复制错误的时间戳。
- last_update_timestamp – 上次更新副本状态的时间戳。在 Aurora PostgreSQL 13.9 中，您所连接的数据库实例的 last_update_timestamp 值设置为 NULL。
- feedback_xmin – 副本的热备用 feedback_xmin。数据库实例使用的最小（最早）活动事务 ID。
- feedback_epoch – 数据库实例在生成热备用信息时使用的纪元。
- replica_lag_in_msec – 读取器实例滞后于写入器实例的时间（以毫秒为单位）。
- log_stream_speed_in_kib_per_second – 日志流吞吐量（以每秒千字节为单位）。
- log_buffer_sequence_number – 日志缓冲区序列号。
- oldest_read_view_trx_id – 未使用。
- oldest_read_view_lsn – 数据库实例用于从存储中读取的最早 LSN。
- pending_read_ios – 未完成的页面读取在副本上仍处于待处理状态。
- read_ios – 副本上的页面读取总数。
- iops – 未使用。
- cpu – 集群中每个节点的 Aurora 存储进程守护程序的 CPU 使用率。有关实例的 CPU 使用情况的信息，请参阅Amazon Aurora 的实例级指标。

使用说明

Aurora PostgreSQL 的当前所有可用版本都支持此功能。aurora_replica_status 函数从 Aurora PostgreSQL 数据库集群的副本状态管理器返回值。您可以使用此函数获取有关 Aurora PostgreSQL 数据库集群上复制状态的信息，包括 Aurora 数据库集群中所有数据库实例的指标。例如，您可以执行以下操作：

- 获取有关 Aurora PostgreSQL 数据库集群中实例类型（写入器、读取器）的信息 – 您可以通过检查以下列的值来获取此信息：
  - server_id – 包含创建实例时指定的实例名称。在某些情况下，例如对于主（写入器）实例，通常会通过在为 Aurora PostgreSQL 数据库集群创建的名称后面附加 -instance-1 来为您创建名称。

  - session_id – session_id 字段指示实例是读取器还是写入器。对于写入器实例，session_id 始终设置为 "MASTER_SESSION_ID"。对于读取器实例，session_id 已设置为特定读取器的 UUID。
- 诊断常见的复制问题，例如副本滞后 – 副本滞后是指读取器实例的页面缓存落后于写入器实例页面缓存的时间（以毫秒为单位）。出现此滞后的原因是 Aurora 集群使用异步复制，如 使用 Amazon Aurora 进行复制中所述。它显示在该函数返回的结果中的 replica_lag_in_msec 列中。当由于与备用服务器上的恢复冲突而取消查询时，也可能会发生滞后。您可以检查 pg_stat_database_conflicts() 以验证此类冲突是否导致副本滞后。有关更多信息，请参阅 PostgreSQL 文档中的统计数据收集器。要了解有关高可用性和复制的更多信息，请参阅 Amazon Aurora 常见问题。

  Amazon CloudWatch 会随着时间的推移存储 replica_lag_in_msec 结果，作为 AuroraReplicaLag 指标。有关将 CloudWatch 指标用于 Aurora 的信息，请参阅使用 Amazon CloudWatch 监控 Amazon Aurora 指标。

要了解有关 Aurora 只读副本故障排查和重新启动的更多信息，请参阅 AWS 支持 中心内的为什么我的 Amazon Aurora 只读副本滞后并重新启动？

示例

以下示例说明如何获取 Aurora PostgreSQL 数据库集群中所有实例的复制状态：

```
=> SELECT *
FROM aurora_replica_status();
```

以下示例显示 docs-lab-apg-main Aurora PostgreSQL 数据库集群中的写入器实例：

```
=> SELECT server_id,
    CASE
        WHEN 'MASTER_SESSION_ID' = session_id THEN 'writer'
        ELSE 'reader'
    END AS instance_role
FROM aurora_replica_status()
WHERE session_id = 'MASTER_SESSION_ID';
        server_id       | instance_role
------------------------+---------------
 db-119-001-instance-01 | writer
```

以下示例列出集群中的所有读取器实例：

```
=> SELECT server_id,
    CASE
        WHEN 'MASTER_SESSION_ID' = session_id THEN 'writer'
        ELSE 'reader'
    END AS instance_role
FROM aurora_replica_status()
WHERE session_id <> 'MASTER_SESSION_ID';
        server_id       | instance_role
------------------------+---------------
db-119-001-instance-02  | reader
db-119-001-instance-03  | reader
db-119-001-instance-04  | reader
db-119-001-instance-05  | reader
(4 rows)
```

以下示例列出所有实例、每个实例滞后于写入器的程度，以及自上次更新以来经过的时间：

```
=> SELECT server_id,
    CASE
        WHEN 'MASTER_SESSION_ID' = session_id THEN 'writer'
        ELSE 'reader'
    END AS instance_role,
    replica_lag_in_msec AS replica_lag_ms,
    round(extract (epoch FROM (SELECT age(clock_timestamp(), last_update_timestamp))) *
 1000) AS last_update_age_ms
FROM aurora_replica_status()
ORDER BY replica_lag_in_msec NULLS FIRST;
       server_id        | instance_role | replica_lag_ms | last_update_age_ms
------------------------+---------------+----------------+--------------------
 db-124-001-instance-03 | writer        |         [NULL] |               1756
 db-124-001-instance-01 | reader        |             13 |               1756
 db-124-001-instance-02 | reader        |             13 |               1756
(3 rows)
```

aurora_stat_activity

每个服务器进程返回一行，显示与该进程的当前活动相关的信息。

语法

```
aurora_stat_activity();
```

参数

无

返回类型

每个服务器进程返回一行。除了 pg_stat_activity 列之外，还添加了以下字段：

- planid：计划标识符

使用说明

pg_stat_activity 的一个补充视图，可返回相同的列以及显示当前查询执行计划的额外 plan_id 列。

必须启用 aurora_compute_plan_id，视图才能返回 plan_id。

Aurora PostgreSQL 版本 14.10、15.5 以及所有其它更高版本都提供了此函数。

示例

下面的示例查询按 query_id 和 plan_id 汇总了最高负载。

```
db1=# select count(*), query_id, plan_id
db1-# from aurora_stat_activity() where state = 'active'
db1-# and pid <> pg_backend_pid()
db1-# group by query_id, plan_id
db1-# order by 1 desc;

count |  query_id             |  plan_id
-------+-----------------------+-------------
 11    | -5471422286312252535 | -2054628807
 3     | -6907107586630739258 | -815866029
 1     | 5213711845501580017  |  300482084
(3 rows)
```

如果用于 query_id 的计划发生变化，aurora_stat_activity 将报告新的 plan_id。

```
count  |  query_id            |  plan_id
-------+----------------------+-------------
 10    | -5471422286312252535 | 1602979607
 1     | -6907107586630739258 | -1809935983
 1     | -2446282393000597155 | -207532066
(3 rows)
```

aurora_stat_backend_wait

显示特定后端进程等待活动的统计数据。

语法

```
aurora_stat_backend_waits(pid)
```

参数

pid – 后端进程的 ID。您可以通过使用 pg_stat_activity 视图获取进程 ID。

返回类型

包含以下列的 SETOF 记录：

- type_id – 表示等待事件类型的数字，例如 1 表示轻量级锁（LWLock），3 表示锁，或者 6 表示客户端会话。当您将此函数的结果与 aurora_stat_wait_type 函数中的列连接时，这些值变得有意义，如示例中所示。
- event_id – 等待事件的识别号码。将此值与来自 aurora_stat_wait_event 的列连接以获取有意义的事件名称。
- waits – 指定进程 ID 累积的等待次数的计数。
- wait_time – 以毫秒为单位的等待时间。

使用说明

您可以使用此函数分析自连接打开以来发生的特定后端（会话）等待事件。要获取更多有关等待事件名称和类型的有意义信息，您可以通过使用 JOIN 组合此函数 aurora_stat_wait_type 和 aurora_stat_wait_event，如示例所示。

示例

此示例显示了后端进程 ID 3027 的所有等待、类型和事件名称。

```
=> SELECT type_name, event_name, waits, wait_time
        FROM aurora_stat_backend_waits(3027)
NATURAL JOIN aurora_stat_wait_type()
NATURAL JOIN aurora_stat_wait_event();
type_name |       event_name       | waits | wait_time
-----------+------------------------+-------+------------
 LWLock    | ProcArrayLock          |     3 |         27
 LWLock    | wal_insert             |   423 |      16336
 LWLock    | buffer_content         | 11840 |    1033634
 LWLock    | lock_manager           | 23821 |    5664506
 Lock      | tuple                  | 10258 |  152280165
 Lock      | transactionid          | 78340 | 1239808783
 Client    | ClientRead             | 34072 |   17616684
 IO        | ControlFileSyncUpdate  |     2 |          0
 IO        | ControlFileWriteUpdate |     4 |         32
 IO        | RelationMapRead        |     2 |        795
 IO        | WALWrite               | 36666 |      98623
 IO        | XactSync               |  4867 |    7331963
```

此示例显示了所有活动会话的当前和累积等待类型及等待事件（pg_stat_activity state <> 'idle'）[但不包括调用此函数的当前会话（pid <> pg_backend_pid()）]。

```
=> SELECT a.pid,
            a.usename,
            a.app_name,
            a.current_wait_type,
            a.current_wait_event,
            a.current_state,
            wt.type_name AS wait_type,
            we.event_name AS wait_event,
            a.waits,
            a.wait_time
        FROM (SELECT pid,
                     usename,
                     left(application_name,16) AS app_name,
                     coalesce(wait_event_type,'CPU') AS current_wait_type,
                     coalesce(wait_event,'CPU') AS current_wait_event,
                     state AS current_state,
                     (aurora_stat_backend_waits(pid)).*
```

```
                FROM pg_stat_activity
               WHERE pid <> pg_backend_pid()
                 AND state <> 'idle') a
NATURAL JOIN aurora_stat_wait_type() wt
NATURAL JOIN aurora_stat_wait_event() we;
  pid  | usename  | app_name | current_wait_type | current_wait_event | current_state |
 wait_type |       wait_event       | waits | wait_time
-------+----------+----------+-------------------+--------------------+---------------
+-----------+------------------------+-------+-----------
 30099 | postgres | pgbench  | Lock              | transactionid      | active        |
 LWLock    | wal_insert             |  1937 |     29975
 30099 | postgres | pgbench  | Lock              | transactionid      | active        |
 LWLock    | buffer_content         | 22903 |    760498
 30099 | postgres | pgbench  | Lock              | transactionid      | active        |
 LWLock    | lock_manager           | 10012 |    223207
 30099 | postgres | pgbench  | Lock              | transactionid      | active        |
 Lock      | tuple                  | 20315 |  63081529
 .
 .
 .
 30099 | postgres | pgbench  | Lock              | transactionid      | active        |
 IO        | WALWrite               | 93293 |    237440
 30099 | postgres | pgbench  | Lock              | transactionid      | active        |
 IO        | XactSync               | 13010 |  19525143
 30100 | postgres | pgbench  | Lock              | transactionid      | active        |
 LWLock    | ProcArrayLock          |     6 |        53
 30100 | postgres | pgbench  | Lock              | transactionid      | active        |
 LWLock    | wal_insert             |  1913 |     25450
 30100 | postgres | pgbench  | Lock              | transactionid      | active        |
 LWLock    | buffer_content         | 22874 |    778005
 .
 .
 .
 30109 | postgres | pgbench  | IO                | XactSync           | active        |
 LWLock    | ProcArrayLock          |     3 |        71
 30109 | postgres | pgbench  | IO                | XactSync           | active        |
 LWLock    | wal_insert             |  1940 |     27741
 30109 | postgres | pgbench  | IO                | XactSync           | active        |
 LWLock    | buffer_content         | 22962 |    776352
 30109 | postgres | pgbench  | IO                | XactSync           | active        |
 LWLock    | lock_manager           |  9879 |    218826
 30109 | postgres | pgbench  | IO                | XactSync           | active        |
 Lock      | tuple                  | 20401 |  63581306
```

```
 30109 | postgres | pgbench  | IO               | XactSync          | active        |
 Lock      | transactionid          | 50769 | 211645008
 30109 | postgres | pgbench  | IO               | XactSync          | active        |
 Client    | ClientRead             | 89901 |  44192439
```

此示例显示了所有活动会话的当前和前三（3）个累积等待类型和等待事件（pg_stat_activity state <> 'idle'），不包括当前会话（pid <>pg_backend_pid()）。

```
=> SELECT top3.*
       FROM (SELECT a.pid,
                    a.usename,
                    a.app_name,
                    a.current_wait_type,
                    a.current_wait_event,
                    a.current_state,
                    wt.type_name AS wait_type,
                    we.event_name AS wait_event,
                    a.waits,
                    a.wait_time,
                    RANK() OVER (PARTITION BY pid ORDER BY a.wait_time DESC)
               FROM (SELECT pid,
                            usename,
                            left(application_name,16) AS app_name,
                            coalesce(wait_event_type,'CPU') AS current_wait_type,
                            coalesce(wait_event,'CPU') AS current_wait_event,
                            state AS current_state,
                            (aurora_stat_backend_waits(pid)).*
                       FROM pg_stat_activity
                      WHERE pid <> pg_backend_pid()
                        AND state <> 'idle') a
       NATURAL JOIN aurora_stat_wait_type() wt
       NATURAL JOIN aurora_stat_wait_event() we) top3
 WHERE RANK <=3;
  pid  | usename  | app_name | current_wait_type | current_wait_event | current_state |
 wait_type |   wait_event    |  waits  | wait_time  | rank
-------+----------+----------+-------------------+--------------------+---------------
+-----------+-----------------+---------+------------+------
 20567 | postgres | psql     | CPU               | CPU                | active        |
 LWLock    | wal_insert      |   25000 |   67512003 |    1
 20567 | postgres | psql     | CPU               | CPU                | active        |
 IO        | WALWrite        | 3071758 |    1016961 |    2
 20567 | postgres | psql     | CPU               | CPU                | active        |
 IO        | BufFileWrite    |   20750 |     184559 |    3
```

```
 27743 | postgres | pgbench  | Lock              | transactionid      | active        |
 Lock      | transactionid   |  237350 | 1265580011 |    1
 27743 | postgres | pgbench  | Lock              | transactionid      | active        |
 Lock      | tuple           |   93641 |  341472318 |    2
 27743 | postgres | pgbench  | Lock              | transactionid      | active        |
 Client    | ClientRead      |  417556 |  204796837 |    3
 .
 .
 .
 27745 | postgres | pgbench  | IO                | XactSync           | active        |
 Lock      | transactionid   |  238068 | 1265816822 |    1
 27745 | postgres | pgbench  | IO                | XactSync           | active        |
 Lock      | tuple           |   93210 |  338312247 |    2
 27745 | postgres | pgbench  | IO                | XactSync           | active        |
 Client    | ClientRead      |  419157 |  207836533 |    3
 27746 | postgres | pgbench  | Lock              | transactionid      | active        |
 Lock      | transactionid   |  237621 | 1264528811 |    1
 27746 | postgres | pgbench  | Lock              | transactionid      | active        |
 Lock      | tuple           |   93563 |  339799310 |    2
 27746 | postgres | pgbench  | Lock              | transactionid      | active        |
 Client    | ClientRead      |  417304 |  208372727 |    3
```

aurora_stat_bgwriter

`aurora_stat_bgwriter` 是一个统计视图，显示有关优化型读取缓存写入的信息。

语法

```
aurora_stat_bgwriter()
```

参数

无

返回类型

包含所有 `pg_stat_bgwriter` 列和以下附加列的 SETOF 记录。有关 `pg_stat_bgwriter` 列的更多信息，请参阅 pg_stat_bgwriter。

您可以使用 `pg_stat_reset_shared("bgwriter")` 重置此函数的统计信息。

- `orcache_blks_written` – 已写入的优化型读取缓存数据块的总数。

- orcache_blk_write_time – 如果已启用 track_io_timing，它将跟踪写入优化型读取缓存数据文件块所花费的总时间（以毫秒为单位）。有关更多信息，请参阅 track_io_timing。

使用说明

此函数可用于以下 Aurora PostgreSQL 版本：

- 15.4 及更高的 15 版本
- 14.9 及更高的 14 版本

示例

```
=> select * from aurora_stat_bgwriter();
-[ RECORD 1 ]-----------------+-----------
orcache_blks_written          | 246522
orcache_blk_write_time        | 339276.404
```

aurora_stat_database

它携带 pg_stat_database 的所有列，并在最后添加新列。

语法

```
aurora_stat_database()
```

参数

无

返回类型

包含所有 pg_stat_database 列和以下附加列的 SETOF 记录。有关 pg_stat_database 列的更多信息，请参阅 pg_stat_database。

- storage_blks_read – 从该数据库的 aurora 存储中读取的共享块总数。
- orcache_blks_hit – 此数据库中优化型读取缓存命中总数。
- local_blks_read – 在此数据库中读取的本地块总数。

- `storage_blk_read_time` – 如果已启用 `track_io_timing`，它将跟踪从 aurora 存储中读取数据文件块所花费的总时间（以毫秒为单位），否则该值为零。有关更多信息，请参阅 track_io_timing。
- `local_blk_read_time` – 如果已启用 `track_io_timing`，它将跟踪读取本地数据文件块所花费的总时间（以毫秒为单位），否则该值为零。有关更多信息，请参阅 track_io_timing。
- `orcache_blk_read_time` – 如果已启用 `track_io_timing`，它将跟踪从优化型读取缓存中读取数据文件块所花费的总时间（以毫秒为单位），否则该值为零。有关更多信息，请参阅 track_io_timing。

> **Note**
>
> `blks_read` 的值是`storage_blks_read`、`orcache_blks_hit` 和 `local_blks_read` 的总和。
> `blk_read_time` 的值是`storage_blk_read_time`、`orcache_blk_read_time` 和 `local_blk_read_time` 的总和。

使用说明

此函数可用于以下 Aurora PostgreSQL 版本：

- 15.4 及更高的 15 版本
- 14.9 及更高的 14 版本

示例

以下示例显示了它如何携带所有 `pg_stat_database` 列并在最后附加 6 个新列：

```
=> select * from aurora_stat_database() where datid=14717;
-[ RECORD 1 ]------------+------------------------------
datid                    | 14717
datname                  | postgres
numbackends              | 1
xact_commit              | 223
xact_rollback            | 4
blks_read                | 1059
blks_hit                 | 11456
tup_returned             | 27746
```

```
tup_fetched              | 5220
tup_inserted             | 165
tup_updated              | 42
tup_deleted              | 91
conflicts                | 0
temp_files               | 0
temp_bytes               | 0
deadlocks                | 0
checksum_failures        |
checksum_last_failure    |
blk_read_time            | 3358.689
blk_write_time           | 0
session_time             | 1076007.997
active_time              | 3684.371
idle_in_transaction_time | 0
sessions                 | 10
sessions_abandoned       | 0
sessions_fatal           | 0
sessions_killed          | 0
stats_reset              | 2023-01-12 20:15:17.370601+00
orcache_blks_hit         | 425
orcache_blk_read_time    | 89.934
storage_blks_read        | 623
storage_blk_read_time    | 3254.914
local_blks_read          | 0
local_blk_read_time      | 0
```

aurora_stat_dml_activity

报告 Aurora PostgreSQL 集群中数据库上每种类型的数据操作语言（DML）操作的累积活动。

语法

```
aurora_stat_dml_activity(database_oid)
```

参数

database_oid

Aurora PostgreSQL 集群中数据库的对象 ID（OID）。

返回类型

SETOF 记录

使用说明

aurora_stat_dml_activity 函数仅适用于 Aurora PostgreSQL 版本 3.1，与 PostgreSQL 引擎 11.6 及更高版本兼容。

在具有大量数据库的 Aurora PostgreSQL 集群上使用此函数，以确定哪些数据库具有更多或更慢的 DML 活动，或两者兼有。

aurora_stat_dml_activity 函数返回运行操作的次数以及选择、插入、更新和删除操作的累积延迟（以微秒为单位）。该报告仅包括成功的 DML 操作。

您可以使用 PostgreSQL 统计信息访问函数 pg_stat_reset 重置此统计信息。您可以使用 pg_stat_get_db_stat_reset_time 函数检查上次重置此统计信息的时间。有关 PostgreSQL 统计访问函数的更多信息，请参阅 PostgreSQL 文档中的统计收集器。

示例

以下示例说明如何报告连接的数据库的 DML 活动统计数据库。

```
--Define the oid variable from connected database by using \gset
=> SELECT oid,
          datname
     FROM pg_database
    WHERE datname=(select current_database()) \gset
=> SELECT *
     FROM aurora_stat_dml_activity(:oid);
select_count | select_latency_microsecs | insert_count | insert_latency_microsecs |
 update_count | update_latency_microsecs | delete_count | delete_latency_microsecs
--------------+--------------------------+--------------+--------------------------
+--------------+--------------------------+--------------+--------------------------
       178957 |                 66684115 |       171065 |                 28876649 |
    519538 |            1454579206167 |            1 |                    53027


-- Showing the same results with expanded display on
=> SELECT *
     FROM aurora_stat_dml_activity(:oid);
-[ RECORD 1 ]------------+-------------
select_count             | 178957
```

```
select_latency_microsecs | 66684115
insert_count             | 171065
insert_latency_microsecs | 28876649
update_count             | 519538
update_latency_microsecs | 1454579206167
delete_count             | 1
delete_latency_microsecs | 53027
```

以下示例显示了 Aurora PostgreSQL 集群中所有数据库的 DML 活动统计信息。该集群有两个数据库 postgres 和 mydb。逗号分隔的列表对应 select_count、select_latency_microsecs、insert_count、insert_latency_microsecs、up 和 delete_latency_microsecs 字段。

Aurora PostgreSQL 创建并使用名为 rdsadmin 的系统数据库来支持管理操作，例如备份、还原、运行状况检查、复制等。这些 DML 操作对您的 Aurora PostgreSQL 集群没有任何影响。

```
=> SELECT oid,
    datname,
    aurora_stat_dml_activity(oid)
    FROM pg_database;
oid  |    datname     |                   aurora_stat_dml_activity
-------+---------------
+-----------------------------------------------------------------
 14006 | template0      | (,,,,,,,)
 16384 | rdsadmin       | (2346623,1211703821,4297518,817184554,0,0,0,0)
     1 | template1      | (,,,,,,,)
 14007 | postgres       |
 (178961,66716329,171065,28876649,519538,1454579206167,1,53027)
 16401 | mydb           | (200246,64302436,200036,107101855,600000,83659417514,0,0)
```

以下示例显示了所有数据库的 DML 活动统计信息，这些统计信息按列进行组织，以提高可读性。

```
SELECT db.datname,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 1), '()') AS select_count,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 2), '()') AS select_latency_microsecs,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 3), '()') AS insert_count,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 4), '()') AS insert_latency_microsecs,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 5), '()') AS update_count,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 6), '()') AS update_latency_microsecs,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 7), '()') AS delete_count,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 8), '()') AS delete_latency_microsecs
FROM  (SELECT datname,
```

```
              aurora_stat_dml_activity(oid) AS asdmla
         FROM pg_database
      ) AS db;

    datname     | select_count | select_latency_microsecs | insert_count |
 insert_latency_microsecs | update_count | update_latency_microsecs | delete_count |
 delete_latency_microsecs
----------------+--------------+--------------------------+--------------
+--------------------------+--------------+--------------------------+--------------
+--------------------------
 template0      |              |                          |              |
             |              |                          |              |
 rdsadmin       | 4206523      | 2478812333               | 7009414      | 1338482258
             | 0            | 0                        | 0            | 0
 template1      |              |                          |              |
             |              |                          |              |
 fault_test     | 66           | 452099                   | 0            | 0
             | 0            | 0                        | 0            | 0
 db_access_test | 1            | 5982                     | 0            | 0
             | 0            | 0                        | 0            | 0
 postgres       | 42035        | 95179203                 | 5752         | 2678832898
             | 21157        | 441883182488             | 2            | 1520
 mydb           | 71           | 453514                   | 0            | 0
             | 1            | 190                      | 1            | 152
```

以下示例显示具有 OID 16401 的数据库每个 DML 操作的平均累积延迟时间（累积延迟除以计数）。

```
=> SELECT select_count,
          select_latency_microsecs,
          select_latency_microsecs/NULLIF(select_count,0) select_latency_per_exec,
          insert_count,
          insert_latency_microsecs,
          insert_latency_microsecs/NULLIF(insert_count,0) insert_latency_per_exec,
          update_count,
          update_latency_microsecs,
          update_latency_microsecs/NULLIF(update_count,0) update_latency_per_exec,
          delete_count,
          delete_latency_microsecs,
          delete_latency_microsecs/NULLIF(delete_count,0) delete_latency_per_exec
     FROM aurora_stat_dml_activity(16401);
-[ RECORD 1 ]------------+-------------
select_count             | 451312
select_latency_microsecs | 80205857
```

```
select_latency_per_exec  | 177
insert_count             | 451001
insert_latency_microsecs | 123667646
insert_latency_per_exec  | 274
update_count             | 1353067
update_latency_microsecs | 200900695615
update_latency_per_exec  | 148478
delete_count             | 12
delete_latency_microsecs | 448
delete_latency_per_exec  | 37
```

aurora_stat_get_db_commit_latency

获取 Aurora PostgreSQL 数据库的累积提交延迟时间（以微秒为单位）。按照为客户端提交提交请求到接收提交确认之间的时间测量提交延迟。

语法

```
aurora_stat_get_db_commit_latency(database_oid)
```

参数

database_oid

Aurora PostgreSQL 数据库的对象 ID（OID）。

返回类型

SETOF 记录

使用说明

Amazon CloudWatch 使用此函数来计算平均提交延迟。有关 Amazon CloudWatch 指标以及如何解决高提交延迟问题的更多信息，请参阅在 Amazon RDS 控制台中查看指标和使用 Amazon CloudWatch 指标更好地制定有关 Amazon RDS 的决策。

您可以使用 PostgreSQL 统计信息访问函数 `pg_stat_reset` 重置此统计信息。您可以使用 `pg_stat_get_db_stat_reset_time` 函数检查上次重置此统计信息的时间。有关 PostgreSQL 统计访问函数的更多信息，请参阅 PostgreSQL 文档中的统计收集器。

示例

以下示例获取 pg_database 集群中每个数据库的累积提交延迟。

```
=> SELECT oid,
    datname,
    aurora_stat_get_db_commit_latency(oid)
    FROM pg_database;

  oid  |    datname     | aurora_stat_get_db_commit_latency
-------+----------------+-----------------------------------
 14006 | template0      |                                 0
 16384 | rdsadmin       |                         654387789
     1 | template1      |                                 0
 16401 | mydb           |                            229556
 69768 | postgres       |                             22011
```

以下示例获取当前连接的数据库的累积提交延迟。在调用 aurora_stat_get_db_commit_latency 函数之前，该示例首先使用 \gset 为参数定义 oid 变量并从连接的数据库中设置其值。

```
--Get the oid value from the connected database before calling
 aurora_stat_get_db_commit_latency
=> SELECT oid
     FROM pg_database
    WHERE datname=(SELECT current_database()) \gset
=> SELECT *
     FROM aurora_stat_get_db_commit_latency(:oid);

 aurora_stat_get_db_commit_latency
-----------------------------------
                        1424279160
```

以下示例获取 pg_database 集群中 mydb 数据库的累积提交延迟。然后，它使用 pg_stat_reset 函数重置此统计信息并显示结果。最后，它使用 pg_stat_get_db_stat_reset_time 函数来检查上次重置此统计信息的时间。

```
=> SELECT oid,
    datname,
    aurora_stat_get_db_commit_latency(oid)
    FROM pg_database
    WHERE datname = 'mydb';
```

```
  oid  |  datname  | aurora_stat_get_db_commit_latency
-------+-----------+-----------------------------------
 16427 | mydb      |                           3320370


=> SELECT pg_stat_reset();
 pg_stat_reset
---------------


=> SELECT oid,
          datname,
          aurora_stat_get_db_commit_latency(oid)
     FROM pg_database
    WHERE datname = 'mydb';
  oid  |  datname  | aurora_stat_get_db_commit_latency
-------+-----------+-----------------------------------
 16427 | mydb      |                                 6


=> SELECT *
     FROM pg_stat_get_db_stat_reset_time(16427);

 pg_stat_get_db_stat_reset_time
--------------------------------
 2021-04-29 21:36:15.707399+00
```

aurora_stat_logical_wal_cache

显示每个插槽的逻辑预写日志（WAL）缓存使用情况。

语法

```
SELECT * FROM aurora_stat_logical_wal_cache()
```

参数

无

返回类型

包含以下列的 SETOF 记录：

- name – 复制插槽的名称。
- active_pid – walsender 进程的 ID。
- cache_hit – 自上次重置以来的 wal 缓存命中总数。
- cache_miss – 自上次重置以来的 wal 缓存未命中总数。
- blks_read – wal 缓存读取请求的总数。
- hit_rate – WAL 缓存命中率（cache_hit/blks_read）。
- last_reset_timestamp – 上次重置计数器的时间。

使用说明

此函数可用于以下 Aurora PostgreSQL 版本。

- 15.2 及所有更高版本
- 14.7 及更高版本
- 13.8 及更高版本
- 12.12 及更高版本
- 11.17 及更高版本

示例

以下示例显示了两个仅具有一个活动的 aurora_stat_logical_wal_cache 函数的复制插槽。

```
=> SELECT *
     FROM aurora_stat_logical_wal_cache();
    name    | active_pid | cache_hit | cache_miss | blks_read | hit_rate |
 last_reset_timestamp
------------+------------+-----------+------------+-----------+----------
+-------------------------------
 test_slot1 |      79183 |        24 |          0 |        24 | 100.00%  | 2022-08-05
 17:39:56.830635+00
 test_slot2 |            |         1 |          0 |         1 | 100.00%  | 2022-08-05
 17:34:04.036795+00
(2 rows)
```

aurora_stat_memctx_usage

报告每个 PostgreSQL 进程的内存上下文使用情况。

语法

```
aurora_stat_memctx_usage()
```

参数

无

返回类型

包含以下列的 SETOF 记录：

- pid – 进程的 ID。
- name – 内存上下文的名称。
- allocated – 内存上下文从底层内存子系统获得的字节数。
- used – 提交给内存上下文客户端的字节数。
- instances – 当前存在的此类上下文的计数。

使用说明

此函数显示每个 PostgreSQL 进程的内存上下文使用情况。有些进程标为 anonymous。这些进程不会被公开，因为其中包含受限制的关键字。

从以下 Aurora PostgreSQL 版本开始提供此函数：

- 15.3 及更高的 15 版本
- 14.8 及更高的 14 版本
- 13.11 及更高的 13 版本
- 12.15 及更高的 12 版本
- 11.20 及更高的 11 版本

示例

以下示例显示了调用 aurora_stat_memctx_usage 函数的结果。

```
=> SELECT *
     FROM aurora_stat_memctx_usage();

    pid| name                            | allocated |   used   | instances
```

```
-------+---------------------------------+-----------+---------+-----------
123864 | Miscellaneous                   |     19520 |   15064 |         3
123864 | Aurora File Context             |      8192 |     616 |         1
123864 | Aurora WAL Context              |      8192 |     296 |         1
123864 | CacheMemoryContext              |    524288 |  422600 |         1
123864 | Catalog tuple context           |     16384 |   13736 |         1
123864 | ExecutorState                   |     32832 |   28304 |         1
123864 | ExprContext                     |      8192 |    1720 |         1
123864 | GWAL record construction        |      1024 |     832 |         1
123864 | MdSmgr                          |      8192 |     296 |         1
123864 | MessageContext                  |    532480 |  353832 |         1
123864 | PortalHeapMemory                |      1024 |     488 |         1
123864 | PortalMemory                    |      8192 |     576 |         1
123864 | printtup                        |      8192 |     296 |         1
123864 | RelCache hash table entries     |      8192 |    8152 |         1
123864 | RowDescriptionContext           |      8192 |    1344 |         1
123864 | smgr relation context           |      8192 |     296 |         1
123864 | Table function arguments        |      8192 |     352 |         1
123864 | TopTransactionContext           |      8192 |     632 |         1
123864 | TransactionAbortContext         |     32768 |     296 |         1
123864 | WAL record construction         |     50216 |   43904 |         1
123864 | hash table                      |     65536 |   52744 |         6
123864 | Relation metadata               |    191488 |  124240 |        87
104992 | Miscellaneous                   |      9280 |    7728 |         3
104992 | Aurora File Context             |      8192 |     376 |         1
104992 | Aurora WAL Context              |      8192 |     296 |         1
104992 ||Autovacuum Launcher            |      8192 |     296 |         1
104992 | Autovacuum database list        |     16384 |     744 |         2
104992 | CacheMemoryContext              |    262144 |  140288 |         1
104992 | Catalog tuple context           |      8192 |     296 |         1
104992 | GWAL record construction        |      1024 |     832 |         1
104992 | MdSmgr                          |      8192 |     296 |         1
104992 | PortalMemory                    |      8192 |     296 |         1
104992 | RelCache hash table entries     |      8192 |     296 |         1
104992 | smgr relation context           |      8192 |     296 |         1
104992 | Autovacuum start worker (tmp)   |      8192 |     296 |         1
104992 | TopTransactionContext           |     16384 |     592 |         2
104992 | TransactionAbortContext         |     32768 |     296 |         1
104992 | WAL record construction         |     50216 |   43904 |         1
104992 | hash table                      |     49152 |   34024 |         4
(39 rows)
```

一些受限制的关键字将被隐藏，输出将如下所示：

```
postgres=>SELECT *
     FROM aurora_stat_memctx_usage();

    pid| name                              | allocated |   used   | instances
-------+-----------------------------------+-----------+----------+-----------
  5482 | anonymous                         |      8192 |     456 |         1
  5482 | anonymous                         |      8192 |     296 |         1
```

aurora_stat_optimized_reads_cache

此函数显示分层缓存统计信息。

语法

```
aurora_stat_optimized_reads_cache()
```

参数

无

返回类型

包含以下列的 SETOF 记录：

- `total_size` – 优化型读取缓存总大小。
- `used_size` – 优化型读取缓存中使用的页面大小。

使用说明

此函数可用于以下 Aurora PostgreSQL 版本：

- 15.4 及更高的 15 版本
- 14.9 及更高的 14 版本

示例

下面的示例显示了 r6gd.8xlarge 实例上的输出：

```
=> select pg_size_pretty(total_size) as total_size, pg_size_pretty(used_size)
```

```
                as used_size from aurora_stat_optimized_reads_cache();
total_size | used_size
-----------+-----------
1054 GB    | 975 GB
```

aurora_stat_plans

为每个跟踪的执行计划返回一行。

语法

```
aurora_stat_plans(
    showtext
)
```

参数

- showtext：显示查询和计划文本。有效值为 NULL、true 或 false。True 将显示查询和计划文本。

返回类型

为每个跟踪的计划返回一行，其中包含来自 aurora_stat_statements 的所有列和以下附加列。

- planid：计划标识符
- explain_plan：解释计划文本
- plan_type：
  - no plan：未捕获到任何计划
  - estimate：使用估计成本捕获的计划
  - actual：使用 EXPLAIN ANALYZE 捕获的计划
- plan_captured_time：上次捕获计划的时间

使用说明

必须启用 aurora_compute_plan_id 并且 pg_stat_statements 必须处于 shared_preload_libraries 状态才能跟踪计划。

可用计划的数量由在 pg_stat_statements.max 参数中设置的值控制。启用 aurora_compute_plan_id 后，您可以在 aurora_stat_plans 中跟踪不超过此指定值的计划。

Aurora PostgreSQL 版本 14.10、15.5 以及所有其它更高版本都提供了此函数。

示例

在下面的示例中，将捕获用于查询标识符 5471422286312252535 的两个计划，并通过 planid 跟踪语句统计数据。

```
db1=# select calls, total_exec_time, planid, plan_captured_time, explain_plan
db1-# from aurora_stat_plans(true)
db1-# where queryid = '-5471422286312252535'

calls    |  total_exec_time   |   planid    |      plan_captured_time       |
                 explain_plan
---------+--------------------+-------------+-------------------------------
+-----------------------------------------------------------------
 1532632 |  3209846.097107853 |  1602979607 | 2023-10-31 03:27:16.925497+00 | Update on
 pgbench_branches                                       +
         |                    |             |                               |   ->
 Bitmap Heap Scan on pgbench_branches                      +
         |                    |             |                               |
 Recheck Cond: (bid = 76)                                  +
         |                    |             |                               |         -
>  Bitmap Index Scan on pgbench_branches_pkey          +
         |                    |             |                               |
     Index Cond: (bid = 76)
   61365 | 124078.18012200127 | -2054628807 | 2023-10-31 03:20:09.85429+00  | Update on
 pgbench_branches                                       +
         |                    |             |                               |   ->
 Index Scan using pgbench_branches_pkey on pgbench_branches+
         |                    |             |                               |
 Index Cond: (bid = 17)
```

aurora_stat_reset_wal_cache

重置逻辑 wal 缓存的计数器。

语法

重置特定插槽

```
SELECT * FROM aurora_stat_reset_wal_cache('slot_name')
```

重置所有插槽

```
SELECT * FROM aurora_stat_reset_wal_cache(NULL)
```

参数

NULL 或 slot_name

返回类型

状态消息，文本字符串

- 重置逻辑 wal 缓存计数器 – 成功消息。函数成功时将返回此文本。
- 未找到复制插槽。请重试。– 失败消息。当函数不成功时，将返回此文本。

使用说明

此函数可用于以下版本。

- Aurora PostgreSQL 14.5 及更高版本
- Aurora PostgreSQL 版本 13.8 及更高版本
- Aurora PostgreSQL 版本 12.12 及更高版本
- Aurora PostgreSQL 版本 11.17 及更高版本

示例

以下示例使用 aurora_stat_reset_wal_cache 函数重置名为 test_results 的插槽，然后尝试重置不存在的插槽。

```
=> SELECT *
     FROM aurora_stat_reset_wal_cache('test_slot');
aurora_stat_reset_wal_cache
-------------------------------------
 Reset the logical wal cache counter.
(1 row)
```

```
=> SELECT *
     FROM aurora_stat_reset_wal_cache('slot-not-exist');
aurora_stat_reset_wal_cache
-----------------------------------------------
 Replication slot not found. Please try again.
(1 row)
```

aurora_stat_resource_usage

报告所有 Aurora PostgreSQL 后端进程的实时资源利用率，其中包括后端资源指标和 CPU 使用率。

语法

```
aurora_stat_resource_usage()
```

参数

无

返回类型

带有以下各列的 SETOF 记录：

- pid：进程标识符
- allocated_memory：进程分配的内存总量（以字节为单位）
- used_memory：进程实际使用的内存（以字节为单位）
- cpu_usage_percent：进程的 CPU 使用率百分比

使用说明

此函数显示每个 Aurora PostgreSQL 后端进程的后端资源使用情况。

从以下 Aurora PostgreSQL 版本开始提供此函数：

- Aurora PostgreSQL 17.5 及更高的 17 版本
- Aurora PostgreSQL 16.9 及更高的 16 版本
- Aurora PostgreSQL 15.13 及更高的 15 版本
- Aurora PostgreSQL 14.18 及更高的 14 版本

- Aurora PostgreSQL 13.21 及更高的 13 版本

示例

下面的示例显示 aurora_stat_resource_usage 函数的输出。

```
=> select * from aurora_stat_resource_usage();
 pid  | allocated_memory | used_memory |   cpu_usage_percent
------+------------------+-------------+-----------------------
  666 |          1074032 |      333544 |   0.00729274882897963
  667 |           787312 |      287360 | 0.0029263928146372746
  668 |          3076776 |     1563488 |  0.006013116835953961
  684 |           803744 |      307480 |  0.002226855426881142
 2401 |          1232992 |      943144 |                     0
  647 |             8000 |         944 |   0.48853387812429855
  659 |           319344 |      243000 | 0.0004135602076683591
  663 |           262000 |      185736 |  0.008181301476644002
  664 |             9024 |        1216 |   0.10992313082653653
(9 rows)
```

aurora_stat_statements

显示所有 pg_stat_statements 列并在最后添加更多列。

语法

```
aurora_stat_statements(showtext boolean)
```

参数

showtext boolean

返回类型

包含所有 pg_stat_statements 列和以下附加列的 SETOF 记录。有关 pg_stat_statements 列的更多信息，请参阅 pg_stat_statements。

您可以使用 pg_stat_statements_reset() 重置此函数的统计信息。

- storage_blks_read – 从该语句的 aurora 存储中读取的共享块总数。
- orcache_blks_hit – 此语句中优化型读取缓存命中总数。

- storage_blk_read_time – 如果已启用 track_io_timing，它将跟踪语句从 aurora 存储中读取共享块所花费的总时间（以毫秒为单位），否则该值为零。有关更多信息，请参阅 track_io_timing。
- local_blk_read_time – 如果已启用 track_io_timing，它将跟踪语句读取本地块所花费的总时间（以毫秒为单位），否则该值为零。有关更多信息，请参阅 track_io_timing。
- orcache_blk_read_time – 如果已启用 track_io_timing，它将跟踪语句从优化型读取缓存中读取共享块所花费的总时间（以毫秒为单位），否则该值为零。有关更多信息，请参阅 track_io_timing。
- total_plan_peakmem – 对此语句的所有调用在计划阶段的峰值内存值的总和。要查看语句在计划期间的平均峰值内存，请将该值除以调用数。
- min_plan_peakmem – 在计划期间，在对此语句的所有调用中看到的最小峰值内存值。
- max_plan_peakmem – 在计划期间，在对此语句的所有调用中看到的最大峰值内存值。
- total_exec_peakmem – 对此语句的所有调用在执行阶段的峰值内存值的总和。要查看语句在执行期间的平均峰值内存，请将该值除以调用数。
- min_exec_peakmem – 在对此语句的所有调用的执行期间，出现的最小峰值内存值，以字节为单位。
- max_exec_peakmem – 在对此语句的所有调用的执行期间，出现的最大峰值内存值，以字节为单位。

> **Note**
>
> 仅当设置 pg_stat_statements.track_planning 为开启时，total_plan_peakmen、min_plan_peakmem 和 max_plan_peakmem 才会受到监控。

使用说明

要使用 aurora_stat_statements() 函数，必须在 shared_preload_libraries 参数中包含 pg_stat_statements 扩展。

此函数可用于以下 Aurora PostgreSQL 版本：

- 15.4 及更高的 15 版本
- 14.9 及更高的 14 版本

以下版本中提供了显示峰值内存的列：

- 16.3 及更高版本
- 15.7 及更高版本
- 14.12 及更高版本

示例

以下示例显示了它如何携带所有 pg_stat_statements 列并在最后附加 11 个新列：

```
=> select * from aurora_stat_statements(true) where query like 'with window_max%';
-[ RECORD 1 ]----------
+-----------------------------------------------------------------------------------------------
userid                 | 16409
dbid                   | 5
toplevel               | t
queryid                | -8347523682669847482
query                  | with window_max as (select custid, max(scratch) over (order by
 scratch rows between $1 preceding
and $2 following) wmax from ts) select sum(wmax), max(custid) from window_max
plans                  | 0
total_plan_time        | 0
min_plan_time          | 0
max_plan_time          | 0
mean_plan_time         | 0
stddev_plan_time       | 0
calls                  | 4
total_exec_time        | 254.105121
min_exec_time          | 57.503164000000005
max_exec_time          | 68.687418
mean_exec_time         | 63.52628025
stddev_exec_time       | 5.150765359979643
rows                   | 4
shared_blks_hit        | 200192
shared_blks_read       | 0
shared_blks_dirtied    | 0
shared_blks_written    | 0
local_blks_hit         | 0
local_blks_read        | 0
local_blks_dirtied     | 0
local_blks_written     | 0
temp_blks_read         | 0
```

```
temp_blks_written      | 0
blk_read_time          | 0
blk_write_time         | 0
temp_blk_read_time     | 0
temp_blk_write_time    | 0
wal_records            | 0
wal_fpi                | 0
wal_bytes              | 0
jit_functions          | 0
jit_generation_time    | 0
jit_inlining_count     | 0
jit_inlining_time      | 0
jit_optimization_count | 0
jit_optimization_time  | 0
jit_emission_count     | 0
jit_emission_time      | 0
storage_blks_read      | 0
orcache_blks_hit       | 0
storage_blk_read_time  | 0
local_blk_read_time    | 0
orcache_blk_read_time  | 0
total_plan_peakmem     | 0
min_plan_peakmem       | 0
max_plan_peakmem       | 0
total_exec_peakmem     | 6356224
min_exec_peakmem       | 1589056
max_exec_peakmem       | 1589056
```

aurora_stat_system_waits

报告 Aurora PostgreSQL 数据库实例的等待事件信息。

语法

```
aurora_stat_system_waits()
```

参数

无

返回类型

SETOF 记录

使用说明

此函数返回由您当前连接的数据库实例生成的每个等待事件的累积等待次数和累积等待时间。

返回的记录集中包含以下字段：

- type_id – 等待事件类型的 ID。
- event_id – 等待事件的 ID。
- waits – 等待事件发生的次数。
- wait_time – 等待此事件所花费的总时间（以微秒为单位）。

当数据库实例重新启动时，此函数返回的统计信息将重置。

示例

以下示例显示了调用 aurora_stat_system_waits 函数的结果。

```
=> SELECT *
    FROM aurora_stat_system_waits();
 type_id | event_id  |   waits   |  wait_time
---------+-----------+-----------+--------------
       1 |  16777219 |        11 |        12864
       1 |  16777220 |       501 |       174473
       1 |  16777270 |     53171 |     23641847
       1 |  16777271 |        23 |       319668
       1 |  16777274 |        60 |        12759
.
.
.
      10 | 167772231 |    204596 |    790945212
      10 | 167772232 |         2 |        47729
      10 | 167772234 |         1 |          888
      10 | 167772235 |         2 |           64
```

以下示例显示了如何将此函数与 aurora_stat_wait_event 和 aurora_stat_wait_type 一起使用并产生更具可读性的结果。

```
=> SELECT type_name,
            event_name,
            waits,
            wait_time
```

```
        FROM aurora_stat_system_waits()
NATURAL JOIN aurora_stat_wait_event()
NATURAL JOIN aurora_stat_wait_type();

 type_name |       event_name       |  waits   |  wait_time
-----------+------------------------+----------+--------------
 LWLock    | XidGenLock             |       11 |        12864
 LWLock    | ProcArrayLock          |      501 |       174473
 LWLock    | buffer_content         |    53171 |     23641847
 LWLock    | rdsutils               |        2 |        12764
 Lock      | tuple                  |    75686 |   2033956052
 Lock      | transactionid          |  1765147 |  47267583409
 Activity  | AutoVacuumMain         |   136868 |  56305604538
 Activity  | BgWriterHibernate      |     7486 |  55266949471
 Activity  | BgWriterMain           |     7487 |   1508909964
.
.
.
 IO        | SLRURead               |        3 |        11756
 IO        | WALWrite               | 52544463 |    388850428
 IO        | XactSync               |   187073 |    597041642
 IO        | ClogRead               |        2 |        47729
 IO        | OutboundCtrlRead       |        1 |          888
 IO        | OutboundCtrlWrite      |        2 |           64
```

aurora_stat_wait_event

列出 Aurora PostgreSQL 的所有支持等待事件。有关 Aurora PostgreSQL 等待事件的信息，请参阅 Amazon Aurora PostgreSQL 等待事件。

语法

```
aurora_stat_wait_event()
```

参数

无

返回类型

包含以下列的 SETOF 记录：

- type_id – 等待事件类型的 ID。

- event_id – 等待事件的 ID。
- type_name – 等待类型名称
- event_name – 等待事件名称

使用说明

要查看具有事件类型而不是 ID 的事件名称，请将此函数与其他函数（如 aurora_stat_wait_type 和 aurora_stat_system_waits）一起使用。此函数返回的等待事件名称与 aurora_wait_report 函数返回的事件名称相同。

示例

以下示例显示了调用 aurora_stat_wait_event 函数的结果。

```
=>  SELECT *
    FROM aurora_stat_wait_event();

 type_id | event_id  |                event_name
---------+-----------+-------------------------------------------
       1 |  16777216 | <unassigned:0>
       1 |  16777217 | ShmemIndexLock
       1 |  16777218 | OidGenLock
       1 |  16777219 | XidGenLock
.
.
.
       9 | 150994945 | PgSleep
       9 | 150994946 | RecoveryApplyDelay
      10 | 167772160 | BufFileRead
      10 | 167772161 | BufFileWrite
      10 | 167772162 | ControlFileRead
.
.
.
      10 | 167772226 | WALInitWrite
      10 | 167772227 | WALRead
      10 | 167772228 | WALSync
      10 | 167772229 | WALSyncMethodAssign
      10 | 167772230 | WALWrite
      10 | 167772231 | XactSync
.
.
```

```
.
      11 | 184549377 | LsnAllocate
```

以下示例连接 aurora_stat_wait_type 和 aurora_stat_wait_event 来返回类型名称和事件名称以提高可读性。

```
=> SELECT *
    FROM aurora_stat_wait_type() t
    JOIN aurora_stat_wait_event() e
      ON t.type_id = e.type_id;

 type_id | type_name | type_id | event_id  |                event_name
---------+-----------+---------+-----------+-------------------------------------------
       1 | LWLock    |       1 |  16777216 | <unassigned:0>
       1 | LWLock    |       1 |  16777217 | ShmemIndexLock
       1 | LWLock    |       1 |  16777218 | OidGenLock
       1 | LWLock    |       1 |  16777219 | XidGenLock
       1 | LWLock    |       1 |  16777220 | ProcArrayLock
.
.
.
       3 | Lock      |       3 |  50331648 | relation
       3 | Lock      |       3 |  50331649 | extend
       3 | Lock      |       3 |  50331650 | page
       3 | Lock      |       3 |  50331651 | tuple
.
.
.
      10 | IO        |      10 | 167772214 | TimelineHistorySync
      10 | IO        |      10 | 167772215 | TimelineHistoryWrite
      10 | IO        |      10 | 167772216 | TwophaseFileRead
      10 | IO        |      10 | 167772217 | TwophaseFileSync
.
.
.
      11 | LSN       |      11 | 184549376 | LsnDurable
```

aurora_stat_wait_type

列出 Aurora PostgreSQL 的所有支持等待类型。

语法

```
aurora_stat_wait_type()
```

参数

无

返回类型

包含以下列的 SETOF 记录：

- type_id – 等待事件类型的 ID。
- type_name – 等待类型名称。

使用说明

要查看具有等待事件类型而不是 ID 的等待事件名称，请将此函数与其他函数（如 aurora_stat_wait_event 和 aurora_stat_system_waits）一起使用。此函数返回的等待类型名称与 aurora_wait_report 函数返回的事件名称相同。

示例

以下示例显示了调用 aurora_stat_wait_type 函数的结果。

```
=> SELECT *
     FROM aurora_stat_wait_type();
 type_id | type_name
---------+-----------
       1 | LWLock
       3 | Lock
       4 | BufferPin
       5 | Activity
       6 | Client
       7 | Extension
       8 | IPC
       9 | Timeout
      10 | IO
      11 | LSN
```

aurora_version

返回 Amazon Aurora PostgreSQL 兼容版本号的字符串值。

语法

```
aurora_version()
```

参数

无

返回类型

CHAR 或 VARCHAR 字符串

使用说明

此函数显示 Amazon Aurora PostgreSQL 兼容版数据库引擎的版本。版本号以字符串形式返回，格式为 *major*.*minor*.*patch*。有关 Aurora PostgreSQL 版本号的更多信息，请参阅Aurora 版本号。

您可以通过设置 Aurora PostgreSQL 数据库集群的维护时段来选择何时应用次要版本升级。要了解如何操作，请参阅维护 Amazon Aurora 数据库集群。

从发布的 Aurora PostgreSQL 版本 13.3、12.8、11.13、10.18 开始至这以后发布的所有更高版本，Aurora 版本号都与 PostgreSQL 版本号保持一致。有关所有 Aurora PostgreSQL 版本的更多信息，请参阅《Aurora PostgreSQL 发布说明》中的 Amazon Aurora PostgreSQL 更新。

示例

以下示例显示在运行 PostgreSQL 12.7、Aurora PostgreSQL 版本 4.2 的 Aurora PostgreSQL 数据库集群上调用 `aurora_version` 函数，然后在运行 Aurora PostgreSQL 版本 13.3 的集群上运行相同函数的结果。

```
=> SELECT * FROM aurora_version();
aurora_version
----------------
 4.2.2
SELECT * FROM aurora_version();
aurora_version
----------------
 13.3.0
```

此示例说明如何将函数与各种选项结合使用，以获取有关 Aurora PostgreSQL 版本的更多详细信息。此示例具有与 PostgreSQL 版本号不同的 Aurora 版本号。

```
=> SHOW SERVER_VERSION;
 server_version
----------------
 12.7
(1 row)

=> SELECT * FROM aurora_version();
aurora_version
----------------
 4.2.2
(1 row)

=> SELECT current_setting('server_version') AS "PostgreSQL Compatiblility";
PostgreSQL Compatiblility
--------------------------
 12.7
(1 row)

=> SELECT version() AS "PostgreSQL Compatiblility Full String";
PostgreSQL Compatiblility Full String
-------------------------------------------------------------------------------------------------
 PostgreSQL 12.7 on aarch64-unknown-linux-gnu, compiled by aarch64-unknown-linux-gnu-
gcc (GCC) 7.4.0, 64-bit
(1 row)

=> SELECT 'Aurora: '
    || aurora_version()
    || ' Compatible with PostgreSQL: '
    || current_setting('server_version') AS "Instance Version";
Instance Version
-----------------------------------------------
 Aurora: 4.2.2 Compatible with PostgreSQL: 12.7
(1 row)
```

下一个示例使用具有与上一个示例相同选项的函数。此示例不具有与 PostgreSQL 版本号不同的 Aurora 版本号。

```
=> SHOW SERVER_VERSION;
server_version
----------------
 13.3
```

```
=> SELECT * FROM aurora_version();
aurora_version
---------------
 13.3.0
=> SELECT current_setting('server_version') AS "PostgreSQL Compatiblility";
PostgreSQL Compatiblility
--------------------------
 13.3

=> SELECT version() AS "PostgreSQL Compatiblility Full String";
PostgreSQL Compatiblility Full String
-------------------------------------------------------------------------------------------------------
 PostgreSQL 13.3 on x86_64-pc-linux-gnu, compiled by x86_64-pc-linux-gnu-gcc (GCC)
 7.4.0, 64-bit
=> SELECT 'Aurora: '
    || aurora_version()
    || ' Compatible with PostgreSQL: '
    || current_setting('server_version') AS "Instance Version";
Instance Version
-----------------------------------------------------
 Aurora: 13.3.0 Compatible with PostgreSQL: 13.3
```

aurora_volume_logical_start_lsn

返回一个日志序列号（LSN），此序列号用于标识 Aurora 集群卷的逻辑预写日志（WAL）流中记录的开头。

语法

```
aurora_volume_logical_start_lsn()
```

参数

无

返回类型

pg_lsn

使用说明

此函数标识给定 Aurora 集群卷的逻辑 WAL 流中记录的开头。在使用逻辑复制和 Aurora 快速克隆执行主要版本升级时，您可以使用此函数来确定用于拍摄快照或制作数据库克隆的 LSN。然后，您可以使用逻辑复制来持续流式传输 LSN 之后记录的较新数据，并将更改从发布者同步到订阅用户。

有关使用逻辑复制进行主要版本升级的更多信息，请参阅使用逻辑复制对 Aurora PostgreSQL 执行主要版本升级。

此函数可用于以下 Aurora PostgreSQL 版本：

- 15.2 及更高的 15 版本
- 14.3 及更高的 14 版本
- 13.6 及更高的 13 版本
- 12.10 及更高的 12 版本
- 11.15 及更高的 11 版本
- 10.20 及更高的 10 版本

示例

您可以使用以下查询获取日志序列号（LSN）：

```
postgres=> SELECT aurora_volume_logical_start_lsn();

aurora_volume_logical_start_lsn
---------------
0/402E2F0
(1 row)
```

aurora_wait_report

此函数显示一段时间内的等待事件活动。

语法

```
aurora_wait_report([time])
```

参数

时间（可选）

以秒为单位的时间。默认为 10 秒。

返回类型

包含以下列的 SETOF 记录：

- type_name – 等待类型名称
- event_name – 等待事件名称
- 等待 – 等待次数
- wait_time – 以毫秒为单位的等待时间
- ms_per_wait – 按等待次数计算的平均毫秒数
- waits_per_xact – 按一个事务数计算的平均等待次数
- ms_per_xact – 按事务数计算的平均毫秒数

使用说明

此函数从与 PostgreSQL 9.6.6 及更高版本兼容的 Aurora PostgreSQL 版本 1.1 开始提供。

要使用此函数，您首先需要创建 Aurora PostgreSQL `aurora_stat_utils` 扩展，如下所示：

```
=> CREATE extension aurora_stat_utils;
CREATE EXTENSION
```

有关可用的 urora PostgreSQL 扩展版本的更多信息，请参阅《Aurora PostgreSQL 发布说明》中的适用于 Amazon Aurora PostgreSQL 的扩展版本。

此函数通过将 aurora_stat_system_waits() 函数和 pg_stat_database PostgreSQL 统计信息视图中的两个统计信息数据快照进行比较来计算实例级等待事件。

有关 `aurora_stat_system_waits()` 和 `pg_stat_database` 的更多信息，请参阅 PostgreSQL 文档中的统计信息收集器。

运行时，此函数会拍摄初始快照，等待指定的秒数，然后再拍摄第二个快照。此函数会比较两个快照并返回差异。此差异代表实例在该时间间隔内的活动。

在写入器实例上，此函数还显示已提交的事务数和 TPS（每秒事务数）。此函数返回实例级别的信息，包括实例上的所有数据库。

示例

此示例说明如何创建 aurora_stat_utils 扩展，以便能够使用 aurora_wait_report 函数。

```
=> CREATE extension aurora_stat_utils;
CREATE EXTENSION
```

此示例说明如何检查 10 秒的等待报告。

```
=> SELECT *
     FROM aurora_wait_report();
NOTICE:  committed 34 transactions in 10 seconds (tps 3)
 type_name |    event_name     | waits | wait_time | ms_per_wait | waits_per_xact |
 ms_per_xact
-----------+-------------------+-------+-----------+-------------+----------------
+-------------
 Client    | ClientRead        |    26 |  30003.00 |    1153.961 |           0.76 |
 882.441
 Activity  | WalWriterMain     |    50 |  10051.32 |     201.026 |           1.47 |
 295.627
 Timeout   | PgSleep           |     1 |  10049.52 |   10049.516 |           0.03 |
 295.574
 Activity  | BgWriterHibernate |     1 |  10048.15 |   10048.153 |           0.03 |
 295.534
 Activity  | AutoVacuumMain    |    18 |   9941.66 |     552.314 |           0.53 |
 292.402
 Activity  | BgWriterMain      |     1 |    201.09 |     201.085 |           0.03 |
   5.914
 IO        | XactSync          |    15 |     25.34 |       1.690 |           0.44 |
   0.745
 IO        | RelationMapRead   |    12 |      0.54 |       0.045 |           0.35 |
   0.016
 IO        | WALWrite          |    84 |      0.21 |       0.002 |           2.47 |
   0.006
 IO        | DataFileExtend    |     1 |      0.02 |       0.018 |           0.03 |
   0.001
```

此示例说明如何检查 60 秒的等待报告。

```
=> SELECT *
```

```
     FROM aurora_wait_report(60);
NOTICE:  committed 1544 transactions in 60 seconds (tps 25)
 type_name |       event_name       |  waits  | wait_time | ms_per_wait |
 waits_per_xact | ms_per_xact
-----------+------------------------+---------+-----------+-------------
+----------------+-------------
 Lock      | transactionid          |    6422 | 477000.53 |      74.276 |
 4.16 |     308.938
 Client    | ClientRead             |    8265 | 270752.99 |      32.759 |
 5.35 |     175.358
 Activity  | CheckpointerMain       |       1 |  60100.25 |   60100.246 |
 0.00 |      38.925
 Timeout   | PgSleep                |       1 |  60098.49 |   60098.493 |
 0.00 |      38.924
 Activity  | WalWriterMain          |     296 |  60010.99 |     202.740 |
 0.19 |      38.867
 Activity  | AutoVacuumMain         |     107 |  59827.84 |     559.139 |
 0.07 |      38.749
 Activity  | BgWriterMain           |     290 |  58821.83 |     202.834 |
 0.19 |      38.097
 IO        | XactSync               |    1295 |  55220.13 |      42.641 |
 0.84 |      35.764
 IO        | WALWrite               | 6602259 |  47810.94 |       0.007 |
 4276.07 |      30.966
 Lock      | tuple                  |     473 |  29880.67 |      63.173 |
 0.31 |      19.353
 LWLock    | buffer_mapping         |     142 |   3540.13 |      24.930 |
 0.09 |       2.293
 Activity  | BgWriterHibernate      |     290 |   1124.15 |       3.876 |
 0.19 |       0.728
 IO        | BufFileRead            |    7615 |    618.45 |       0.081 |
 4.93 |       0.401
 LWLock    | buffer_content         |      73 |    345.93 |       4.739 |
 0.05 |       0.224
 LWLock    | lock_manager           |      62 |    191.44 |       3.088 |
 0.04 |       0.124
 IO        | RelationMapRead        |      72 |      5.16 |       0.072 |
 0.05 |       0.003
 LWLock    | ProcArrayLock          |       1 |      2.01 |       2.008 |
 0.00 |       0.001
 IO        | ControlFileWriteUpdate |       2 |      0.03 |       0.013 |
 0.00 |       0.000
 IO        | DataFileExtend         |       1 |      0.02 |       0.018 |
 0.00 |       0.000
```

```
IO        | ControlFileSyncUpdate  |       1 |      0.00 |       0.000 |
0.00 |       0.000
```

Amazon Aurora PostgreSQL 参数

您可以使用数据库参数组中的参数按照与管理 Amazon RDS 数据库实例相同的方法管理 Amazon Aurora 数据库集群。但是，Amazon Aurora 与 Amazon RDS 的不同之处在于，Aurora 数据库集群有多个数据库实例。您用于管理 Amazon Aurora 数据库集群的一些参数适用于整个集群，而另一些参数仅适用于数据库集群中的给定数据库实例，如下所示：

- DB cluster parameter group（数据库集群参数组）：数据库集群参数组包含应用于整个 Aurora 数据库集群的引擎配置参数集。例如，集群缓存管理是由 `apg_ccm_enabled` 参数控制的一项 Aurora 数据库集群特征，该参数是数据库集群参数组的一部分。数据库集群参数组还包含组成集群的数据库实例的数据库参数组默认设置。
- DB parameter group（数据库参数组）：数据库参数组是应用于相应引擎类型的特定数据库实例的引擎配置值集。PostgreSQL 数据库引擎的数据库参数组由 RDS for PostgreSQL 数据库实例和 Aurora PostgreSQL 数据库集群使用。这些配置设置适用于在 Aurora 集群内的数据库实例之间可能不同的属性，如内存缓冲区的大小。

您可以在数据库集群参数组中管理集群级参数。您可以在数据库参数组中管理实例级参数。您可以使用 Amazon RDS 控制台、AWS CLI 或 Amazon RDS API 管理参数。可以使用单独的命令管理集群级参数和实例级参数。

- 要管理数据库集群参数组中的集群级参数，可以使用 modify-db-cluster-parameter-group AWS CLI 命令。
- 要为数据库集群中数据库实例管理数据库参数组中的实例级参数，可以使用 modify-db-parameter-group AWS CLI 命令。

要了解有关 AWS CLI 的更多信息，请参阅《AWS Command Line Interface 用户指南》中的使用 AWS CLI。

有关参数组的更多信息，请参阅 Amazon Aurora 的参数组。

查看 Aurora PostgreSQL 数据库集群和数据库参数

您可以在 AWS Management Console 中查看 RDS for PostgreSQL 数据库实例和 Aurora PostgreSQL 数据库集群的所有可用默认参数组。各 AWS 区域的所有数据库引擎和数据库集群类型及版本的默认参数组均已列出。同时，所有自定义参数组也已列出。

您还可以使用 AWS CLI 或 Amazon RDS API 列出数据库集群参数组和数据库参数组中包含的参数，无需在 AWS Management Console 中查看。例如，要列出数据库集群参数组中的参数，可以使用 describe-db-cluster-parameters AWS CLI 命令，如下所示：

```
aws rds describe-db-cluster-parameters --db-cluster-parameter-group-name
 default.aurora-postgresql12
```

该命令将返回每个参数的详细 JSON 描述。为了减少返回的信息量，您可以使用 --query 选项指定所需内容。例如，可以获取 Aurora PostgreSQL 12 数据库集群默认参数组的参数名称、描述和允许值，如下所示：

对于 Linux、macOS 或 Unix：

```
aws rds describe-db-cluster-parameters --db-cluster-parameter-group-name
 default.aurora-postgresql12 \
    --query 'Parameters[].
[{ParameterName:ParameterName,Description:Description,ApplyType:ApplyType,AllowedValues:Allowed
```

对于 Windows：

```
aws rds describe-db-cluster-parameters --db-cluster-parameter-group-name
 default.aurora-postgresql12 ^
    --query "Parameters[].
[{ParameterName:ParameterName,Description:Description,ApplyType:ApplyType,AllowedValues:Allowed
```

Aurora 数据库集群参数组包括数据库实例参数组和给定 Aurora 数据库引擎的默认值。您可以使用 describe-db-parameters AWS CLI 命令从相同的 Aurora PostgreSQL 默认参数组中获取数据库参数列表，如下所示。

对于 Linux、macOS 或 Unix：

```
aws rds describe-db-parameters --db-parameter-group-name default.aurora-postgresql12 \
    --query 'Parameters[].
[{ParameterName:ParameterName,Description:Description,ApplyType:ApplyType,AllowedValues:Allowed
```

对于 Windows：

```
aws rds describe-db-parameters --db-parameter-group-name default.aurora-postgresql12 ^
    --query "Parameters[].
[{ParameterName:ParameterName,Description:Description,ApplyType:ApplyType,AllowedValues:Allowed
```

上述命令将返回数据库集群或数据库参数组中的参数列表，其中包括查询中指定的描述和其他详细信息。以下是示例响应。

```
[
    [
        {
            "ParameterName": "apg_enable_batch_mode_function_execution",
            "ApplyType": "dynamic",
            "Description": "Enables batch-mode functions to process sets of rows at a
 time.",
            "AllowedValues": "0,1"
        }
    ],
    [
        {
            "ParameterName": "apg_enable_correlated_any_transform",
            "ApplyType": "dynamic",
            "Description": "Enables the planner to transform correlated ANY Sublink
 (IN/NOT IN subquery) to JOIN when possible.",
            "AllowedValues": "0,1"
        }
    ],...
```

下表包含 Aurora PostgreSQL 版本 14 数据库集群默认参数和数据库默认参数的值。

Aurora PostgreSQL 集群级参数

您可以使用 AWS 管理控制台、AWS CLI 或 Amazon RDS API 查看特定 Aurora PostgreSQL 版本可用的集群级参数。有关在 RDS 控制台中查看 Aurora PostgreSQL 数据库集群参数组中的参数的信息，请参阅在 Amazon Aurora 中查看数据库集群参数组的参数值。

某些集群级参数并非在所有版本中都可用，有些正要弃用。有关查看特定 Aurora PostgreSQL 版本的参数的信息，请参阅查看 Aurora PostgreSQL 数据库集群和数据库参数。

例如，下表列出了 Aurora PostgreSQL 版本 14 的默认数据库集群参数组中可用的参数。如果在未指定您自定义数据库参数组的情况下创建 Aurora PostgreSQL 数据库集群，则将使用所选版本的 Aurora 数据库集群默认参数组（例如 `default.aurora-postgresql14`、`default.aurora-postgresql13` 等）来创建数据库集群。

有关此相同默认数据库集群参数组的数据库实例参数的列表，请参阅 Aurora PostgreSQL 实例级参数。

参数名称	描述	默认
ansi_constraint_trigger_ordering	更改约束触发条件的触发顺序，使其与 ANSI SQL 标准兼容。	–
ansi_force_foreign_key_checks	无论操作存在的各种触发条件的上下文如何，确保级联删除和级联更新等引用操作始终会发生。	–
ansi_qualified_update_set_target	支持 UPDATE...SET 语句中的 表格和架构限定符。	–
apg_ccm_enabled	启用或禁用集群缓存管理功能。	–
apg_enable_batch_mode_function_execution	启用批处理模式函数，以便一次处理多行。	–
apg_enable_correlated_any_transform	让计划程序尽可能将关联的 ANY 子链接（IN/NOT IN 子查询）转换为 JOIN。	–
apg_enable_function_migration	让计划程序将符合条件的标量函数迁移到 FROM 子句。	–

参数名称	描述	默认
apg_enable_not_in_transform	让计划程序尽可能将 NOT IN 子查询转换为 ANTI JOIN。	–
apg_enable_remove_redundant_inner_joins	让计划程序能够删除冗余的内部联接。	–
apg_enable_semijoin_push_down	让哈希联接能够使用半联接筛选条件。	–
apg_plan_mgmt.capture_plan_baselines	捕获计划基准模式。手动：为任何 SQL 语句启用计划捕获；关闭：禁用计划捕获；自动：为 pg_stat_statements 中满足资格条件的语句启用计划捕获。	off
apg_plan_mgmt.max_databases	设置可以使用 apg_plan_mgmt 管理查询的最大数据库数。	10
apg_plan_mgmt.max_plans	设置 apg_plan_mgmt 可以缓存的最大计划数。	10000
apg_plan_mgmt.plan_retention_period	自上次使用计划至自动删除计划之间的最大天数。	32
apg_plan_mgmt.unapproved_plan_execution_threshold	预估的计划总开销，低于该值则将执行未批准计划。	0
apg_plan_mgmt.use_plan_baselines	仅对托管式语句使用已批准计划或固定计划。	false
application_name	设置要在统计数据和日志中报告的应用程序名称。	–
array_nulls	允许在阵列中输入 NULL 元素。	–

参数名称	描述	默认
aurora_compute_plan_id	可以监控查询执行计划，以检测导致当前数据库负载的执行计划，并跟踪一段时间内执行计划的性能统计信息。有关更多信息，请参阅监控 Aurora PostgreSQL 的查询执行计划。	on
authentication_timeout	(s) 设置允许完成客户端身份验证的最长时间。	–
auto_explain.log_analyze	使用 EXPLAIN ANALYZE 记录计划。	–
auto_explain.log_buffers	记录缓冲区的使用情况。	–
auto_explain.log_format	用于计划记录的 EXPLAIN 格式。	–
auto_explain.log_min_duration	设置如超出即记录计划的最短执行时间。	–
auto_explain.log_nested_statements	记录嵌套语句。	–
auto_explain.log_timing	不仅收集行数，还收集时序数据。	–
auto_explain.log_triggers	在计划中包括触发条件统计数据。	–
auto_explain.log_verbose	使用 EXPLAIN VERBOSE 记录计划。	–
auto_explain.sample_rate	待处理查询的占比。	–
autovacuum	启动 Autovacuum 子进程。	–

参数名称	描述	默认
autovacuum_analyze_scale_factor	在分析之前插入、更新或删除元组的次数，以相对于 reltuple 的占比计。	0.05
autovacuum_analyze_threshold	在分析之前插入、更新或删除元组的最小次数。	–
Autovacuum_freeze_max_age	对表进行 Autovacuum 以防事务 ID 重叠的期限。	–
Autovacuum_max_workers	设置同时运行的 Autovacuum 工作者的最大数量。	GREATEST(DBInstanceClassMemory/64371566592,3)
autovacuum_multixact_freeze_max_age	Autovacuum 表格以防止 Multixact 重叠的 Multixact 期限。	–
Autovacuum_naptime	(s) 两次 Autovacuum 运行之间的睡眠时间。	5
Autovacuum_vacuum_cost_delay	Autovacuum 的 Vacuum 开销延迟，以毫秒为单位（ms）。	5
autovacuum_vacuum_cost_limit	Autovacuum 在小睡之前可用的真空开销量。	GREATEST(log(DBInstanceClassMemory/21474836480)*600,200)
autovacuum_vacuum_insert_scale_factor	Vacuum 之前插入元组的次数，以相对于 reltuple 的占比计。	–
autovacuum_vacuum_insert_threshold	在 vacuum 之前插入元组的最小次数，-1 则表示禁用插入 vacuum。	–
autovacuum_vacuum_scale_factor	Vacuum 之前更新或删除元组的次数，以相对于 reltuple 的占比计。	0.1

参数名称	描述	默认
autovacuum_vacuum_threshold	Vacuum 之前更新或删除元组的最小次数。	–
Autovacuum_work_mem	(kB) 设置要由每个 Autovacuum 工件进程使用的最大内存。	GREATEST(DBInstanceClassMemory/32768,131072)
babelfishpg_tds.default_server_name	默认 Babelfish 服务器名称	Microsoft SQL Server
babelfishpg_tds.listen_addresses	设置要监听 TDS 的主机名或 IP 地址。	*
babelfishpg_tds.port	设置服务器监听的 TDS TCP 端口。	1433
babelfishpg_tds.tds_debug_log_level	设置 TDS 中的日志记录级别，0 表示禁用日志记录	1
babelfishpg_tds.tds_default_numeric_precision	如果引擎未指定要在 TDS 列元数据中发送的数字类型的默认精度，则设置默认精度。	38
babelfishpg_tds.tds_default_numeric_scale	如果引擎未指定要在 TDS 列元数据中发送的数字类型的默认小数位数，则设置默认小数位数。	8
babelfishpg_tds.tds_default_packet_size	设置用于所有连接的 SQL Server 客户端的默认数据包大小	4096
babelfishpg_tds.tds_default_protocol_version	设置用于所有连接的客户端的默认 TDS 协议版本	DEFAULT
babelfishpg_tds.tds_ssl_crypt	设置 SSL 加密选项	0

参数名称	描述	默认
babelfishpg_tds.tds_ssl_max_protocol_version	设置 TDS 会话使用的最高 SSL/TLS 协议版本。	TLSv1.2
babelfishpg_tds.tds_ssl_min_protocol_version	设置 TDS 会话使用的最低 SSL/TLS 协议版本。	在 Aurora PostgreSQL 版本 16 中，为 TLSv1.2；对于比 Aurora PostgreSQL 版本 16 更早的版本，为 TLSv1
babelfishpg_tsql.default_locale	由 CREATE COLLATION 创建的排序规则使用的默认区域设置。	en-US
babelfishpg_tsql.migration_mode	定义是否支持多个用户数据库	在 Aurora PostgreSQL 版本 16 中，为 multi-db；对于比 Aurora PostgreSQL 版本 16 更早的版本，为 single-db
babelfishpg_tsql.server_collation_name	默认服务器排序规则的名称	sql_latin1_general_cp1_ci_as
babelfishpg_tsql.version	设置 @@VERSION 变量的输出	默认值
backend_flush_after	(8 kB) 先前执行的写入操作刷新到磁盘的页数。	–
backslash_quote	设置字符串文本中是否允许有 \\。	–
backtrace_functions	记录这些函数中错误的回溯。	–
bytea_output	设置字节的输出格式。	–
check_function_bodies	在 CREATE FUNCTION 期间检查函数体。	–

参数名称	描述	默认
client_connection_check_interval	设置在运行查询时检查断开连接的时间间隔。	–
client_encoding	设置客户端字符集编码。	UTF8
client_min_messages	设置发送到客户端的消息级别。	–
compute_query_id	计算查询标识符。	自动
config_file	设置服务器主配置文件。	/rdsdbdata/config/postgresql.conf
constraint_exclusion	使计划程序可使用约束优化查询。	–
cpu_index_tuple_cost	设置计划程序预估索引扫描期间处理每个索引条目的开销。	–
cpu_operator_cost	设置计划程序预估处理每个运算符或函数调用的开销。	–
cpu_tuple_cost	设置计划程序预估处理每个元组（行）的开销。	–
cron.database_name	设置数据库存储 pg_cron 元数据表	postgres
cron.log_run	将所有任务运行记录到 job_run_details 表中	on
cron.log_statement	在执行之前记录所有 cron 语句。	off
cron.max_running_jobs	可以同时运行的最大任务数。	5
cron.use_background_workers	启用 pg_cron 的背景工件	on
cursor_tuple_fraction	设置计划程序预估待检索光标行的占比。	–
data_directory	设置服务器数据目录。	/rdsdbdata/db
datestyle	设置日期和时间值的显示格式。	–

参数名称	描述	默认
db_user_namespace	启用各数据库的用户名。	–
deadlock_timeout	(ms) 设置在检查死锁之前等待锁定的时间。	–
debug_pretty_print	缩进分析树和计划树的显示内容。	–
debug_print_parse	记录每个查询的分析树。	–
debug_print_plan	记录每个查询的执行计划。	–
debug_print_rewritten	记录每个查询重写的分析树。	–
default_statistics_target	设置默认统计数据目标。	–
default_tablespace	设置要从中创建表和索引的默认表空间。	–
default_toast_compression	为可压缩值设置默认压缩方法。	–
default_transaction_deferrable	设置新事务的默认可延迟状态。	–
default_transaction_isolation	设置每个新事务的事务隔离级别。	–
default_transaction_read_only	设置新事务的默认只读状态。	–
effective_cache_size	(8kB) 设置计划程序对于磁盘缓存大小的假设。	SUM(DBInstanceClassMemory/12038,-50003)
effective_io_concurrency	磁盘子系统可有效处理的并行请求数。	–
enable_async_append	使计划程序可使用异步追加计划。	–
enable_bitmapscan	使计划程序可使用位图扫描计划。	–

参数名称	描述	默认
enable_gathermerge	使计划程序可使用收集合并计划。	–
enable_hashagg	使计划程序可使用哈希聚合计划。	–
enable_hashjoin	使计划程序可使用哈希联接计划。	–
enable_incremental_sort	使计划程序可使用增量排序步骤。	–
enable_indexonlyscan	使计划程序可使用仅限索引的扫描计划。	–
enable_indexscan	使计划程序可使用索引扫描计划。	–
enable_material	使计划程序可使用具体化。	–
enable_memoize	使计划程序可使用记忆功能	–
enable_mergejoin	使计划程序可使用合并联接计划。	–
enable_nestloop	使计划程序可使用嵌套循环的联接计划。	–
enable_parallel_append	使计划程序可使用并行追加计划。	–
enable_parallel_hash	使计划程序可使用并行哈希计划。	–
enable_partition_pruning	启用计划时和运行时分区修剪。	–
enable_partitionwise_aggregate	启用智能分区聚合和分组。	–
enable_partitionwise_join	启用智能分区联接。	–
enable_seqscan	使计划程序可使用顺序扫描计划。	–
enable_sort	使计划程序可使用显式排序步骤。	–

参数名称	描述	默认
enable_tidscan	使计划程序可使用 TID 扫描计划。	–
escape_string_warning	警告在普通字符串文本中有反斜杠转义符。	–
exit_on_error	发生任何错误时终止会话。	–
extra_float_digits	设置所显示的浮点值位数。	–
force_parallel_mode	强制使用并行查询功能。	–
from_collapse_limit	设置超出其即不折叠子查询的 FROM 列表大小。	–
geqo	启用基因查询优化。	–
geqo_effort	GEQO：用于设置其他 GEQO 参数默认值的工作负载。	–
geqo_generations	GEQO：算法的迭代次数。	–
geqo_pool_size	GEQO：群体中的个体数。	–
geqo_seed	GEQO：随机路径选择的种子。	–
geqo_selection_bias	GEQO：群体中的选择性压力。	–
geqo_threshold	设置超出其即使用 GEQO 的 FROM 项阈值。	–
gin_fuzzy_search_limit	通过允许由 GIN 进行的精确搜索得出的最大结果数。	–
gin_pending_list_limit	(kB) 为 GIN 索引设置待处理列表的最大尺寸。	–
hash_mem_multiplier	用于哈希表的 work_mem 的倍数。	–
hba_file	设置服务器 hba 配置文件。	/rdsdbdata/config/pg_hba.conf

参数名称	描述	默认
hot_standby_feedback	允许从热备用项向主备用项提供反馈，避免查询产生冲突。	on
huge_pages	可以减少数据库实例处理大量连续内存数据块（例如共享缓冲区使用的内存数据块）时的开销。对于除 t3.medium、db.t3.large、db.t4g.medium、db.t4g.large 实例类之外的所有数据库实例类，默认情况下都会开启此功能。	on
ident_file	设置服务器身份配置文件。	/rdsdbdata/config/pg_ident.conf
idle_in_transaction_session_timeout	(ms) 设置允许任何空闲事务的最长持续时间。	86400000
idle_session_timeout	终止空闲（即等待客户端查询）时间超过指定的时间量但未处于未结事务内的任何会话	–
intervalstyle	设置间隔值的显示格式。	–
join_collapse_limit	设置超出其即不平展 JOIN 结构的 FROM 列表大小。	–
krb_caseins_users	设置是否应不区分大小写（true）处理 GSSAPI（通用安全服务 API）用户名。默认情况下，此参数设置为 false，因此 Kerberos 希望用户名区分大小写。有关更多信息，请参阅 PostgreSQL 文档中的 GSSAPI 身份验证。	false
lc_messages	设置显示消息的语言。	–
lc_monetary	设置用于使货币金额格式化的区域设置。	–
lc_numeric	设置用于使数字格式化的区域设置。	–
lc_time	设置用于使日期和时间值格式化的区域设置。	–
listen_addresses	设置要监听的主机名或 IP 地址。	*

参数名称	描述	默认
lo_compat_privileges	启用大型对象权限检查的向后兼容模式。	0
log_autovacuum_min_duration	(ms) 设置如超出即记录 Autovacuum 操作的最短执行时间。	10000
log_connections	记录每个成功的连接。	–
log_destination	设置服务器日志输出的目标。	stderr
log_directory	设置日志文件的目标目录。	/rdsdbdata/log/error
log_disconnections	记录会话结束，包括持续时间。	–
log_duration	记录每个完成的 SQL 语句的持续时间。	–
log_error_verbosity	设置记录消息的详细程度。	–
log_executor_stats	向服务器日志写入执行者性能统计数据。	–
log_file_mode	设置日志文件的文件权限。	0644
log_filename	设置日志文件的文件名模式。	postgresql.log.%Y-%m-%d-%H%M
logging_collector	启动子进程将 stderr 输出和/或 csvlog 捕获到日志文件中。	1
log_hostname	在连接日志中记录主机名。	0
logical_decoding_work_mem	(kB) 在溢出到磁盘之前，每个内部重新排序缓冲区都可以使用这么多内存。	–
log_line_prefix	控制每个日志行前缀的信息。	%t:%r:%u@%d:%p]:
log_lock_waits	记录长锁定等待次数。	–
log_min_duration_sample	(ms) 设置如超出即记录语句示例的最短执行时间。采样率由 log_statement_sample_rate 决定。	–

参数名称	描述	默认
log_min_duration_statement	(ms) 设置如超出即记录语句的最短执行时间。	–
log_min_error_statement	导致记录所有产生此级别或更高级别错误的语句。	–
log_min_messages	设置记录的消息级别。	–
log_parameter_max_length	(B) 记录语句时，将记录的参数值限制为前 N 个字节。	–
log_parameter_max_length_on_error	(B) 报告错误时，将记录的参数值限制为前 N 个字节。	–
log_parser_stats	向服务器日志写入分析器性能统计数据。	–
log_planner_stats	向服务器日志写入计划程序性能统计数据。	–
log_replication_commands	记录每个复制命令。	–
log_rotation_age	(min) 将在 N 分钟后进行日志文件自动轮换。	60
log_rotation_size	(kB) 将在 N kB 后进行日志文件自动轮换。	100000
log_statement	设置所记录的语句类型。	–
log_statement_sample_rate	超过要记录的 log_min_duration_sample 的语句占比。	–
log_statement_stats	向服务器日志写入累计性能统计数据。	–
log_temp_files	(kB) 记录对大于此 kB 数的临时文件的使用情况。	–
log_timezone	设置要在日志消息中使用的时区。	UTC
log_transaction_sample_rate	设置要为新事务记录的事务的占比。	–

参数名称	描述	默认
log_truncate_on_rotation	在日志轮换期间截断同名的现有日志文件。	0
maintenance_io_concurrency	用于维护工作的 effective_io_concurrency 变体。	1
maintenance_work_mem	(kB) 设置要用于维护操作的最大内存。	GREATEST(DBInstanceClassMemory/63963136*1024,65536)
max_connections	设置最大并行连接数。	LEAST(DBInstanceClassMemory/9531392,5000)
max_files_per_process	设置同时为每个服务器进程打开的最大文件数。	–
max_locks_per_transaction	设置每个事务的最大锁定数。	64
max_logical_replication_workers	逻辑复制工件进程的最大数量。	–
max_parallel_maintenance_workers	设置每个维护操作的最大并行进程数。	–
max_parallel_workers	设置可同时处于活动状态的最大并行工件数。	GREATEST($DBInstanceVCPU/2,8)
max_parallel_workers_per_gather	设置每个执行程序节点的最大并行进程数。	–
max_pred_locks_per_page	设置每页谓词锁定元组的最大数量。	–

参数名称	描述	默认
max_pred_locks_per_relation	设置每个关系谓词锁定页和元组的最大数量。	–
max_pred_locks_per_transaction	设置每个事务的最大谓词锁定数。	–
max_prepared_transactions	设置同时准备的最大事务数。	0
max_replication_slots	设置服务器可以支持的最大复制插槽数。	20
max_slot_wal_keep_size	(MB) 如果 WAL 占用了磁盘这么多空间，复制插槽将标记为失败，并释放分段进行删除或循环使用。	–
max_stack_depth	设置最大堆栈长度，以 kB 计。	6144
max_standby_streaming_delay	(ms) 设置在有热备用服务器处理流式 WAL 数据时取消查询之前的最大延迟。	14000
max_sync_workers_per_subscription	每个订阅的最大同步工件数量	2
max_wal_senders	设置同时运行的 WAL 发送方的最大数量。	10
max_worker_processes	设置最大并发工件进程数。	GREATEST($DBInstanceVCPU*2, 8)
min_dynamic_shared_memory	启动时预留的动态共享内存量（MB）。	–
min_parallel_index_scan_size	(8kB) 设置并行扫描的最小索引数据量。	–
min_parallel_table_scan_size	(8kB) 设置并行扫描的最小表格数据量。	–

参数名称	描述	默认
old_snapshot_threshold	在快照因时间过长而无法读取在快照捕获后更改的页面的时间（min）。	–
orafce.nls_date_format	模拟 Oracle 的日期输出行为。	–
orafce.timezone	指定用于 sysdate 函数的时区。	–
parallel_leader_participation	控制 Gather 和 Gather Merge 是否也运行子计划。	–
parallel_setup_cost	设置计划程序预估为并行查询启动工件进程的开销。	–
parallel_tuple_cost	设置计划程序预估将每个元组（行）从工件后端传递给主后端的开销。	–
password_encryption	加密密码。	–
pgaudit.log	指定会话审核日志将记录哪些类语句。	–
pgaudit.log_catalog	指定如果语句中的所有关系都在 pg_catalog 中，则启用会话日志记录。	–
pgaudit.log_level	指定将用于日志条目的日志级别。	–
pgaudit.log_parameter	指定审核日志记录应包括随语句传递的参数。	–
pgaudit.log_relation	指定会话审核日志记录是否应为 SELECT 或 DML 语句中引用的每个关系（TABLE、VIEW 等）创建单独的日志条目。	–
pgaudit.log_statement_once	指定日志记录在语句/子语句组合中的第一个日志条目中包含语句文本和参数，还是在每个条目中都包含。	–
pgaudit.role	指定用于对象审核日志记录的主角色。	–

参数名称	描述	默认
pg_bigm.enable_recheck	其将指定是否执行 Recheck，这是全文搜索的内部过程。	on
pg_bigm.gin_key_limit	其将指定用于全文搜索的搜索关键字的最大 2 元数。	0
pg_bigm.last_update	其将报告 pg_bigm 模块的最后更新日期。	2013.11.22
pg_bigm.similarity_limit	其将指定相似性搜索使用的最低阈值。	0.3
pg_hint_plan.debug_print	记录提示解析的结果。	–
pg_hint_plan.enable_hint	强制计划程序使用查询之前的提示注释中指定的计划。	–
pg_hint_plan.enable_hint_table	强制计划程序不通过使用表查找来获得提示。	–
pg_hint_plan.message_level	调试消息的消息级别。	–
pg_hint_plan.parse_messages	解析错误的消息级别。	–
pglogical.batch_inserts	在可能时执行批量插入	–
pglogical.conflict_log_level	设置用于记录已解决冲突的日志级别。	–
pglogical.conflict_resolution	设置用于解决可解决冲突的方法。	–
pglogical.extra_connection_options	要添加到所有对等节点连接的连接选项	–

参数名称	描述	默认
pglogical.synchronous_commit	pglogical 专用的同步提交值	–
pglogical.use_spi	使用 SPI 而非低级 API 来应用更改	–
pgtle.clientauth_databases_to_skip	要使用 clientauth 功能需要跳过的数据库列表。	–
pgtle.clientauth_db_name	控制用于 clientauth 功能的数据库。	–
pgtle.clientauth_num_parallel_workers	用于 clientauth 功能的后台工作进程数量。	–
pgtle.clientauth_users_to_skip	要使用 clientauth 功能需要跳过的用户列表。	–
pgtle.enable_clientauth	启用 clientauth 功能。	–
pgtle.passcheck_db_name	设置用于集群范围的密码检查功能的数据库。	–
pg_prewarm.autoprewarm	启动自动预热工件。	–
pg_prewarm.autoprewarm_interval	设置共享缓冲区转储之间的间隔	–
pg_similarity.block_is_normalized	设置结果值是否标准化。	–
pg_similarity.block_threshold	设置 Block 相似性函数使用的阈值。	–
pg_similarity.block_tokenizer	设置 Block 相似性函数的标记器。	–

参数名称	描述	默认
pg_similarity.cosine_is_normalized	设置结果值是否标准化。	–
pg_similarity.cosine_threshold	设置 Cosine 相似性函数使用的阈值。	–
pg_similarity.cosine_tokenizer	设置 Cosine 相似性函数的标记器。	–
pg_similarity.dice_is_normalized	设置结果值是否标准化。	–
pg_similarity.dice_threshold	设置 Dice 相似性测度所使用的阈值。	–
pg_similarity.dice_tokenizer	设置 Dice 相似性测度的标记器。	–
pg_similarity.euclidean_is_normalized	设置结果值是否标准化。	–
pg_similarity.euclidean_threshold	设置 Euclidean 相似性测度所使用的阈值。	–
pg_similarity.euclidean_tokenizer	设置 Euclidean 相似性测度的标记器。	–
pg_similarity.hamming_is_normalized	设置结果值是否标准化。	–
pg_similarity.hamming_threshold	设置 Block 相似性指标使用的阈值。	–
pg_similarity.jaccard_is_normalized	设置结果值是否标准化。	–
pg_similarity.jaccard_threshold	设置 Jaccard 相似性测度所使用的阈值。	–

参数名称	描述	默认
pg_similarity.jaccard_tokenizer	设置 Jaccard 相似性测度的标记器。	–
pg_similarity.jaro_is_normalized	设置结果值是否标准化。	–
pg_similarity.jaro_threshold	设置 Jaro 相似性测度所使用的阈值。	–
pg_similarity.jarowinkler_is_normalized	设置结果值是否标准化。	–
pg_similarity.jarowinkler_threshold	设置 Jarowinkler 相似性测度所使用的阈值。	–
pg_similarity.levenshtein_is_normalized	设置结果值是否标准化。	–
pg_similarity.levenshtein_threshold	设置 Levenshtein 相似性测度所使用的阈值。	–
pg_similarity.matching_is_normalized	设置结果值是否标准化。	–
pg_similarity.matching_threshold	设置匹配系数相似性测度所使用的阈值。	–
pg_similarity.matching_tokenizer	设置匹配系数相似性测度的标记器。	–
pg_similarity.mongeelkan_is_normalized	设置结果值是否标准化。	–
pg_similarity.mongeelkan_threshold	设置 Monge-Elkan 相似性测度所使用的阈值。	–
pg_similarity.mongeelkan_tokenizer	设置 Monge-Elkan 相似性测度的标记器。	–

参数名称	描述	默认
pg_similarity.nw_gap_penalty	设置 Needleman-Wunsch 相似性测度所使用的空位罚分。	–
pg_similarity.nw_is_normalized	设置结果值是否标准化。	–
pg_similarity.nw_threshold	设置 Needleman-Wunsch 相似性测度所使用的阈值。	–
pg_similarity.overlap_is_normalized	设置结果值是否标准化。	–
pg_similarity.overlap_threshold	设置重叠系数相似性测度所使用的阈值。	–
pg_similarity.overlap_tokenizer	设置重叠系数相似性测度的标记器。	–
pg_similarity.qgram_is_normalized	设置结果值是否标准化。	–
pg_similarity.qgram_threshold	设置 Q-Gram 相似性测度所使用的阈值。	–
pg_similarity.qgram_tokenizer	设置 Q-Gram 测度的标记器。	–
pg_similarity.swg_is_normalized	设置结果值是否标准化。	–
pg_similarity.swg_threshold	设置 Smith-Waterman-Gotoh 相似性测度所使用的阈值。	–
pg_similarity.sw_is_normalized	设置结果值是否标准化。	–
pg_similarity.sw_threshold	设置 Smith-Waterman 相似性测度所使用的阈值。	–

参数名称	描述	默认
pg_stat_statements.max	设置 pg_stat_statements 跟踪的最大语句数。	–
pg_stat_statements.save	保存 pg_stat_statements 在服务器关闭期间的统计数据。	–
pg_stat_statements.track	选择 pg_stat_statements 跟踪哪些语句。	–
pg_stat_statements.track_planning	选择 pg_stat_statements 是否跟踪计划持续时间。	–
pg_stat_statements.track_utility	选择 pg_stat_statements 是否跟踪实用程序命令。	–
plan_cache_mode	控制计划程序对自定义或通用计划的选择。	–
端口	设置服务器侦听的 TCP 端口。	EndPointPort
postgis.gdal_enabled_drivers	启用或禁用 Postgres 9.3.5 及更高版本中与 PostGIS 搭配使用的 GDAL 驱动程序。	ENABLE_ALL
quote_all_identifiers	在生成 SQL 片段时，向所有标识符添加引号。	–
random_page_cost	设置计划程序预估非连续提取磁盘页面的开销。	–
rdkit.dice_threshold	Dice 相似度阈值较低。通过 # 运算，相似性低于阈值的分子不相似。	–
rdkit.do_chiral_sss	子结构匹配中是否考虑到立体化学。如果为 false，则子结构匹配中未使用立体化学信息。	–
rdkit.tanimoto_threshold	Tanimoto 相似度阈值较低。通过 % 运算，相似性低于阈值的分子不相似。	–
rds.accepted_password_auth_method	使用本地存储的密码强制对连接进行身份验证。	md5+scram

参数名称	描述	默认
rds.adaptive_autovacuum	用于启用/禁用自适应 Autovacuum 的 RDS 参数。	1
rds.babelfish_status	用于启用/禁用 Babelfish for Aurora PostgreSQL 的 RDS 参数。	off
rds.enable_plan_management	启用或禁用 apg_plan_mgmt 扩展。	0

参数名称	描述	默认
rds.extensions	RDS 提供的扩展的列表	address_standardizer、address_standardizer_data_us、apg_plan_mgmt、aurora_stat_utils、amcheck、autoinc、aws_commons、aws_ml、aws_s3、aws_lambda、bool_plperl、bloom、btree_gin、btree_gist、citext、cube、dblink、dict_int、dict_xsyn、earthdistance、fuzzystrmatch、hll、hstore、hstore_plperl、insert_username、intagg、intarray、ip4r、isn、jsonb_plperl、lo、log_fdw、ltree、moddatetime、old_snapshot、oracle_fdw、orafce、pgaudit、pgcrypto、pglogical、pgrouting、pgrowlocks、pgstattuple、pgtap、pg_bigm、pg_buffercache、pg_cron、pg_freespacemap、pg_hint_plan、pg_partm

参数名称	描述	默认
		an、pg_prewarm、pg_proctab、pg_repack、pg_similarity、pg_stat_statements、pg_trgm、pg_visibility、plcoffee、plls、plperl、plpgsql、plprofiler、pltcl、plv8、postgis、postgis_tiger_geocoder、postgis_raster、postgis_topology、postgres_fdw、prefix、rdkit、rds_tools、refint、sslinfo、tablefunc、tds_fdw、test_parser、tsm_system_rows、tsm_system_time、unaccent、uuid-ossp
rds.force_admin_logging_level	请参阅客户数据库中 RDS 管理员用户操作的日志消息。	–
rds.force_autovacuum_logging_level	请参阅与 Autovacuum 操作相关的日志消息。	WARNING
rds.force_ssl	强制执行 SSL 连接。	0

参数名称	描述	默认
rds.global_db_rpo	恢复点目标阈值（以秒为单位），违反该值时会阻止用户提交（s）。 **Important** 此参数适用于基于 Aurora PostgreSQL 的全球数据库。对于非全球数据库，请将其保留为默认值。有关使用该参数的更多信息，请参阅the section called “管理 RPO（Aurora PostgreSQL）”。	–
rds.logical_replication	启用逻辑解码。	0
rds.logically_replicate_unlogged_tables	逻辑上复制未记录的表。	1
rds.log_retention_period	Amazon RDS 将删除超过 N 分钟的 PostgreSQL 日志。	4320
rds.pg_stat_ramdisk_size	统计数据虚拟磁盘的大小（以 MB 为单位）。将以非零值设置虚拟磁盘。此参数仅在 Aurora PostgreSQL 14 及更低版本中可用。	0
rds.rds_superuser_reserved_connections	设置为 rds_superuser 保留的连接插槽数。此参数仅在版本 15 及更高版本中提供。有关更多信息，请参阅 PostgreSQL 文档预留连接。	2
rds.restrict_password_commands	将与密码相关的命令限制为 rds_password 的成员	–
rds.superuser_variables	提升 rds_superuser 修改语句的仅限超级用户的变量列表。	session_replication_role
recovery_init_sync_method	设置在崩溃恢复之前同步数据目录的方法。	syncfs

参数名称	描述	默认
remove_temp_files_after_crash	后端崩溃后删除临时文件。	0
restart_after_crash	后端崩溃后重新初始化服务器。	–
row_security	启用行安全。	–
search_path	设置针对非架构限定名称的架构搜索顺序。	–
seq_page_cost	设置计划程序预估连续提取磁盘页面的开销。	–
session_replication_role	设置触发器和重写规则的会话行为。	–
shared_buffers	(8 kB) 设置服务器使用的共享内存缓冲区数。	SUM(DBInstanceClassMemory/12038,-50003)
shared_preload_libraries	列出要预加载到服务器的共享库。	pg_stat_statements
ssl	启用 SSL 连接。	1
ssl_ca_file	SSL 服务器授权文件的位置。	/rdsdbdata/rds-metadata/ca-cert.pem
ssl_cert_file	SSL 服务器证书文件的位置。	/rdsdbdata/rds-metadata/server-cert.pem
ssl_ciphers	设置允许在安全连接上使用的 TLS 密码的列表。	–
ssl_crl_dir	SSL 证书吊销列表目录的位置。	/rdsdbdata/rds-metadata/ssl_crl_dir/
ssl_key_file	SSL 服务器私有密钥文件的位置	/rdsdbdata/rds-metadata/server-key.pem

参数名称	描述	默认
ssl_max_protocol_version	设置允许的最高 SSL/TLS 协议版本	–
ssl_min_protocol_version	设置允许的最低 SSL/TLS 协议版本	TLSv1.2
standard_conforming_strings	导致 ... 字符串按字面处理反斜杠。	–
statement_timeout	(ms) 设置任何语句允许的最长持续时间。	–
stats_temp_directory	将临时统计数据文件写入指定目录。	/rdsdbdata/db/pg_stat_tmp
superuser_reserved_connections	设置为超级用户预留的连接插槽数。	3
synchronize_seqscans	启用同步顺序扫描。	–
synchronous_commit	设置当前事务同步级别。	on
tcp_keepalives_count	重新传输 TCP 保持连接信号的最大次数。	–
tcp_keepalives_idle	(s) 发出两次 TCP 保持连接信号之间的时间。	–
tcp_keepalives_interval	(s) 两次 TCP 保持连接信号重新传输之间的时间。	–
temp_buffers	(8 kB) 设置每个会话使用的临时缓冲区的最大数量。	–
temp_file_limit	限制给定 PostgreSQL 进程可用于临时文件的总磁盘空间（以 kB 为单位），不包括用于显式临时表的空间	–1
temp_tablespaces	选择用于临时表和排序文件的表空间。	–

参数名称	描述	默认
timezone	设置用于显示和解译时间戳的时区。	UTC
track_activities	收集有关执行命令的信息。	–
track_activity_query_size	设置为 pg_stat_activity.current_query 保留的大小，以字节计。	4096
track_commit_timestamp	收集事务提交时间。	–
track_counts	收集有关数据库活动的统计数据。	–
track_functions	收集有关数据库活动的函数级别统计数据。	pl
track_io_timing	收集有关数据库 IO 活动的时序统计数据。	1
track_wal_io_timing	收集 WAL I/O 活动的时序统计数据。	–
transform_null_equals	将 expr=NULL 视为 expr IS NULL。	–
update_process_title	更新进程标题以显示活动的 SQL 命令。	–
vacuum_cost_delay	(ms) Vacuum 开销延迟，以毫秒为单位。	–
vacuum_cost_limit	小睡之前可用的真空开销量。	–
vacuum_cost_page_dirty	由真空弄脏的页面的真空开销。	–
vacuum_cost_page_hit	在缓冲区缓存中找到的页面的真空开销。	–
vacuum_cost_page_miss	在缓冲区缓存中未找到的页面的真空开销。	0
vacuum_defer_cleanup_age	VACUUM 和 HOT 清理应推迟的事务数（如果有）。	–

参数名称	描述	默认
vacuum_failsafe_age	VACUM 应触发故障保护以避免全面停机的期限。	1200000000
vacuum_freeze_min_age	VACUUM 应冻结表格行的最短期限。	–
vacuum_freeze_table_age	VACUUM 应扫描整个表以冻结元组的期限。	–
vacuum_multixact_failsafe_age	VACUM 应触发故障保护以避免全面停机的 Multixact 期限。	1200000000
vacuum_multixact_freeze_min_age	VACUUM 应冻结表格行中的 MultiXactId 的最短期限。	–
vacuum_multixact_freeze_table_age	VACUUM 应扫描整个表以冻结元组的 Multixact 期限。	–
wal_buffers	设置 WAL 的共享内存中的磁盘页面缓冲区数（8kB）。	–
wal_receiver_create_temp_slot	设置如果没有配置永久插槽，WAL 接收方是否应创建临时复制插槽。	0
wal_receiver_status_interval	(s) 设置 WAL 接收方状态报告到主服务器之间的最大间隔。	–
wal_receiver_timeout	(ms) 设置从主服务器接收数据的最长等待时间。	30000
wal_sender_timeout	(ms) 设置等待 WAL 复制的最长时间。	–
work_mem	(kB) 设置要用于查询工作区的最大内存。	–
xmlbinary	设置如何将二进制值编码到 XML 中。	–
xmloption	设置要将隐式分析和序列化操作中的 XML 数据视为文档还是内容片段。	–

Aurora PostgreSQL 实例级参数

您可以使用 AWS 管理控制台、AWS CLI 或 Amazon RDS API 查看特定 Aurora PostgreSQL 版本可用的实例级参数。有关在 RDS 控制台中查看 Aurora PostgreSQL 数据库参数组中的参数的信息，请参阅在 Amazon Aurora 中查看数据库参数组的参数值。

某些实例级参数并非在所有版本中都可用，有些正要弃用。有关查看特定 Aurora PostgreSQL 版本的参数的信息，请参阅查看 Aurora PostgreSQL 数据库集群和数据库参数。

例如，下表列出了适用于 Aurora PostgreSQL 数据库集群中特定数据库实例的参数。此列表是通过运行 describe-db-parameters AWS CLI 命令生成，命令中 `--db-parameter-group-name` 值为 `default.aurora-postgresql14`。

有关此相同默认数据库参数组的数据库集群参数的列表，请参阅 Aurora PostgreSQL 集群级参数。

参数名称	描述	默认
apg_enable_batch_mode_function_execution	启用批处理模式函数，以便一次处理多行。	–
apg_enable_correlated_any_transform	让计划程序尽可能将关联的 ANY 子链接（IN/NOT IN 子查询）转换为 JOIN。	–
apg_enable_function_migration	让计划程序将符合条件的标量函数迁移到 FROM 子句。	–
apg_enable_not_in_transform	让计划程序尽可能将 NOT IN 子查询转换为 ANTI JOIN。	–
apg_enable_remove_redundant_inner_joins	让计划程序能够删除冗余的内部联接。	–
apg_enable_semijoin_push_down	让哈希联接能够使用半联接筛选条件。	–
apg_plan_mgmt.capture_plan_baselines	捕获计划基准模式。手动：为任何 SQL 语句启用计划捕获；关闭：禁用计划捕获；自动：为 pg_stat_statements 中满足资格条件的语句启用计划捕获。	off

参数名称	描述	默认
apg_plan_mgmt.max_databases	设置可以使用 apg_plan_mgmt 管理查询的最大数据库数。	10
apg_plan_mgmt.max_plans	设置 apg_plan_mgmt 可以缓存的最大计划数。	10000
apg_plan_mgmt.plan_retention_period	自上次使用计划至自动删除计划之间的最大天数。	32
apg_plan_mgmt.unapproved_plan_execution_threshold	预估的计划总开销，低于该值则将执行未批准计划。	0
apg_plan_mgmt.use_plan_baselines	仅对托管式语句使用已批准计划或固定计划。	false
application_name	设置要在统计数据和日志中报告的应用程序名称。	–
aurora_compute_plan_id	可以监控查询执行计划，以检测导致当前数据库负载的执行计划，并跟踪一段时间内执行计划的性能统计信息。有关更多信息，请参阅监控 Aurora PostgreSQL 的查询执行计划。	on
aurora_temp_space_size	(MB）在具有支持的实例类的 Aurora I/O 优化型集群上为启用了优化型读取功能的临时对象设置分配空间的大小。	DBInstanceClassMemory/524288
authentication_timeout	设置允许完成客户端身份验证的最长时间（s）。	–
auto_explain.log_analyze	使用 EXPLAIN ANALYZE 记录计划。	–
auto_explain.log_buffers	记录缓冲区的使用情况。	–

参数名称	描述	默认
auto_explain.log_format	用于计划记录的 EXPLAIN 格式。	–
auto_explain.log_min_duration	设置如超出即记录计划的最短执行时间。	–
auto_explain.log_nested_statements	记录嵌套语句。	–
auto_explain.log_timing	不仅收集行数，还收集时序数据。	–
auto_explain.log_triggers	在计划中包括触发条件统计数据。	–
auto_explain.log_verbose	使用 EXPLAIN VERBOSE 记录计划。	–
auto_explain.sample_rate	待处理查询的占比。	–
babelfishpg_tds.listen_addresses	设置要侦听 TDS 的主机名或 IP 地址。	*
babelfishpg_tds.tds_debug_log_level	设置 TDS 中的日志记录级别，0 表示禁用日志记录	1
backend_flush_after	先前执行的写入操作刷写到磁盘的页数（8kB）。	–
bytea_output	设置字节的输出格式。	–
check_function_bodies	在 CREATE FUNCTION 期间检查函数体。	–
client_connection_check_interval	设置在运行查询时检查断开连接的时间间隔。	–

参数名称	描述	默认
client_min_messages	设置发送到客户端的消息级别。	–
config_file	设置服务器主配置文件。	/rdsdbdata/config/postgresql.conf
constraint_exclusion	使计划程序可使用约束优化查询。	–
cpu_index_tuple_cost	设置计划程序预估索引扫描期间处理每个索引条目的开销。	–
cpu_operator_cost	设置计划程序预估处理每个运算符或函数调用的开销。	–
cpu_tuple_cost	设置计划程序预估处理每个元组（行）的开销。	–
cron.database_name	设置数据库存储 pg_cron 元数据表	postgres
cron.log_run	将所有任务运行记录到 job_run_details 表中	on
cron.log_statement	在执行之前记录所有 cron 语句。	off
cron.max_running_jobs	可以同时运行的最大任务数。	5
cron.use_background_workers	启用 pg_cron 的背景工件	on
cursor_tuple_fraction	设置计划程序预估待检索光标行的占比。	–
db_user_namespace	启用各数据库的用户名。	–
deadlock_timeout	设置在检查死锁之前等待锁定的时间（ms）。	–
debug_pretty_print	缩进分析树和计划树的显示内容。	–
debug_print_parse	记录每个查询的分析树。	–
debug_print_plan	记录每个查询的执行计划。	–

参数名称	描述	默认
debug_print_rewritten	记录每个查询重写的分析树。	–
default_statistics_target	设置默认统计数据目标。	–
default_transaction_deferrable	设置新事务的默认可延迟状态。	–
default_transaction_isolation	设置每个新事务的事务隔离级别。	–
default_transaction_read_only	设置新事务的默认只读状态。	–
effective_cache_size	设置计划程序对于磁盘缓存大小的假设（8kB）。	SUM(DBInstanceClassMemory/12038,-50003
effective_io_concurrency	磁盘子系统可有效处理的并行请求数。	–
enable_async_append	使计划程序可使用异步追加计划。	–
enable_bitmapscan	使计划程序可使用位图扫描计划。	–
enable_gathermerge	使计划程序可使用收集合并计划。	–
enable_hashagg	使计划程序可使用哈希聚合计划。	–
enable_hashjoin	使计划程序可使用哈希联接计划。	–
enable_incremental_sort	使计划程序可使用增量排序步骤。	–
enable_indexonlyscan	使计划程序可使用仅限索引的扫描计划。	–
enable_indexscan	使计划程序可使用索引扫描计划。	–
enable_material	使计划程序可使用具体化。	–

参数名称	描述	默认
enable_memoize	使计划程序可使用记忆功能	–
enable_mergejoin	使计划程序可使用合并联接计划。	–
enable_nestloop	使计划程序可使用嵌套循环的联接计划。	–
enable_parallel_append	使计划程序可使用并行追加计划。	–
enable_parallel_hash	使计划程序可使用并行哈希计划。	–
enable_partition_pruning	启用计划时和运行时分区修剪。	–
enable_partitionwise_aggregate	启用智能分区聚合和分组。	–
enable_partitionwise_join	启用智能分区联接。	–
enable_seqscan	使计划程序可使用顺序扫描计划。	–
enable_sort	使计划程序可使用显式排序步骤。	–
enable_tidscan	使计划程序可使用 TID 扫描计划。	–
escape_string_warning	警告在普通字符串文本中有反斜杠转义符。	–
exit_on_error	发生任何错误时终止会话。	–
force_parallel_mode	强制使用并行查询功能。	–
from_collapse_limit	设置超出其即不折叠子查询的 FROM 列表大小。	–
geqo	启用基因查询优化。	–

参数名称	描述	默认
geqo_effort	GEQO：用于设置其他 GEQO 参数默认值的工作负载。	–
geqo_generations	GEQO：算法的迭代次数。	–
geqo_pool_size	GEQO：群体中的个体数。	–
geqo_seed	GEQO：随机路径选择的种子。	–
geqo_selection_bias	GEQO：群体中的选择性压力。	–
geqo_threshold	设置超出其即使用 GEQO 的 FROM 项阈值。	–
gin_fuzzy_search_limit	通过允许由 GIN 进行的精确搜索得出的最大结果数。	–
gin_pending_list_limit	为 GIN 索引设置待处理列表的最大大小（kB）。	–
hash_mem_multiplier	用于哈希表的 work_mem 的倍数。	–
hba_file	设置服务器 hba 配置文件。	/rdsdbdata/config/pg_hba.conf
hot_standby_feedback	允许从热备用项向主备用项提供反馈，避免查询产生冲突。	on
ident_file	设置服务器身份配置文件。	/rdsdbdata/config/pg_ident.conf
idle_in_transaction_session_timeout	设置允许任何空闲事务的最长持续时间（ms）。	86400000
idle_session_timeout	终止空闲（即等待客户端查询）时间超过指定的时间量但未处于未结事务内的任何会话	–
join_collapse_limit	设置超出其即不平展 JOIN 结构的 FROM 列表大小。	–

参数名称	描述	默认
lc_messages	设置显示消息的语言。	–
listen_addresses	设置要侦听的主机名或 IP 地址。	*
lo_compat_privileges	启用大型对象权限检查的向后兼容模式。	0
log_connections	记录每个成功的连接。	–
log_destination	设置服务器日志输出的目标。	stderr
log_directory	设置日志文件的目标目录。	/rdsdbdata/log/error
log_disconnections	记录会话结束，包括持续时间。	–
log_duration	记录每个完成的 SQL 语句的持续时间。	–
log_error_verbosity	设置记录消息的详细程度。	–
log_executor_stats	向服务器日志写入执行者性能统计数据。	–
log_file_mode	设置日志文件的文件权限。	0644
log_filename	设置日志文件的文件名模式。	postgresql.log.%Y-%m-%d-%H%M
logging_collector	启动子进程将 stderr 输出和/或 csvlog 捕获到日志文件中。	1
log_hostname	在连接日志中记录主机名。	0
logical_decoding_work_mem	在溢出到磁盘之前，每个内部重新排序缓冲区都可以使用这么多内存（kB）。	–
log_line_prefix	控制每个日志行前缀的信息。	%t:%r:%u@%d:%p]:
log_lock_waits	记录长锁定等待次数。	–

参数名称	描述	默认
log_min_duration_sample	设置如超出即记录语句示例的最短执行时间（ms）。采样率由 log_statement_sample_rate 决定。	–
log_min_duration_statement	设置如超出即记录语句的最短执行时间（ms）。	–
log_min_error_statement	导致记录所有产生此级别或更高级别错误的语句。	–
log_min_messages	设置记录的消息级别。	–
log_parameter_max_length	记录语句时，将记录的参数值限制为前 N 个字节（B）。	–
log_parameter_max_length_on_error	报告错误时，将记录的参数值限制为前 N 个字节（B）。	–
log_parser_stats	向服务器日志写入分析器性能统计数据。	–
log_planner_stats	向服务器日志写入计划程序性能统计数据。	–
log_replication_commands	记录每个复制命令。	–
log_rotation_age	N 分钟后将发生日志文件自动轮换（min）。	60
log_rotation_size	在 N kB 后将发生日志文件自动轮换（kB）。	100000
log_statement	设置所记录的语句类型。	–
log_statement_sample_rate	超过要记录的 log_min_duration_sample 的语句占比。	–
log_statement_stats	向服务器日志写入累计性能统计数据。	–
log_temp_files	记录大于此 kB 数的临时文件的使用情况（kB）。	–

参数名称	描述	默认
log_timezone	设置要在日志消息中使用的时区。	UTC
log_truncate_on_rotation	在日志轮换期间截断同名的现有日志文件。	0
maintenance_io_concurrency	用于维护工作的 effective_io_concurrency 变体。	1
maintenance_work_mem	设置要用于维护操作的最大内存（kB）。	GREATEST(DBInstanceClassMemory/63963136*1024,65536
max_connections	设置最大并行连接数。	LEAST(DBInstanceClassMemory/9531392,5000
max_files_per_process	设置同时为每个服务器进程打开的最大文件数。	–
max_locks_per_transaction	设置每个事务的最大锁定数。	64
max_parallel_maintenance_workers	设置每个维护操作的最大并行进程数。	–
max_parallel_workers	设置可同时处于活动状态的最大并行工件数。	GREATEST($DBInstanceVCPU/2,8
max_parallel_workers_per_gather	设置每个执行程序节点的最大并行进程数。	–
max_pred_locks_per_page	设置每页谓词锁定元组的最大数量。	–

参数名称	描述	默认
max_pred_locks_per_relation	设置每个关系谓词锁定页和元组的最大数量。	–
max_pred_locks_per_transaction	设置每个事务的最大谓词锁定数。	–
max_slot_wal_keep_size	如果 WAL 占用了磁盘这么多空间，复制插槽将标记为失败，并释放分段进行删除或循环使用（MB）。	–
max_stack_depth	设置最大堆栈长度，以 kB 计（kB）。	6144
max_standby_streaming_delay	设置在有热备用服务器处理流式传输 WAL 数据时取消查询之前的最大延迟（ms）。	14000
max_worker_processes	设置最大并发工件进程数。	GREATEST($DBInstanceVCPU*2, 8
min_dynamic_shared_memory	启动时预留的动态共享内存量（MB）。	–
min_parallel_index_scan_size	设置并行扫描的最小索引数据量（8kB）。	–
min_parallel_table_scan_size	设置并行扫描的最小表格数据量（8kB）。	–
old_snapshot_threshold	在快照因时间过长而无法读取捕获快照后更改的页面的时间（min）。	–
parallel_leader_participation	控制 Gather 和 Gather Merge 是否也运行子计划。	–
parallel_setup_cost	设置计划程序预估为并行查询启动工件进程的开销。	–

参数名称	描述	默认
parallel_tuple_cost	设置计划程序预估将每个元组（行）从工件后端传递给主后端的开销。	–
pgaudit.log	指定会话审核日志将记录哪些类语句。	–
pgaudit.log_catalog	指定如果语句中的所有关系都在 pg_catalog 中，则启用会话日志记录。	–
pgaudit.log_level	指定将用于日志条目的日志级别。	–
pgaudit.log_parameter	指定审核日志记录应包括随语句传递的参数。	–
pgaudit.log_relation	指定会话审核日志记录是否应为 SELECT 或 DML 语句中引用的每个关系（TABLE、VIEW 等）创建单独的日志条目。	–
pgaudit.log_statement_once	指定日志记录在语句/子语句组合中的第一个日志条目中包含语句文本和参数，还是在每个条目中都包含。	–
pgaudit.role	指定用于对象审核日志记录的主角色。	–
pg_bigm.enable_recheck	其将指定是否执行 Recheck，这是全文搜索的内部过程。	on
pg_bigm.gin_key_limit	其将指定用于全文搜索的搜索关键字的最大 2 元数。	0
pg_bigm.last_update	其将报告 pg_bigm 模块的最后更新日期。	2013.11.22
pg_bigm.similarity_limit	其将指定相似性搜索使用的最低阈值。	0.3
pg_hint_plan.debug_print	记录提示解析的结果。	–
pg_hint_plan.enable_hint	强制计划程序使用查询之前的提示注释中指定的计划。	–

参数名称	描述	默认
pg_hint_plan.enable_hint_table	强制计划程序不通过使用表查找来获得提示。	–
pg_hint_plan.message_level	调试消息的消息级别。	–
pg_hint_plan.parse_messages	解析错误的消息级别。	–
pglogical.batch_inserts	在可能时执行批量插入	–
pglogical.conflict_log_level	设置用于记录已解决冲突的日志级别。	–
pglogical.conflict_resolution	设置用于解决可解决冲突的方法。	–
pglogical.extra_connection_options	要添加到所有对等节点连接的连接选项	–
pglogical.synchronous_commit	pglogical 专用的同步提交值	–
pglogical.use_spi	使用 SPI 而非低级 API 来应用更改	–
pg_similarity.block_is_normalized	设置结果值是否标准化。	–
pg_similarity.block_threshold	设置 Block 相似性函数使用的阈值。	–
pg_similarity.block_tokenizer	设置 Block 相似性函数的标记器。	–
pg_similarity.cosine_is_normalized	设置结果值是否标准化。	–

参数名称	描述	默认
pg_similarity.cosine_threshold	设置 Cosine 相似性函数使用的阈值。	–
pg_similarity.cosine_tokenizer	设置 Cosine 相似性函数的标记器。	–
pg_similarity.dice_is_normalized	设置结果值是否标准化。	–
pg_similarity.dice_threshold	设置 Dice 相似性测度所使用的阈值。	–
pg_similarity.dice_tokenizer	设置 Dice 相似性测度的标记器。	–
pg_similarity.euclidean_is_normalized	设置结果值是否标准化。	–
pg_similarity.euclidean_threshold	设置 Euclidean 相似性测度所使用的阈值。	–
pg_similarity.euclidean_tokenizer	设置 Euclidean 相似性测度的标记器。	–
pg_similarity.hamming_is_normalized	设置结果值是否标准化。	–
pg_similarity.hamming_threshold	设置 Block 相似性指标使用的阈值。	–
pg_similarity.jaccard_is_normalized	设置结果值是否标准化。	–
pg_similarity.jaccard_threshold	设置 Jaccard 相似性测度所使用的阈值。	–
pg_similarity.jaccard_tokenizer	设置 Jaccard 相似性测度的标记器。	–

参数名称	描述	默认
pg_similarity.jaro_is_normalized	设置结果值是否标准化。	–
pg_similarity.jaro_threshold	设置 Jaro 相似性测度所使用的阈值。	–
pg_similarity.jarowinkler_is_normalized	设置结果值是否标准化。	–
pg_similarity.jarowinkler_threshold	设置 Jarowinkler 相似性测度所使用的阈值。	–
pg_similarity.levenshtein_is_normalized	设置结果值是否标准化。	–
pg_similarity.levenshtein_threshold	设置 Levenshtein 相似性测度所使用的阈值。	–
pg_similarity.matching_is_normalized	设置结果值是否标准化。	–
pg_similarity.matching_threshold	设置匹配系数相似性测度所使用的阈值。	–
pg_similarity.matching_tokenizer	设置匹配系数相似性测度的标记器。	–
pg_similarity.mongeelkan_is_normalized	设置结果值是否标准化。	–
pg_similarity.mongeelkan_threshold	设置 Monge-Elkan 相似性测度所使用的阈值。	–
pg_similarity.mongeelkan_tokenizer	设置 Monge-Elkan 相似性测度的标记器。	–
pg_similarity.nw_gap_penalty	设置 Needleman-Wunsch 相似性测度所使用的空位罚分。	–

参数名称	描述	默认
pg_similarity.nw_is_normalized	设置结果值是否标准化。	–
pg_similarity.nw_threshold	设置 Needleman-Wunsch 相似性测度所使用的阈值。	–
pg_similarity.overlap_is_normalized	设置结果值是否标准化。	–
pg_similarity.overlap_threshold	设置重叠系数相似性测度所使用的阈值。	–
pg_similarity.overlap_tokenizer	设置重叠系数相似性测度的标记器。	–
pg_similarity.qgram_is_normalized	设置结果值是否标准化。	–
pg_similarity.qgram_threshold	设置 Q-Gram 相似性测度所使用的阈值。	–
pg_similarity.qgram_tokenizer	设置 Q-Gram 测度的标记器。	–
pg_similarity.swg_is_normalized	设置结果值是否标准化。	–
pg_similarity.swg_threshold	设置 Smith-Waterman-Gotoh 相似性测度所使用的阈值。	–
pg_similarity.sw_is_normalized	设置结果值是否标准化。	–
pg_similarity.sw_threshold	设置 Smith-Waterman 相似性测度所使用的阈值。	–
pg_stat_statements.max	设置 pg_stat_statements 跟踪的最大语句数。	–

参数名称	描述	默认
pg_stat_statements.save	保存 pg_stat_statements 在服务器关闭期间的统计数据。	–
pg_stat_statements.track	选择 pg_stat_statements 跟踪哪些语句。	–
pg_stat_statements.track_planning	选择 pg_stat_statements 是否跟踪计划持续时间。	–
pg_stat_statements.track_utility	选择 pg_stat_statements 是否跟踪实用程序命令。	–
postgis.gdal_enabled_drivers	启用或禁用 Postgres 9.3.5 及更高版本中与 PostGIS 搭配使用的 GDAL 驱动程序。	ENABLE_ALL
quote_all_identifiers	在生成 SQL 片段时，向所有标识符添加引号。	–
random_page_cost	设置计划程序预估非连续提取磁盘页面的开销。	–
rds.enable_memory_management	改进了 Aurora PostgreSQL 12.17、13.13、14.10、15.5 及更高版本中的内存管理功能，可防止因可用内存不足而导致的稳定性问题和数据库重启。有关更多信息，请参阅 改进了 Aurora PostgreSQL 中的内存管理。	True
rds.force_admin_logging_level	请参阅客户数据库中 RDS 管理员用户操作的日志消息。	–
rds.log_retention_period	Amazon RDS 将删除超过 N 分钟的 PostgreSQL 日志。	4320
rds.memory_allocation_guard	改进了 Aurora PostgreSQL 11.21、12.16、13.12、14.9、15.4 及更早版本中的内存管理功能，可防止因可用内存不足而导致的稳定性问题和数据库重启。有关更多信息，请参阅 改进了 Aurora PostgreSQL 中的内存管理。	False

参数名称	描述	默认
rds.pg_stat_ramdisk_size	统计数据虚拟磁盘的大小（以 MB 为单位）。将以非零值设置虚拟磁盘。	0
rds.rds_superuser_reserved_connections	设置为 rds_superuser 保留的连接插槽数。此参数仅在版本 15 及更高版本中提供。有关更多信息，请参阅 PostgreSQL 文档预留连接。	2
rds.superuser_variables	提升 rds_superuser 修改语句的仅限超级用户的变量列表。	session_replication_role
remove_temp_files_after_crash	后端崩溃后删除临时文件。	0
restart_after_crash	后端崩溃后重新初始化服务器。	–
row_security	启用行安全。	–
search_path	设置针对非架构限定名称的架构搜索顺序。	–
seq_page_cost	设置计划程序预估连续提取磁盘页面的开销。	–
session_replication_role	设置触发器和重写规则的会话行为。	–
shared_buffers	(8 kB) 设置服务器使用的共享内存缓冲区数。	SUM(DBInstanceClassMemory/12038,-50003)
shared_preload_libraries	列出要预加载到服务器的共享库。	pg_stat_statements
ssl_ca_file	SSL 服务器授权文件的位置。	/rdsdbdata/rds-metadata/ca-cert.pem
ssl_cert_file	SSL 服务器证书文件的位置。	/rdsdbdata/rds-metadata/server-cert.pem

参数名称	描述	默认
ssl_crl_dir	SSL 证书吊销列表目录的位置。	/rdsdbdata/rds-metadata/ssl_crl_dir/
ssl_key_file	SSL 服务器私有密钥文件的位置	/rdsdbdata/rds-metadata/server-key.pem
standard_conforming_strings	导致 ... 字符串按字面处理反斜杠。	–
statement_timeout	设置任何语句允许的最长持续时间（ms）。	–
stats_temp_directory	将临时统计数据文件写入指定目录。	/rdsdbdata/db/pg_stat_tmp
superuser_reserved_connections	设置为超级用户预留的连接插槽数。	3
synchronize_seqscans	启用同步顺序扫描。	–
tcp_keepalives_count	重新传输 TCP 保持连接信号的最大次数。	–
tcp_keepalives_idle	发出两次 TCP 保持连接信号之间的时间（s）。	–
tcp_keepalives_interval	两次 TCP 保持连接信号重新传输之间的时间（s）。	–
temp_buffers	设置每个会话使用的临时缓冲区的最大数量（8kB）。	–
temp_file_limit	限制给定 PostgreSQL 进程可用于临时文件的总磁盘空间（以 kB 为单位），不包括用于显式临时表的空间	–1
temp_tablespaces	选择用于临时表和排序文件的表空间。	–
track_activities	收集有关执行命令的信息。	–

参数名称	描述	默认
track_activity_query_size	设置为 pg_stat_activity.current_query 保留的大小，以字节计。	4096
track_counts	收集有关数据库活动的统计数据。	–
track_functions	收集有关数据库活动的函数级别统计数据。	pl
track_io_timing	收集有关数据库 IO 活动的时序统计数据。	1
transform_–_equals	将 expr=– 视为 expr IS –。	–
update_process_title	更新进程标题以显示活动的 SQL 命令。	–
wal_receiver_status_interval	设置向主服务器报告 WAL 接收方状态之间的最大间隔（s）。	–
work_mem	设置要用于查询工作区的最大内存（kB）。	–
xmlbinary	设置如何将二进制值编码到 XML 中。	–
xmloption	设置要将隐式分析和序列化操作中的 XML 数据视为文档还是内容片段。	–

Amazon Aurora PostgreSQL 等待事件

以下是 Aurora PostgreSQL 的常见等待事件。要了解有关等待事件和调整 Aurora PostgreSQL 数据库集群的更多信息，请参阅使用 Aurora PostgreSQL 的等待事件进行优化。

Activity:ArchiverMain

存档程序正在等待活动。

Activity:AutoVacuumMain

Autovacuum 启动程序进程正在等待活动。

Activity:BgWriterHibernate

后台写入进程在等待活动时进入休眠状态。

Activity:BgWriterMain

后台写入器进程正在等待活动。

Activity:CheckpointerMain

检查指针进程正在等待活动。

Activity:LogicalApplyMain

逻辑复制应用进程正在等待活动。

Activity:LogicalLauncherMain

逻辑复制启动器进程正在等待活动。

Activity:PgStatMain

统计数据收集器进程正在等待活动。

Activity:RecoveryWalAll

进程正在等待恢复时来自流的预写日志 (WAL)。

Activity:RecoveryWalStream

启动进程正在等待流式恢复期间到达的预写日志 (WAL)。

Activity:SysLoggerMain

syslogger 过程正在等待活动。

Activity:WalReceiverMain

预写日志 (WAL) 接收方进程正在等待活动。

Activity:WalSenderMain

预写日志 (WAL) 发件人进程正在等待活动。

Activity:WalWriterMain

预写日志 (WAL) 写入器进程正在等待活动。

BufferPin:BufferPin

进程正在等待在缓冲区上获取专属 PIN。

Client:GSSOpenServer

进程正在等待从客户端读取数据，同时建立通用安全服务应用程序接口 (GSSAPI) 会话。

Client:ClientRead

后端进程正在等待从 PostgreSQL 客户端接收数据。有关更多信息，请参阅 Client:ClientRead。

Client:ClientWrite

后端进程正在等待将更多数据发送到 PostgreSQL 客户端。有关更多信息，请参阅 Client:ClientWrite。

Client:LibPQWalReceiverConnect

进程正在等待预写日志 (WAL) 接收方与远程服务器建立连接。

Client:LibPQWalReceiverReceive

进程在预写日志 (WAL) 接收方等待从远程服务器接收数据。

Client:SSLOpenServer

进程正在等待尝试连接时的安全套接字层 (SSL)。

Client:WalReceiverWaitStart

进程正在等待启动过程发送初始数据以进行流式复制。

Client:WalSenderWaitForWAL

进程正等待在 WAL 发件人进程中刷新预写日志 (WAL)。

Client:WalSenderWriteData

进程正在等待 WAL 发件人进程中处理来自预写日志 (WAL) 接收方的回复时的任何活动。

CPU

后端进程处于活动状态或正在等待 CPU。有关更多信息，请参阅 CPU。

Extension:extension

后端进程正在等待扩展或模块定义的条件。

IO:AuroraEnhancedLogicalWALRead

一个后端进程正在从更改数据捕获（CDC）卷获取日志记录。

IO:AuroraOptimizedReadsCacheRead

进程正在等待从优化型读取分层缓存中读取数据，因为页面在共享内存中不可用。

IO:AuroraOptimizedReadsCacheSegmentTruncate

进程正在等待优化型分层缓存段文件被截断。

IO:AuroraOptimizedReadsCacheWrite

后台写入器进程正在等待在优化型读取分层缓存中进行写入。

IO:AuroraStorageLogAllocate

会话正在分配元数据并准备写入事务日志。

IO:BufFileRead

当操作需要的内存超过工作内存参数定义的数量时，引擎将在磁盘上创建临时文件。当操作从临时文件中读取时，会发生此等待事件。有关更多信息，请参阅 IO:BufFileRead 和 IO:BufFileWrite。

IO:BufFileWrite

当操作需要的内存超过工作内存参数定义的数量时，引擎将在磁盘上创建临时文件。当操作写入临时文件时，会发生此等待事件。有关更多信息，请参阅 IO:BufFileRead 和 IO:BufFileWrite。

IO:ControlFileRead

进程正在等待从 `pg_control` 文件中进行读取。

IO:ControlFileSync

进程正在等待 `pg_control` 文件到达持久存储空间。

IO:ControlFileSyncUpdate

进程正在等待 `pg_control` 文件更新以到达持久存储空间。

IO:ControlFileWrite

进程正在等待写入 `pg_control` 文件。

IO:ControlFileWriteUpdate

进程正在等待写入以更新 `pg_control` 文件。

IO:CopyFileRead

进程正在等待文件复制操作期间进行读取。

IO:CopyFileWrite

在文件复制操作期间，进程正在等待写入。

IO:DataFileExtend

进程正在等待扩展关系数据文件。

IO:DataFileFlush

进程正在等待关系数据文件到达持久存储空间。

IO:DataFileImmediateSync

进程正在等待关系数据文件被立即同步到持久存储空间。

IO:DataFilePrefetch

进程正在等待从关系数据文件中进行异步预提取。

IO:DataFileSync

进程正在等待关系数据文件的更改到达持久存储空间。

IO:DataFileRead

后端进程试图在共享缓冲区中找到一个页面，但没有找到该页面，因此从存储中读取它。有关更多信息，请参阅 IO:DataFileRead。

IO:DataFileTruncate

进程正在等待关系数据文件被截断。

IO:DataFileWrite

进程正在等待写入关系数据文件。

IO:DSMFillZeroWrite

进程正在等待向动态共享内存备份文件写入零字节。

IO:LockFileAddToDataDirRead

进程正在等待向数据目录锁定文件添加行时进行读取。

IO:LockFileAddToDataDirSync

在向数据目录锁定文件添加行时，进程正在等待数据到达持久存储空间。

IO:LockFileAddToDataDirWrite

在向数据目录锁定文件添加行时，进程正在等待写入。

IO:LockFileCreateRead

在创建数据目录锁定文件时，进程正在等待读取。

IO:LockFileCreateSync

在创建数据目录锁定文件时，进程正在等待数据到达持久存储空间。

IO:LockFileCreateWrite

在创建数据目录锁定文件时，进程正在等待写入。

IO:LockFileReCheckDataDirRead

进程正在等待重新检查数据目录锁定文件期间进行读取。

IO:LogicalRewriteCheckpointSync

进程正在等待逻辑重写映射在检查点期间到达持久存储空间。

IO:LogicalRewriteMappingSync

进程正在等待映射数据在逻辑重写期间到达持久存储空间。

IO:LogicalRewriteMappingWrite

进程正在等待逻辑重写期间写入映射数据。

IO:LogicalRewriteSync

进程正在等待逻辑重写映射到达持久存储空间。

IO:LogicalRewriteTruncate

在逻辑重写期间，进程正在等待映射数据的截断。

IO:LogicalRewriteWrite

进程正在等待逻辑重写映射的写入。

IO:RelationMapRead

进程正在等待关系映射文件的读取。

IO:RelationMapSync

进程正在等待关系映射文件到达持久存储空间。

IO:RelationMapWrite

进程正在等待写入关系映射文件。

IO:ReorderBufferRead

进程正在重新排序缓冲区管理期间等待读取。

IO:ReorderBufferWrite

进程正在重新排序缓冲区管理期间等待写入。

IO:ReorderLogicalMappingRead

进程正在重新排序缓冲区管理期间等待逻辑映射读取。

IO:ReplicationSlotRead

进程正在等待从复制插槽控制文件中读取。

IO:ReplicationSlotRestoreSync

进程正在等待复制插槽控制文件在将其恢复到内存时到达持久存储空间。

IO:ReplicationSlotSync

进程正在等待复制插槽控制文件到达持久存储空间。

IO:ReplicationSlotWrite

进程正在等待写入复制插槽控制文件中。

IO:SLRUFlushSync

进程正在等待检查点或数据库关闭期间最近使用最少的（SLRU）的简单数据到达持久存储空间。

IO:SLRURead

进程正在等待读取最近使用最少（SLRU）的简单页面。

IO:SLRUSync

进程正在等待写入页面后最近最少使用（SLRU）的简单数据到达持久存储空间。

IO:SLRUWrite

进程正在等待写入最近使用最少（SLRU）的简单页面。

IO:SnapbuildRead

进程正在等待读取序列化的历史目录快照。

IO:SnapbuildSync

进程正在等待序列化的历史目录快照到达持久存储空间。

IO:SnapbuildWrite

进程正在等待写入序列化的历史目录快照。

IO:TimelineHistoryFileSync

进程正在等待通过流式复制接收的时间线历史记录文件到达持久存储空间。

IO:TimelineHistoryFileWrite

进程正在等待写入通过流式复制接收的时间表历史记录文件。

IO:TimelineHistoryRead

进程正在等待读取时间线历史记录文件。

IO:TimelineHistorySync

进程正在等待新创建的时间线历史记录文件到达持久存储空间。

IO:TimelineHistoryWrite

进程正在等待写入新创建的时间线历史记录文件。

IO:TwophaseFileRead

进程正在等待读取两阶段状态文件。

IO:TwophaseFileSync

进程正在等待两阶段状态文件到达持久存储空间。

IO:TwophaseFileWrite

进程正在等待写入两阶段状态文件。

IO:WALBootstrapSync

进程正在等待引导启动过程中到达持久存储空间的预写日志 (WAL)。

IO:WALBootstrapWrite

进程正在等待引导启动过程中写入预写日志 (WAL) 页面。

IO:WALCopyRead

进程正在等待通过复制现有预写日志段创建新的预写日志 (WAL) 段时进行读取。

IO:WALCopySync

进程正等待通过复制现有预写日志段创建的新预写日志 (WAL) 段到达持久存储空间。

IO:WALCopyWrite

通过复制现有预写日志段创建新的预写日志 (WAL) 段时，进程正在等待写入。

IO:WALInitSync

进程正在等待新初始化的预写日志 (WAL) 文件到达持久存储空间。

IO:WALInitWrite

进程正在等待新初始化预写日志 (WAL) 文件时进行写入。

IO:WALRead

进程正在等待读取预写日志 (WAL) 文件。

IO:WALSenderTimelineHistoryRead

进程正在等待从 WAL 发件人时间线命令期间的时间线历史记录文件进行读取。

IO:WALSync

进程正在等待预写日志 (WAL) 文件到达持久存储空间。

IO:WALSyncMethodAssign

进程在分配新的预写日志 (WAL) 同步方法时等待数据到达持久存储空间。

IO:WALWrite

进程正等待写入预写日志 (WAL) 文件。

IO:XactSync

后端进程正在等待 Aurora 存储子系统确认常规事务的提交，或者准备好的事务的提交或回滚。有关更多信息，请参阅 IO:XactSync。

IPC:AuroraLogicalSchemaUpdate

两个后端进程正在尝试将同一条目插入架构缓存。一个进程将继续，而另一个进程则等待其完成。

IPC:AuroraOptimizedReadsCacheWriteStop

进程正在等待后台写入器停止写入优化型读取分层缓存。

IPC:BackupWaitWalArchive

进程正在等待成功存档备份所需的预写日志 (WAL) 文件。

IPC:BgWorkerShutdown

进程正在等待后台工件关闭。

IPC:BgWorkerStartup

进程正在等待后台工件开启。

IPC:BtreePage

进程正在等待继续并行 B 树扫描所需的页码变为可用。

IPC:CheckpointDone

进程正在等待检查点完成。

IPC:CheckpointStart

进程正在等待检查点开始。

IPC:ClogGroupUpdate

进程正在等待组领导在事务结束时更新事务状态。

IPC:DamRecordTxAck

后端进程已生成数据库活动流事件，并且在等待该事件变为持久事件。有关更多信息，请参阅 IPC:DamRecordTxAck。

IPC:ExecuteGather

进程正在等待执行 Gather 计划节点时子进程中的活动。

IPC:Hash/Batch/Allocating

进程正在等待所选的并行哈希参与者分配哈希表。

IPC:Hash/Batch/Electing

进程正在选择并行哈希参与者以分配哈希表。

IPC:Hash/Batch/Loading

进程正在等待其他并行哈希参与者完成哈希表的加载。

IPC:Hash/Build/Allocating

进程正在等待所选的并行哈希参与者分配初始哈希表。

IPC:Hash/Build/Electing

进程正在选择并行哈希参与者以分配初始哈希表。

IPC:Hash/Build/HashingInner

进程正在等待其他并行哈希参与者完成内部关系的哈希。

IPC:Hash/Build/HashingOuter

进程正在等待其他并行哈希参与者完成外部关系的分区。

IPC:Hash/GrowBatches/Allocating

进程正在等待所选的并行哈希参与者分配更多批处理。

IPC:Hash/GrowBatches/Deciding

进程正在选择平行哈希参与者来决定未来的批处理增长。

IPC:Hash/GrowBatches/Electing

进程正在选择并行哈希参与者分配更多批处理。

IPC:Hash/GrowBatches/Finishing

进程正在等待所选的平行哈希参与者来决定未来的批处理增长。

IPC:Hash/GrowBatches/Repartitioning

进程正在等待其他并行哈希参与者完成重新分区。

IPC:Hash/GrowBuckets/Allocating

进程正在等待所选的并行哈希参与者完成更多存储桶的分配。

IPC:Hash/GrowBuckets/Electing

进程正在选择并行哈希参与者分配更多存储桶。

IPC:Hash/GrowBuckets/Reinserting

进程正在等待其他并行哈希参与者完成将元组插入到新存储桶的过程。

IPC:HashBatchAllocate

进程正在等待所选的并行哈希参与者分配哈希表。

IPC:HashBatchElect

进程正在等待选择并行哈希参与者以分配哈希表。

IPC:HashBatchLoad

进程正在等待其他并行哈希参与者完成哈希表的加载。

IPC:HashBuildAllocate

进程正在等待所选的并行哈希参与者分配初始哈希表。

IPC:HashBuildElect

进程正在等待选择并行哈希参与者分配初始哈希表。

IPC:HashBuildHashInner

进程正在等待其他并行哈希参与者完成内部关系的哈希。

IPC:'HashBuildHashOuter

进程正在等待其他并行哈希参与者完成外部关系的分区。

IPC:HashGrowBatchesAllocate

进程正在等待所选的并行哈希参与者分配更多批处理。

IPC:'HashGrowBatchesDecide

进程正在等待选择平行哈希参与者来决定未来的批处理增长。

IPC:HashGrowBatchesElect

进程正在等待选择并行哈希参与者分配更多批处理。

IPC:HashGrowBatchesFinish

进程正在等待所选的平行哈希参与者来决定未来的批处理增长。

IPC:HashGrowBatchesRepartition

进程正在等待其他并行哈希参与者完成重新分区。

IPC:HashGrowBucketsAllocate

进程正在等待所选的并行哈希参与者完成更多存储桶的分配。

IPC:HashGrowBucketsElect

进程正在等待选择并行哈希参与者分配更多存储桶。

IPC:HashGrowBucketsReinsert

进程正在等待其他并行哈希参与者完成将元组插入到新存储桶的过程。

IPC:LogicalSyncData

进程正在等待逻辑复制远程服务器发送初始表同步的数据。

IPC:LogicalSyncStateChange

进程正在等待逻辑复制远程服务器更改状态。

IPC:MessageQueueInternal

进程正在等待另进程附加到共享消息队列。

IPC:MessageQueuePutMessage

进程正在等待将协议消息写入共享消息队列。

IPC:MessageQueueReceive

进程正在等待从共享消息队列接收字节。

IPC:MessageQueueSend

进程正在等待将字节发送到共享消息队列。

IPC:ParallelBitmapScan

进程正在等待并行位图扫描初始化。

IPC:ParallelCreateIndexScan

进程正在等待并行 CREATE INDEX 工件完成堆扫描。

IPC:ParallelFinish

进程正在等待并行工件完成计算。

IPC:ProcArrayGroupUpdate

进程正在等待组领导在事务结束时清除事务 ID。

IPC:ProcSignalBarrier

进程正在等待所有后端处理障碍事件。

IPC:Promote

进程正在等待待机提升。

IPC:RecoveryConflictSnapshot

进程正在等待 vacuum 清理的恢复冲突解决方法。

IPC:RecoveryConflictTablespace

进程正在等待删除表空间的恢复冲突解决方法。

IPC:RecoveryPause

进程正在等待恢复。

IPC:ReplicationOriginDrop

进程正在等待复制源变为非活动状态，以便可以将其删除。

IPC:ReplicationSlotDrop

进程正在等待复制插槽变为非活动状态，以便可以将其删除。

IPC:SafeSnapshot

进程正在等待获取 READ ONLY DEFERRABLE 事务的有效快照。

IPC:SyncRep

进程正在同步复制期间等待来自远程服务器的确认。

IPC:XactGroupUpdate

进程正在等待组领导在事务结束时更新事务状态。

Lock:advisory

后端进程请求了一个咨询锁定并在等待它。有关更多信息，请参阅 Lock:advisory。

Lock:extend

后端进程正在等待锁被释放，以便可以扩展关系。需要此锁，因为一次只有一个后端进程可以扩展关系。有关更多信息，请参阅 Lock:extend。

Lock:frozenid

进程正在等待更新 `pg_database.datfrozenxid` 和 `pg_database.datminmxid`。

Lock:object

进程正在等待获取非关联数据库对象的锁定。

Lock:page

进程正在等待关系页面上的锁定。

Lock:Relation

后端进程正等待获取被另个事务锁定的关系上的锁定。有关更多信息，请参阅 Lock:Relation。

Lock:spectoken

进程正在等待获取推测性插入锁。

Lock:speculative token

进程正在等待获取推测性插入锁。

Lock:transactionid

一个事务正在等待行级锁定。有关更多信息，请参阅 Lock:transactionid。

Lock:tuple

后端进程正在等待获取元组上的锁定，而另一个后端进程在同一个元组上保持冲突锁定。有关更多信息，请参阅 Lock:tuple。

Lock:userlock

进程正在等待获取用户锁定。

Lock:virtualxid

进程正在等待获取虚拟事务 ID 锁定。

LWLock:AddinShmemInit

进程正在等待管理扩展在共享内存中的空间分配。

LWLock:AddinShmemInitLock

进程正在等待管理共享内存中的空间分配。

LWLock:async

进程正在等待异步（通知）缓冲区上的输入/输出。

LWLock:AsyncCtlLock

进程正在等待读取或更新共享通知状态。

LWLock:AsyncQueueLock

进程正在等待读取或更新通知消息。

LWLock:AuroraOptimizedReadsCacheMapping

进程正在等待将数据块与优化型读取分层缓存中的页面关联起来。

LWLock:AutoFile

进程正在等待更新 `postgresql.auto.conf` 文件。

LWLock:AutoFileLock

进程正在等待更新 `postgresql.auto.conf` 文件。

LWLock:Autovacuum

进程正在等待读取或更新 Autovacuum 工件的当前状态。

LWLock:AutovacuumLock

一个 Autovacuum 工件或启动程序正在等待更新或读取 Autovacuum 工件的当前状态。

LWLock:AutovacuumSchedule

进程正在等待确保选择用于 Autovacuum 的表仍然需要进行 vacuum 操作。

LWLock:AutovacuumScheduleLock

进程正在等待确保选择用于 vacuum 的表仍然需要进行 vacuum 操作。

LWLock:BackendRandomLock

进程正在等待生成一个随机数。

LWLock:BackgroundWorker

进程正在等待读取或更新后台工件状态。

LWLock:BackgroundWorkerLock

进程正在等待读取或更新后台工件状态。

LWLock:BtreeVacuum

进程正在等待读取或更新 B 树索引的与 vacuum 相关的信息。

LWLock:BtreeVacuumLock

进程正在等待读取或更新 B 树索引的与 vacuum 相关的信息。

LWLock:buffer_content

后端进程正在等待获取对共享内存缓冲区内容的轻量级锁定。有关更多信息，请参阅 LWLock:buffer_content (BufferContent)。

LWLock:buffer_mapping

后端进程正在等待将数据块与共享缓冲池中的缓冲区关联起来。有关更多信息，请参阅 LWLock:buffer_mapping。

LWLock:BufferIO

后端进程想要将页面读取到共享内存中。该进程正在等待其他进程完成页面的输入/输出操作。有关更多信息，请参阅 LWLock:BufferIO (IPC:BufferIO)。

LWLock:Checkpoint

进程正在等待开始检查点。

LWLock:CheckpointLock

进程正在等待执行检查点。

LWLock:CheckpointerComm

进程正在等待管理 `fsync` 请求。

LWLock:CheckpointerCommLock

进程正在等待管理 `fsync` 请求。

LWLock:clog

进程正在等待阻塞（事务状态）缓冲区上的输入/输出。

LWLock:CLogControlLock

进程正在等待读取或更新事务状态。

LWLock:CLogTruncationLock

进程正在等待运行 `txid_status` 或者更新可用的最早的事务 ID。

LWLock:commit_timestamp

进程正在等待提交时间戳缓冲区上的输入/输出。

LWLock:CommitTs

进程正在等待读取或更新为事务提交时间戳设置的最后一个值。

LWLock:CommitTsBuffer

进程正在等待最近使用最少的（SLRU）简单缓冲区上的输入/输出以获得提交时间戳。

LWLock:CommitTsControlLock

进程正在等待读取或更新事务提交时间戳。

LWLock:CommitTsLock

进程正在等待读取或更新为事务时间戳设置的最后一个值。

LWLock:CommitTsSLRU

进程正在等待访问最近使用最少的（SLRU）简单缓存以获取提交时间戳。

LWLock:ControlFile

进程正在等待读取或更新 `pg_control` 文件或创建新的预写日志 (WAL) 文件。

LWLock:ControlFileLock

进程正在等待读取或更新控制文件或创建新的预写日志 (WAL) 文件。

LWLock:DynamicSharedMemoryControl

进程正在等待读取或更新动态共享内存分配信息。

LWLock:DynamicSharedMemoryControlLock

进程正在等待读取或更新动态共享内存状态。

LWLock:lock_manager

后端进程正在等待添加或检查后端进程的锁定。或者它正在等待加入或退出由并行查询使用的锁定组。有关更多信息，请参阅 LWLock:lock_manager。

LWLock:LockFastPath

进程正在等待读取或更新进程的快速路径锁定信息。

LWLock:LogicalRepWorker

进程正在等待读取或更新逻辑复制工件的状态。

LWLock:LogicalRepWorkerLock

进程正在等待对逻辑复制工件的操作完成。

LWLock:LogicalSchemaCache

一个进程修改了架构缓存。

LWLock:multixact_member

进程正在等待 multixact_member 缓冲区上的输入/输出。

LWLock:multixact_offset

进程正在等待 multixact offset 缓冲区上的输入/输出。

LWLock:MultiXactGen

进程正在等待读取或更新共享 multixact 状态。

LWLock:MultiXactGenLock

进程正在等待读取或更新共享 multixact 状态。

LWLock:MultiXactMemberBuffer

进程正在等待 multixact 成员的最近使用最少的（SLRU）简单缓冲区上的输入/输出。有关更多信息，请参阅 LWLock:MultiXact。

LWLock:MultiXactMemberControlLock

进程正在等待读取或更新 multixact 成员映射。

LWLock:MultiXactMemberSLRU

进程正在等待访问 multixact 成员的最近使用最少的（SLRU）简单缓存。有关更多信息，请参阅 LWLock:MultiXact。

LWLock:MultiXactOffsetBuffer

进程正在等待 multixact offset 的最近使用最少的（SLRU）简单缓冲区上的输入/输出。有关更多信息，请参阅 LWLock:MultiXact。

LWLock:MultiXactOffsetControlLock

进程正在等待读取或更新 multixact offset 映射。

LWLock:MultiXactOffsetSLRU

进程正在等待访问 multixact offset 的最近使用最少的（SLRU）简单缓存。有关更多信息，请参阅 LWLock:MultiXact。

LWLock:MultiXactTruncation

进程正在等待读取或截断 multixact 信息。

LWLock:MultiXactTruncationLock

进程正在等待读取或截断 multixact 信息。

LWLock:NotifyBuffer

进程正在等待 NOTIFY 消息的最近使用最少的（SLRU）简单缓冲区上的输入/输出。

LWLock:NotifyQueue

进程正在等待读取或更新 NOTIFY 消息。

LWLock:NotifyQueueTail

进程正在等待更新 NOTIFY 消息存储的限制。

LWLock:NotifyQueueTailLock

进程正在等待更新通知消息存储的限制。

LWLock:NotifySLRU

进程正在等待访问 NOTIFY 消息的最近使用最少的（SLRU）简单缓存。

LWLock:OidGen

进程正在等待分配新的对象 ID (OID)。

LWLock:OidGenLock

进程正在等待分配或指派对象 ID (OID)。

LWLock:oldserxid

进程正在等待 oldserxid 缓冲区上的输入/输出。

LWLock:OldSerXidLock

进程正在等待读取或记录冲突的可序列化事务。

LWLock:OldSnapshotTimeMap

进程正在等待读取或更新旧的快照控制信息。

LWLock:OldSnapshotTimeMapLock

进程正在等待读取或更新旧的快照控制信息。

LWLock:parallel_append

进程在并行追加计划执行期间正在等待选择下一个子计划。

LWLock:parallel_hash_join

进程在并行哈希计划执行期间正在等待分配或交换一块内存或更新计数器。

LWLock:parallel_query_dsa

进程正在等待锁定并行查询的动态共享内存分配。

LWLock:ParallelAppend

进程在并行追加计划执行期间正在等待选择下一个子计划。

LWLock:ParallelHashJoin

进程在计划执行期间正在等待同步工件以进行并行哈希联接。

Lwlock:ParallelQueryDSA

进程正在等待并行查询的动态共享内存分配。

Lwlock:PerSessionDSA

进程正在等待并行查询的动态共享内存分配。

Lwlock:PerSessionRecordType

进程正在等待访问并行查询有关复合类型的信息。

Lwlock:PerSessionRecordTypmod

进程正在等待访问有关识别匿名记录类型的类型修饰符的并行查询的信息。

Lwlock:PerXactPredicateList

进程在并行查询期间正在等待访问当前可序列化事务所持有的谓词锁定的列表。

Lwlock:predicate_lock_manager

进程正在等待添加或检查谓词锁定信息。

Lwlock:PredicateLockManager

进程正在等待访问可序列化事务使用的谓词锁定信息。

Lwlock:proc

进程正在等待读取或更新快速路径锁定信息。

LWLock:ProcArray

进程正在等待访问共享的每个进程的数据结构（通常是获取快照或报告会话的事务 ID

LWLock:ProcArrayLock

进程正在等待获取快照或在事务结束时清除事务 ID。

LWLock:RelationMapping

进程正在等待读取或更新 pg_filenode.map 文件（用于跟踪某些系统目录的文件节点分配）。

LWLock:RelationMappingLock

进程正在等待更新用于存储目录到文件节点映射的关系映射文件。

LWLock:RelCacheInit

进程正在等待读取或更新 pg_internal.init 文件（关系缓存初始化文件）。

LWLock:RelCacheInitLock

进程正在等待读取或写入关系缓存初始化文件。

LWLock:replication_origin

进程正在等待读取或更新复制进度。

LWLock:replication_slot_io

进程正在等待复制插槽上的输入/输出。

LWLock:ReplicationOrigin

进程正在等待创建、删除或使用复制源。

LWLock:ReplicationOriginLock

进程正在等待设置、删除或使用复制源。

LWLock:ReplicationOriginState

进程正在等待读取或更新一个复制源的进度。

LWLock:ReplicationSlotAllocation

进程正在等待分配或释放复制插槽。

LWLock:ReplicationSlotAllocationLock

进程正在等待分配或释放复制插槽。

LWLock:ReplicationSlotControl

进程正在等待读取或更新复制插槽状态。

LWLock:ReplicationSlotControlLock

进程正在等待读取或更新复制插槽状态。

LWLock:ReplicationSlotIO

进程正在等待复制插槽上的输入/输出。

LWLock:SerialBuffer

进程正在等待可序列化事务冲突的最近使用最少的（SLRU）简单缓冲区上的输入/输出。

LWLock:SerializableFinishedList

进程正在等待访问已完成的可序列化事务的列表。

LWLock:SerializableFinishedListLock

进程正在等待访问已完成的可序列化事务的列表。

LWLock:SerializablePredicateList

进程正在等待访问可序列化事务所持有的谓词锁定的列表。

LWLock:SerializablePredicateLockListLock

进程正在等待对可序列化事务持有的锁定的列表执行操作。

LWLock:SerializableXactHash

进程正在等待读取或更新有关可序列化事务的信息。

LWLock:SerializableXactHashLock

进程正在等待读取或更新有关可序列化事务的信息。

LWLock:SerialSLRU

进程正在等待访问可序列化事务冲突的最近使用最少的（SLRU）简单缓存。

LWLock:SharedTidBitmap

进程在并行位图索引扫描期间正在等待访问共享元组标识符 (TID) 位图。

LWLock:SharedTupleStore

进程在并行查询期间正在等待访问共享元组存储。

LWLock:ShmemIndex

进程正在等待查找或分配共享内存中的空间。

LWLock:ShmemIndexLock

进程正在等待查找或分配共享内存中的空间。

LWLock:SInvalRead

进程正在等待从共享目录失效队列中检索消息。

LWLock:SInvalReadLock

进程正在等待从共享失效队列中检索或移除消息。

LWLock:SInvalWrite

进程正在等待向共享目录失效队列中添加消息。

LWLock:SInvalWriteLock

进程正在等待在共享失效队列中添加消息。

LWLock:subtrans

进程正在等待子事务缓冲区上的输入/输出。

LWLock:SubtransBuffer

进程正在等待子事务的最近使用最少的（SLRU）简单缓冲区上的输入/输出。

LWLock:SubtransControlLock

进程正在等待读取或更新子事务信息。

LWLock:SubtransSLRU

进程正在等待访问子事务的最近使用最少的（SLRU）简单缓存。

LWLock:SyncRep

进程正在等待读取或更新有关同步复制状态的信息。

LWLock:SyncRepLock

进程正在等待读取或更新有关同步副本的信息。

LWLock:SyncScan

进程正在等待选择同步表扫描的起始位置。

LWLock:SyncScanLock

进程正在等待获取一个扫描在同步扫描表上的起始位置。

LWLock:TablespaceCreate

进程正在等待创建或删除表空间。

LWLock:TablespaceCreateLock

进程正在等待创建或删除表空间。

LWLock:tbm

进程正在等待树位图 (TBM) 上的共享迭代器锁定。

LWLock:TwoPhaseState

进程正在等待读取或更新准备好的事务的状态。

LWLock:TwoPhaseStateLock

进程正在等待读取或更新准备好的事务的状态。

LWLock:wal_insert

进程正在等待将预写日志 (WAL) 插入内存缓冲区。

LWLock:WALBufMapping

进程正在等待替换预写日志 (WAL) 缓冲区中的页面。

LWLock:WALBufMappingLock

进程正在等待替换预写日志 (WAL) 缓冲区中的页面。

LWLock:WALInsert

进程正在等待将预写日志 (WAL) 数据插入内存缓冲区。

LWLock:WALWrite

进程正等待将预写日志 (WAL) 缓冲区写入磁盘。

LWLock:WALWriteLock

进程正等待将预写日志 (WAL) 缓冲区写入磁盘。

LWLock:WrapLimitsVacuum

进程正在等待更新事务 ID 和 multixact 消耗量的限制。

LWLock:WrapLimitsVacuumLock

进程正在等待更新事务 ID 和 multixact 消耗量的限制。

LWLock:XactBuffer

进程正在等待事务状态的最近使用最少的（SLRU）简单缓冲区上的输入/输出。

LWLock:XactSLRU

进程正在等待访问事务状态的最近使用最少的（SLRU）简单缓存。

LWLock:XactTruncation

进程正在等待运行 pg_xact_status 或者更新可用的最早的事务 ID。

LWLock:XidGen

进程正在等待分配新的事务 ID。

LWLock:XidGenLock

进程正在等待分配或指派事务 ID。

Timeout:BaseBackupThrottle

限制活动时，进程正在基本备份期间等待。

Timeout:PgSleep

后端进程已调用 pg_sleep 函数，正在等待睡眠超时过期。有关更多信息，请参阅 Timeout:PgSleep。

Timeout:RecoveryApplyDelay

由于延迟设置，进程正等待在恢复期间应用预写日志 (WAL)。

Timeout:RecoveryRetrieveRetryInterval

当没有来自任何来源（pg_wal、存档或流）的预写日志 (WAL) 数据时，进程在恢复过程中等待。

Timeout:VacuumDelay

进程正在基于成本的 vacuum 延迟点等待。

有关 PostgreSQL 等待事件的完整列表，请参阅 PostgreSQL 文档中的统计数据收集器 > 等待事件表。

Amazon Aurora PostgreSQL 的数据库引擎更新

在下文中，您可以了解有关 Amazon Aurora PostgreSQL 引擎版本和更新的信息。您还可以了解有关如何升级 Aurora PostgreSQL 引擎的信息。有关 Aurora 正式版本的更多信息，请参阅 Amazon Aurora 版本。

> **Tip**
>
> 您可以使用蓝绿部署，显著减少数据库集群升级所需的停机时间。有关更多信息，请参阅 使用蓝绿部署进行数据库更新。

主题

- 确定 Amazon Aurora PostgreSQL 版本
- Amazon Aurora PostgreSQL 版本和引擎版本
- Amazon Aurora PostgreSQL 的扩展版本
- 升级 Amazon Aurora PostgreSQL 数据库集群
- 使用 Aurora PostgreSQL 长期支持（LTS）版本

确定 Amazon Aurora PostgreSQL 版本

Amazon Aurora 包含某些 Aurora 通用功能，这些功能适用于所有 Aurora 数据库集群。Aurora 包含其自身支持的某个数据库引擎的其他特定功能。这些功能仅适用于使用该数据库引擎的 Aurora 数据库集群，例如 Aurora PostgreSQL。

Aurora 数据库版本具有两个版本号：数据库引擎版本号和 Aurora 版本号。如果 Aurora PostgreSQL 版本有 Aurora 版本号，则该版本号将包含在 Amazon Aurora PostgreSQL 版本和引擎版本 清单中，位于引擎版本号后。

主题

- Aurora 版本号
- PostgreSQL 引擎版本号

Aurora 版本号

Aurora 版本号使用 *major.minor.patch* 命名方案。Aurora 补丁版本包括在发布后添加到次要版本的重要错误修复。有关 Amazon Aurora 主要版本、次要版本和补丁版本的更多信息，请参阅 Amazon Aurora 主要版本、Amazon Aurora 次要版本 和 Amazon Aurora 补丁版本。

您可以使用以下 SQL 查询了解 Aurora PostgreSQL 数据库实例的 Aurora 版本号：

```
postgres=> SELECT aurora_version();
```

从发布的 PostgreSQL 版本 13.3、12.8、11.13、10.18 开始至这以后发布的更高版本，Aurora 版本号与 PostgreSQL 引擎版本更加一致。例如，查询 Aurora PostgreSQL 13.3 数据库集群将返回以下内容：

```
aurora_version
----------------
 13.3.1
(1 row)
```

之前的版本（例如 Aurora PostgreSQL 10.14 数据库集群）将返回类似于以下内容的版本号：

```
aurora_version
----------------
 2.7.3
(1 row)
```

PostgreSQL 引擎版本号

从 PostgreSQL 10 开始，PostgreSQL 数据库引擎的所有版本使用 *major.minor* 编号方案。示例包括 PostgreSQL 10.18、PostgreSQL 12.7 和 PostgreSQL 13.3。

PostgreSQL 10 之前的版本使用了 *major.major.minor* 编号方案，其中前两位数字组成主要版本号，第三位数表示次要版本。例如，PostgreSQL 9.6 是主要版本，次要版本为 9.6.21 或 9.6.22（通过第三位数表示）。

> **Note**
>
> 不再支持 PostgreSQL 引擎版本 9.6。要升级，请参阅 升级 Amazon Aurora PostgreSQL 数据库集群。有关版本策略和发布时间表，请参阅 Amazon Aurora 主要版本可用时间。

您可以通过以下 SQL 查询找到 PostgreSQL 数据库引擎版本号：

```
postgres=> SELECT version();
```

对于 Aurora PostgreSQL 13.3 数据库集群，结果如下：

```
version
-----------------------------------------------------------------------------------------------------
 PostgreSQL 13.3 on x86_64-pc-linux-gnu, compiled by x86_64-pc-linux-gnu-gcc (GCC)
 7.4.0, 64-bit
(1 row)
```

Amazon Aurora PostgreSQL 版本和引擎版本

Amazon Aurora PostgreSQL 兼容版定期更新。更新将在系统维护时段内应用于 Aurora PostgreSQL 数据库集群。何时应用更新取决于数据库集群的更新类型、AWS 区域和维护时段设置。列出的许多版本包括 PostgreSQL 版本号和 Amazon Aurora 版本号。不过，从发布的 PostgreSQL 版本 13.3、12.8、11.13、10.18 开始至这以后发布的更高版本，都不使用 Aurora 版本号。要确定 Aurora PostgreSQL 数据库的版本号，请参阅确定 Amazon Aurora PostgreSQL 版本。

有关扩展和模块的信息，请参阅 Amazon Aurora PostgreSQL 的扩展版本。

> **Note**
>
> 有关 Amazon Aurora 版本策略和发布时间表的信息，请参阅 Amazon Aurora 主要版本可用时间。

有关 Amazon Aurora 的支持的信息，请参阅 Amazon RDS 常见问题。

要确定 AWS 区域中有哪些 Aurora PostgreSQL 数据库引擎版本可用，请使用 AWS CLI 命令 describe-db-engine-versions。例如：

```
aws rds describe-db-engine-versions --engine aurora-postgresql --query '*[].
[EngineVersion]' --output text --region aws-region
```

有关 AWS 区域的列表，请参阅Aurora PostgreSQL 区域可用性。

有关 Aurora PostgreSQL 上可用的 PostgreSQL 版本的详细信息，请参阅 Aurora PostgreSQL 版本注释。

Amazon Aurora PostgreSQL 的扩展版本

您可以安装和配置各种 PostgreSQL 扩展以便与 Aurora PostgreSQL 数据库集群结合使用。例如，您可以使用 PostgreSQL `pg_partman` 扩展自动创建和维护表分区。要了解有关此扩展以及适用于 Aurora PostgreSQL 的其他扩展的更多信息，请参阅使用扩展和外部数据包装器。

有关 Aurora PostgreSQL 支持的 PostgreSQL 扩展的详细信息，请参阅《Aurora PostgreSQL 发布说明》中的 Amazon Aurora PostgreSQL 的扩展版本。

升级 Amazon Aurora PostgreSQL 数据库集群

仅当经过广泛的测试后，Amazon Aurora 才会在 AWS 区域中推出 PostgreSQL 数据库引擎的新版本。当您的区域中推出新版本时，您可以将 Aurora PostgreSQL 数据库集群升级到新版本。

根据数据库集群当前运行的 Aurora PostgreSQL 版本，升级到新版本可能是次要版本升级，也可能是主要版本升级。例如，将 Aurora PostgreSQL 11.15 数据库集群升级到 Aurora PostgreSQL 13.6 就是主要版本升级。将 Aurora PostgreSQL 13.3 数据库集群升级到 Aurora PostgreSQL 13.7 是次要版本升级。在以下主题中，您可以了解有关如何执行这两种升级类型的信息。

目录

- Aurora PostgreSQL 升级过程概述
- 获取您的 AWS 区域中可用版本的列表
- 执行主要版本升级
  - 测试将生产数据库集群升级到新的主要版本
  - 升级后建议
  - 将 Aurora PostgreSQL 引擎升级到新的主要版本
    - 全局数据库的主要版本升级
- 执行次要版本升级
  - 在执行次要版本升级之前
  - 如何执行次要版本升级和应用补丁
  - 次要版本升级和零停机时间修补
  - 零停机时间修补的限制
  - 将 Aurora PostgreSQL 引擎升级到新的次要版本
- 升级 PostgreSQL 扩展
- 备选的蓝绿升级技术

Aurora PostgreSQL 升级过程概述

主要版本升级和次要版本升级之间的区别如下：

次要版本升级和补丁

次要版本升级和补丁仅包含与现有应用程序向后兼容的更改。只有在经过 Aurora PostgreSQL 测试并批准后，您才可以使用次要版本升级和补丁。

Aurora 可以自动为您应用次要版本升级。在您创建新的 Aurora PostgreSQL 数据库集群时，默认情况下启用了启用次要版本升级选项。除非您手动关闭此选项，否则 Aurora 会在您计划的维护时段内定期应用自动次要版本升级。有关自动次要版本升级（AmVU）选项以及如何修改 Aurora 数据库集群以使用该选项的更多信息，请参阅 Aurora 数据库集群的自动次要版本升级。

如果没有为 Aurora PostgreSQL 数据库集群启用自动次要版本升级，则 Aurora PostgreSQL 不会自动升级到新的次要版本。相反，当您的 AWS 区域中发布了新的次要版本，但您的 Aurora PostgreSQL 数据库集群运行的是较旧的次要版本时，Aurora 会提示您进行升级。它通过向集群的维护任务添加建议来实现。

补丁不被视为升级，不会自动应用。Aurora PostgreSQL 通过向 Aurora PostgreSQL 数据库集群的维护任务添加建议，提示您应用任何补丁。有关更多信息，请参阅 如何执行次要版本升级和应用补丁。

> **Note**
>
> 解决安全或其他严重问题的补丁也会添加为维护任务。但是，这些补丁是必需的。确保当安全补丁在待定的维护任务中变为可用时，将它们应用于 Aurora PostgreSQL 数据库集群。

随着集群中的每个实例都升级到新版本，升级过程可能会导致短暂的中断。但是，在 Aurora PostgreSQL 版本 14.3.3、13.7.3、12.11.3、11.16.3、10.21.3 以及这些次要版本的其他更高版本和更高的主要版本之后，升级过程使用零停机修补（ZDP）特征。此特征可最大限度地减少中断，并在大多数情况下彻底消除中断。有关更多信息，请参阅 次要版本升级和零停机时间修补。有关 ZDP 支持的功能及相关限制的更多信息，请参阅零停机时间修补的限制。

主要版本升级

与次要版本升级和补丁不同，Aurora PostgreSQL 没有自动主要版本升级选项。新的主要 PostgreSQL 版本可能包含未与现有应用程序向后兼容的数据库更改。新功能会导致现有应用程序停止正常工作。

为了避免出现任何问题，我们强烈建议您在升级 Aurora PostgreSQL 数据库集群中的数据库实例之前，按照测试将生产数据库集群升级到新的主要版本中概括的过程操作。首先，请按照该过程确保您的应用程序可以在新版本上运行。然后，您可以手动将 Aurora PostgreSQL 数据库集群升级到新版本。

随着集群中的所有实例都升级到新版本，升级过程可能会导致短暂停机。初步计划过程也需要时间。我们建议您始终在集群的维护时段或操作极少时执行升级任务。有关更多信息，请参阅 执行主要版本升级。

> **Note**
>
> 次要版本升级和主要版本升级都可能会导致短暂的中断。因此，我们强烈建议您在维护时段或利用率低的其他时段执行或安排升级。

Aurora PostgreSQL 数据库集群偶尔需要操作系统更新。这些更新可能包含较新版本的 glibc 库。在此类更新期间，我们建议您遵循 Aurora PostgreSQL 中支持的排序规则中所述的指南。

获取您的 AWS 区域中可用版本的列表

通过查询 AWS 区域并使用 AWS CLI 命令 describe-db-engine-versions，您可以获得可用作 Aurora PostgreSQL 数据库集群的升级目标的所有引擎版本列表，如下所示。

对于 Linux、macOS 或 Unix：

```
aws rds describe-db-engine-versions \
  --engine aurora-postgresql \
  --engine-version version-number \
  --query 'DBEngineVersions[*].ValidUpgradeTarget[*].{EngineVersion:EngineVersion}' \
  --output text
```

对于 Windows：

```
aws rds describe-db-engine-versions ^
  --engine aurora-postgresql ^
  --engine-version version-number ^
  --query "DBEngineVersions[*].ValidUpgradeTarget[*].{EngineVersion:EngineVersion}" ^
  --output text
```

例如，要确定 Aurora PostgreSQL 版本 12.10 数据库集群的有效升级目标，请运行以下 AWS CLI 命令：

对于 Linux、macOS 或 Unix：

```
aws rds describe-db-engine-versions \
  --engine aurora-postgresql \
  --engine-version 12.10 \
  --query 'DBEngineVersions[*].ValidUpgradeTarget[*].{EngineVersion:EngineVersion}' \
  --output text
```

对于 Windows：

```
aws rds describe-db-engine-versions ^
  --engine aurora-postgresql ^
  --engine-version 12.10 ^
  --query "DBEngineVersions[*].ValidUpgradeTarget[*].{EngineVersion:EngineVersion}" ^
  --output text
```

在下表中，可以找到适用于不同 Aurora PostgreSQL 数据库版本的主要和次要版本升级目标。为了保持兼容性，并非所有版本都作为升级目标提供。Aurora PostgreSQL 在每个季度的次要版本发布中都引入了新功能和错误修复。有关 Aurora PostgreSQL 次要版本的信息，请参阅 Release Notes for Aurora PostgreSQL。

当前源版本	升级目标
17.5	无
17.4	17.5、17.4
16.9	17.5
16.8	17.5、17.4 16.9
16.6	17.5、17.4 16.9、16.8
16.4	17.5、17.4

当前源版本	升级目标
	16.9、16.8、16.6
16.3	17.5、17.4 16.9、16.8、16.6、16.4
16.2	17.5、17.4 16.9、16.8、16.6、16.4、16.3
16.1	17.5、17.4 16.9、16.8、16.6、16.4、16.3、16.2
15.13	17.5 16.9
15.12	17.5、17.4 16.9、16.8 15.13
15.10	17.5、17.4 16.9、16.8、16.6 15.13、15.12
15.8	17.5、17.4 16.9、16.8、16.6、16.4 15.13、15.12、15.10
15.7	17.5、17.4 16.9、16.8、16.6、16.4、16.3 15.13、15.12、15.10、15.8

当前源版本	升级目标
15.6	17.5、17.4 16.9、16.8、16.6、16.4、16.3、16.2 15.13、15.12、15.10、15.8、15.7
15.5	17.5、17.4 16.9、16.8、16.6、16.4、16.3、16.2、16.1 15.13、15.12、15.10、15.8、15.7、15.6
15.4	17.5、17.4 16.9、16.8、16.6、16.4、16.3、16.2、16.1 15.13、15.12、15.10、15.8、15.7、15.6、15.5
15.3	17.5、17.4 16.9、16.8、16.6、16.4、16.3、16.2、16.1 15.13、15.12、15.10、15.8、15.7、15.6、15.5、15.4
15.2	17.5、17.4 16.9、16.8、16.6、16.4、16.3、16.2、16.1 15.13、15.12、15.10、15.8、15.7、15.6、15.5、15.4、15.3
14.18	17.5 16.9 15.13

当前源版本	升级目标
14.17	17.5、17.4 16.9、16.8 15.13、15.12 14.18
14.15	17.5、17.4 16.9、16.8、16.6 15.13、15.12、15.10 14.18、14.17
14.13	17.5、17.4 16.9、16.8、16.6、16.4 15.13、15.12、15.10、15.8 14.18、14.17、14.15
14.12	17.5、17.4 16.9、16.8、16.6、16.4、16.3 15.13、15.12、15.10、15.8、15.7 14.18、14.17、14.15、14.13
14.11	17.5、17.4 16.9、16.8、16.6、16.4、16.3、16.2 15.13、15.12、15.10、15.8、15.6 14.18、14.17、14.15、14.13、14.12

当前源版本	升级目标
14.10	17.5、17.4 16.9、16.8、16.6、16.4、16.3、16.2、16.1 15.13、15.12、15.10、15.8、15.7、15.6、15.5 14.18、14.17、14.15、14.13、14.12、14.11
14.9	17.5、17.4 16.9、16.8、16.6、16.4、16.3、16.2、16.1 15.13、15.12、15.10、15.8、15.7、15.6、15.5、15.4 14.18、14.17、14.15、14.13、14.12、14.11、14.10
14.8	17.5、17.4 16.9、16.8、16.6、16.4、16.3、16.2、16.1 15.13、15.12、15.10、15.8、15.7、15.6、15.5、15.4、15.3、15.2 14.18、14.17、14.15、14.13、14.12、14.11、14.10、14.9
14.7	17.5、17.4 16.9、16.8、16.6、16.4、16.3、16.2、16.1 15.13、15.12、15.10、15.8、15.7、15.6、15.5、15.4、15.3、15.2 14.18、14.17、14.15、14.13、14.12、14.11、14.10、14.9、14.8
14.6	17.5、17.4 16.9、16.8、16.6、16.4、16.3、16.2、16.1 15.13、15.12、15.10、15.8、15.7、15.6、15.5、15.4、15.3、15.2 14.18、14.17、14.15、14.13、14.12、14.11、14.10、14.9、14.8、14.7

当前源版本	升级目标
14.5	17.5、17.4 16.9、16.8、16.6、16.4、16.3、16.2、16.1 15.13、15.12、15.10、15.8、15.7、15.6、15.5、15.4、15.3、15.2 14.18、14.17、14.15、14.13、14.12、14.11、14.10、14.9、14.8、14.7、14
14.4	17.5、17.4 16.9、16.8、16.6、16.4、16.3、16.2、16.1 15.13、15.12、15.10、15.8、15.7、15.6、15.5、15.4、15.3、15.2 14.18、14.17、14.15、14.13、14.12、14.11、14.10、14.9、14.8、14.7、14
14.3	17.5、17.4 16.9、16.8、16.6、16.4、16.3、16.2、16.1 15.13、15.12、15.10、15.8、15.7、15.6、15.5、15.4、15.3、15.2 14.18、14.17、14.15、14.13、14.12、14.11、14.10、14.9、14.8、14.7、14
13.21	17.5 16.9 15.13 14.18
13.20	17.5、17.4 16.9、16.8 15.13、15.12 14.18、14.17 13.21

当前源版本	升级目标
13.18	17.5、17.4 16.9、16.8、16.6 15.13、15.12、15.10 14.18、14.17、14.15 13.21、13.20
13.16	17.5、17.4 16.9、16.8、16.6、16.4 15.13、15.12、15.10、15.8 14.18、14.17、14.15、14.13 13.21、13.20、13.18
13.15	17.5、17.4 16.9、16.8、16.6、16.4、16.3 15.13、15.12、15.10、15.8、15.7 14.18、14.17、14.15、14.13、14.12 13.21、13.20、13.18、13.16
13.14	17.5、17.4 16.9、16.8、16.6、16.4、16.3、16.2 15.13、15.12、15.10、15.8、15.7、15.6 14.18、14.17、14.15、14.13、14.11 13.21、13.20、13.18、13.16、13.15

当前源版本	升级目标
13.13	17.5、17.4 16.9、16.8、16.6、16.4、16.3、16.2、16.1 15.13、15.12、15.10、15.8、15.7、15.6、15.5 14.18、14.17、14.15、14.13、14.12、14.11、14.10 13.21、13.20、13.18、13.16、13.15、13.14
13.12	17.5、17.4 16.9、16.8、16.6、16.4、16.3、16.2、16.1 15.13、15.12、15.10、15.8、15.7、15.6、15.5、15.4 14.18、14.17、14.15、14.13、14.12、14.11、14.10、14.9 13.21、13.20、13.18、13.16、13.15、13.14、13.13
13.11	17.5、17.4 16.9、16.8、16.6、16.4、16.3、16.2、16.1 15.13、15.12、15.10、15.8、15.7、15.6、15.5、15.4、15.3 14.18、14.17、14.15、14.13、14.12、14.11、14.10、14.9、14.8 13.21、13.20、13.18、13.16、13.15、13.14、13.13、13.12
13.10	17.5、17.4 16.9、16.8、16.6、16.4、16.3、16.2、16.1 15.13、15.12、15.10、15.8、15.7、15.6、15.5、15.4、15.3、15.2 14.18、14.17、14.15、14.13、14.12、14.11、14.10、14.9、14.8、14.7 13.21、13.20、13.18、13.16、13.15、13.14、13.13、13.12、13.11

当前源版本	升级目标
13.9	17.5、17.4 16.9、16.8、16.6、16.4、16.3、16.2、16.1 15.13、15.12、15.10、15.8、15.7、15.6、15.5、15.4、15.3、15.2 14.18、14.17、14.15、14.13、14.12、14.11、14.10、14.9、14.8、14.7、14 13.21、13.20、13.18、13.16、13.15、13.14、13.13,13.12、13.11、13.10
13.8	17.5、17.4 16.9、16.8、16.6、16.4、16.3、16.2、16.1 15.13、15.12、15.10、15.8、15.7、15.6、15.5、15.4、15.3、15.2 14.18、14.17、14.15、14.13、14.12、14.11、14.10、14.9、14.8、14.7、14 13.21、13.20、13.18、13.16、13.15、13.14、13.13、13.12、13.11、13.10、
13.7	17.5、17.4 16.9、16.8、16.6、16.4、16.3、16.2、16.1 15.13、15.12、15.10、15.8、15.7、15.6、15.5、15.4、15.3、15.2 14.18、14.17、14.15、14.13、14.12、14.11、14.10、14.9、14.8、14.7、14 13.21、13.20、13.18、13.16、13.15、13.14、13.13、13.12、13.11、13.10、
12.22	17.5、17.4 16.9、16.8、16.6 15.13、15.12、15.10 14.18、14.17、14.15 13.21、13.20、13.18

当前源版本	升级目标
12.20	17.5、17.4 16.9、16.8、16.6、16.4 15.13、15.12、15.10、15.8 14.15、14.13 13.21、13.20、13.18、13.16 12.22
12.19	17.5、17.4 16.9、16.8、16.6、16.4、16.3 15.13、15.12、15.10、15.8、15.7 14.18、14.17、14.15、14.13、14.12 13.21、13.20、13.18、13.16、13.15 12.22、12.20
12.18	17.5、17.4 16.9、16.8、16.6、16.4、16.3、16.2 15.13、15.12、15.10、15.8、15.7、15.6 14.18、14.17、14.15、14.13、14.12、14.11 13.21、13.20、13.18、13.16、13.15、13.14 12.22、12.20、12.19

当前源版本	升级目标
12.17	17.5、17.4 16.9、16.8、16.6、16.4、16.3、16.2、16.1 15.13、15.12、15.10、15.8、15.7、15.6、15.5 14.18、14.17、14.15、14.13、14.12、14.11、14.10 13.21、13.20、13.18、13.16、13.15、13.14、13.13 12.22、12.20、12.19、12.18
12.16	17.5、17.4 16.9、16.8、16.6、16.4、16.3、16.2、16.1 15.13、15.12、15.10、15.8、15.7、15.6、15.5、15.4 14.18、14.17、14.15、14.13、14.12、14.11、14.10、14.9 13.21、13.20、13.18、13.16、13.15、13.14、13.13、13.12 12.22、12.20、12.19、12.18、12.17
12.15	17.5、17.4 16.9、16.8、16.6、16.4、16.3、16.2、16.1 15.13、15.12、15.10、15.8、15.7、15.6、15.5、15.4、15.3 14.18、14.17、14.15、14.13、14.12、14.11、14.10、14.9、14.8 13.21、13.20、13.18、13.16、13.15、13.14、13.13、13.12、13.11 12.22、12.20、12.19、12.18、12.17、12.16

当前源版本	升级目标
12.14	17.5、17.4 16.9、16.8、16.6、16.4、16.3、16.2、16.1 15.13、15.12、15.10、15.8、15.7、15.6、15.5、15.5、15.4、15.3、15.2 14.18、14.17、14.15、14.13、14.12、14.11、14.10、14.9、14.8、14.7 13.21、13.20、13.18、13.16、13.15、13.14、13.13、13.12、13.11、13.10 12.22、12.20、12.19、12.18、12.17、12.16、12.15
12.13	17.5、17.4 16.9、16.8、16.6、16.4、16.3、16.2、16.1 15.13、15.12、15.10、15.8、15.7、15.6、15.5、15.4、15.3、15.2 14.18、14.17、14.15、14.13、14.12、14.11、14.10、14.9、14.8、14.7、14 13.21、13.20、13.18、13.16、13.15、13.14、13.13、13.12、13.11、13.10、 12.22、12.20、12.19、12.18、12.17、12.16、12.15、12.14
12.12	17.5、17.4 16.9、16.8、16.6、16.4、16.3、16.2、16.1 15.13、15.12、15.10、15.8、15.7、15.6、15.5、15.4、15.3、15.2 14.18、14.17、14.15、14.13、14.12、14.11、14.10、14.9、14.8、14.7、14 13.21、13.20、13.18、13.16、13.15、13.14、13.13、13.12、13.11、13.10、 12.22、12.20、12.19、12.18、12.17、12.16、12.15、12.14、12.13

当前源版本	升级目标
12.11	17.5、17.4 16.9、16.8、16.6、16.4、16.3、16.2、16.1 15.13、15.12、15.10、15.8、15.7、15.6、15.5、15.4、15.3、15.2 14.18、14.17、14.15、14.13、14.12、14.11、14.10、14.9、14.8、14.7、14 13.21、13.20、13.18、13.16、13.15、13.14、13.13、13.12、13.11、13.10、 12.22、12.20、12.19、12.18、12.17、12.16、12.15、12.14、12.13、12.12
12.9	17.5、17.4 16.9、16.8、16.6、16.4、16.3、16.2、16.1 15.13、15.12、15.10、15.8、15.7、15.6、15.5、15.4、15.3、15.2 14.18、14.17、14.15、14.13、14.12、14.11、14.10、14.9、14.8、14.7 13.21、13.20、13.18、13.16、13.14、13.13、13.12、13.11、13.10、13.9、 12.22、12.20、12.19、12.18、12.17、12.16、12.15、12.14、12.13、12.12、
11.21	17.5、17.4 16.9、16.8、16.6、16.4、16.3、16.2、16.1 15.13、15.12、15.10、15.8、15.7、15.6、15.5、15.4 14.18、14.17、14.15、14.13、14.12、14.11、14.10、14.9 13.21、13.20、13.18、13.16、13.15、13.14、13.13、13.12 12.22、12.20、12.19、12.18、12.17、12.16

当前源版本	升级目标
11.9	17.5、17.4 16.9、16.8、16.6、16.4、16.3、16.2、16.1 15.13、15.12、15.10、15.8、15.7、15.6、15.5、15.4、15.3、15.2 14.18、14.17、14.15、14.13、14.12、14.11、14.10、14.9、14.8、14.7、14 13.21、13.20、13.18、13.16、13.15、13.14、13.13、13.12、13.11、13.10、 12.22、12.20、12.19、12.18、12.17、12.16、12.15、12.14、12.13、12.12、 11.21

对于您正在考虑的任何版本，请始终检查集群数据库实例类的可用性。例如，Aurora PostgreSQL 13 不支持 db.r4。如果您的 Aurora PostgreSQL 数据库集群当前使用 db.r4 实例类，则在尝试升级之前需要移至 db.r5。有关数据库实例类的更多信息，包括哪些实例类基于 Graviton2 以及哪些实例基于英特尔，请参阅 Amazon Aurora 数据库实例类。

执行主要版本升级

主要版本升级可能包含不与数据库的以前版本向后兼容的数据库更改。新版本中的新功能会导致现有应用程序停止正常工作。为避免出现问题，Amazon Aurora 不会自动应用主要版本升级。相反，我们建议您按照以下步骤仔细规划主要版本升级：

1. 从表中为您的版本列出的可用目标列表中选择所需的主要版本。通过使用 AWS CLI，您可以获得您的 AWS 区域中针对当前版本推出的版本的精确列表。有关更多信息，请参阅 获取您的 AWS 区域中可用版本的列表。
2. 验证您的应用程序在新版本的试用部署中是否按预期工作。有关完整过程的信息，请参阅测试将生产数据库集群升级到新的主要版本。
3. 在验证应用程序在试用部署中按预期工作后，您可以升级集群。有关更多信息，请参阅 将 Aurora PostgreSQL 引擎升级到新的主要版本。

> **Note**
>
> 您可以执行主要版本升级，以从基于适用于 Aurora PostgreSQL 的 Babelfish 13 的版本（从 13.6 开始）升级到基于 Aurora PostgreSQL 14 的版本（从 14.6 开始）。适用于 Aurora PostgreSQL 的 Babelfish 13.4 和 13.5 不支持主要版本升级。

通过查询 AWS 区域并使用 AWS CLI 命令 describe-db-engine-versions，您可以获得可用作 Aurora PostgreSQL 数据库集群的主要版本升级目标的引擎版本列表，如下所示。

对于 Linux、macOS 或 Unix：

```
aws rds describe-db-engine-versions \
  --engine aurora-postgresql \
  --engine-version version-number \
  --query 'DBEngineVersions[].ValidUpgradeTarget[?IsMajorVersionUpgrade == `true`].
{EngineVersion:EngineVersion}' \
  --output text
```

对于 Windows：

```
aws rds describe-db-engine-versions ^
  --engine aurora-postgresql ^
  --engine-version version-number ^
  --query "DBEngineVersions[].ValidUpgradeTarget[?IsMajorVersionUpgrade == `true`].
{EngineVersion:EngineVersion}" ^
  --output text
```

在某些情况下，要升级到的版本不是当前版本的目标。在这种情况下，请使用versions table中的信息执行次要版本升级，直到您的集群处于其目标行中包含所选目标的版本。

测试将生产数据库集群升级到新的主要版本

每个新的主要版本都包括对查询优化程序的增强，旨在提高性能。但是，您的工作负载可能包括导致新版本中计划性能较差的查询。正因如此，我们建议您在生产环境中进行升级之前，先测试和审查性能。您可以使用查询计划管理 (QPM) 扩展来管理跨版本的查询计划稳定性，详情请参阅 确保主要版本升级后的计划稳定性。

在将生产 Aurora PostgreSQL 数据库集群升级到新的主要版本之前，我们强烈建议您测试升级，以验证所有应用程序是否正常工作：

1. 准备好一个版本兼容的参数组。

   如果您使用的是自定义数据库实例或数据库集群参数组，您可以从两个选项中进行选择：

   a. 为新的数据库引擎版本指定默认数据库实例和/或数据库集群参数组。

   b. 为新的数据库引擎版本创建您自己的自定义参数组。

   如果在升级请求中关联了新的数据库实例或数据库集群参数组，则确保在升级完成后重新启动数据库才能应用参数。如果需要重新启动数据库实例来应用参数组更改，则该实例的参数组状态将显示 pending-reboot。您可以在控制台中查看实例的参数组状态，也可以使用 CLI 命令（例如，describe-db-instances 或 describe-db-clusters）来查看。

2. 检查是否存在无效数据库，并删除所有存在的无效数据库。

   pg_database 目录中的 datconnlimit 列包含值 -2，用于将在 DROP DATABASE 操作期间中断的任何数据库标记为无效。使用以下查询来检查是否存在无效的数据库。

   ```
   SELECT
       datname
   FROM
       pg_database
   WHERE
       datconnlimit = - 2;
   ```

   如果查询返回数据库名称，则这些数据库无效。使用 DROP DATABASE invalid_db_name 语句删除无效的数据库。可以使用以下动态语句来删除所有无效数据库。

   ```
   SELECT
       'DROP DATABASE ' || quote_ident(datname) || ';'
   FROM
       pg_database
   WHERE
       datconnlimit = -2 \gexec
   ```

3. 检查是否有不支持的使用方式：

   - 在尝试升级前，提交或回滚所有打开的已准备事务。您可以使用以下查询确认您的实例上是否没有打开的已准备事务。

     ```
     SELECT count(*) FROM pg_catalog.pg_prepared_xacts;
     ```

- 在尝试升级前取消使用所有 reg* 数据类型。除了 `regtype` 和 `regclass` 以外，您不能升级 reg* 数据类型。pg_upgrade 实用工具（Amazon Aurora 用来执行升级）不能保留这个数据类型。要了解有关此实用程序的更多信息，请参阅 PostgreSQL 文档中的 pg_upgrade。

  要验证是否没有使用不支持的 reg* 数据类型，请对每个数据库使用以下查询。

```
SELECT count(*) FROM pg_catalog.pg_class c, pg_catalog.pg_namespace n,
 pg_catalog.pg_attribute a
  WHERE c.oid = a.attrelid
      AND NOT a.attisdropped
      AND a.atttypid IN ('pg_catalog.regproc'::pg_catalog.regtype,
                         'pg_catalog.regprocedure'::pg_catalog.regtype,
                         'pg_catalog.regoper'::pg_catalog.regtype,
                         'pg_catalog.regoperator'::pg_catalog.regtype,
                         'pg_catalog.regconfig'::pg_catalog.regtype,
                         'pg_catalog.regdictionary'::pg_catalog.regtype)
      AND c.relnamespace = n.oid
      AND n.nspname NOT IN ('pg_catalog', 'information_schema');
```

- 如果对已安装 `pgRouting` 扩展的 Aurora PostgreSQL 10.18 版本或更高版本进行升级，请删除此扩展，然后再升级到 12.4 或更高版本。

  如果您正在升级安装了扩展 `pg_repack` 版本 1.4.3 的 Aurora PostgreSQL 10.x 版本，请在升级到任何更高版本之前删除该扩展。

4. 检查 template1 和 template0 数据库。

   要成功升级，template1 和 template 0 数据库必须存在且应作为模板列出。要对此进行检查，请使用以下命令：

```
SELECT datname, datistemplate FROM pg_database;

datname    | datistemplate
-----------+---------------
template0  | t
rdsadmin   | f
template1  | t
postgres   | f
```

   在命令输出中，template1 和 template0 数据库的 `datistemplate` 值应为 `t`。

5. 删除逻辑复制槽。

如果 Aurora PostgreSQL 数据库集群正在使用任何逻辑复制槽，则升级过程无法继续。逻辑复制槽通常用于短期数据迁移任务，例如使用 AWS DMS 迁移数据或将表从数据库复制到数据湖、BI 工具或其他目标。升级之前，请确保您知道存在的任何逻辑复制槽的用途，并确认可以删除它们。您可以使用以下查询来检查逻辑复制槽：

```
SELECT * FROM pg_replication_slots;
```

如果逻辑复制槽仍在使用中，则不应将其删除，但也无法继续升级。但是，如果不需要逻辑复制槽，则可以使用以下 SQL 将其删除：

```
SELECT pg_drop_replication_slot(slot_name);
```

使用 pglogical 扩展的逻辑复制方案还需要从发布者节点上删除插槽，才能在该节点上成功升级主要版本。但是，可以在升级后从订阅者节点重新启动复制过程。有关更多信息，请参阅 在主要升级后重新建立逻辑复制。

6. 执行备份。

   在升级期间，升级进程创建数据库集群的数据库集群快照。如果您还希望在升级进程之前执行手动备份，有关更多信息，请参阅创建数据库集群快照。

7. 在执行主要版本升级之前，将某些扩展升级到最新的可用版本。要更新的扩展包括以下各项：

   - pgRouting
   - postgis_raster
   - postgis_tiger_geocoder
   - postgis_topology
   - address_standardizer
   - address_standardizer_data_us

   对当前安装的每个扩展运行以下命令。

   ```
   ALTER EXTENSION PostgreSQL-extension UPDATE TO 'new-version';
   ```

   有关更多信息，请参阅 升级 PostgreSQL 扩展。要了解有关升级 PostGIS 的更多信息，请参阅步骤 6：升级 PostGIS 扩展。

8. 如果要升级到版本 11.x，请在执行主要版本升级之前删除它不支持的扩展。要删除的扩展包括：

- chkpass
- tsearch2

9. 删除 unknown 数据类型，具体取决于您的目标版本。

   PostgreSQL 版本 10 不支持 unknown 数据类型。如果版本 9.6 数据库使用 unknown 数据类型，升级到版本 10 将显示错误消息，如下所示。

   ```
   Database instance is in a state that cannot be upgraded: PreUpgrade checks failed:
   The instance could not be upgraded because the 'unknown' data type is used in user
    tables.
   Please remove all usages of the 'unknown' data type and try again."
   ```

   要在数据库中查找 unknown 数据类型，以便删除上述列或将其更改为支持的数据类型，请为每个数据库使用以下 SQL 代码。

   ```
   SELECT n.nspname, c.relname, a.attname
       FROM pg_catalog.pg_class c,
       pg_catalog.pg_namespace n,
       pg_catalog.pg_attribute a
       WHERE c.oid = a.attrelid AND NOT a.attisdropped AND
       a.atttypid = 'pg_catalog.unknown'::pg_catalog.regtype AND
       c.relkind IN ('r','m','c') AND
       c.relnamespace = n.oid AND
       n.nspname !~ '^pg_temp_' AND
       n.nspname !~ '^pg_toast_temp_' AND n.nspname NOT IN ('pg_catalog',
    'information_schema');
   ```

10执行空运行升级。

   我们强烈建议您先使用生产数据库的副本测试主要版本升级，然后再对生产数据库真正执行升级。您可以监控重复测试实例上的执行计划，以了解任何可能的执行计划回归，并评估其性能。要创建副本测试实例，您可以从最近的快照还原数据库，或者克隆数据库。有关更多信息，请参阅 从快照还原 或 克隆 Amazon Aurora 数据库集群卷。

   有关更多信息，请参阅“将 Aurora PostgreSQL 引擎升级到新的主要版本”。

11升级您的生产实例。

   当试验性运行主要版本升级获得成功时，您应该可以放心地升级您的生产数据库。有关更多信息，请参阅 将 Aurora PostgreSQL 引擎升级到新的主要版本。

> **Note**
>
> 在升级过程中，如果数据库集群的备份保留期大于 0，Aurora PostgreSQL 会获取集群快照。在此过程中，您无法执行集群的时间点还原。之后，您可以执行时间点还原，以将实例还原到实例自动快照完成之后、升级操作开始之前的时间。但是，您不能通过执行时间点还原来还原到以前的次要版本。

有关进行中的升级的信息，您可以使用 Amazon RDS 来查看 pg_upgrade 实用工具生成的两个日志。它们是 `pg_upgrade_internal.log` 和 `pg_upgrade_server.log`。Amazon Aurora 会在这些日志的文件名中附加时间戳。您可以像查看其他任何日志一样查看这些日志。有关更多信息，请参阅“监控 Amazon Aurora 日志文件”。

12. 升级 PostgreSQL 扩展。PostgreSQL 升级过程不会升级任何 PostgreSQL 扩展。有关更多信息，请参阅 升级 PostgreSQL 扩展。

升级后建议

完成主要版本升级后，我们建议您执行以下操作：

- 运行 ANALYZE 操作以刷新 `pg_statistic` 表。您应该为所有 PostgreSQL 数据库实例上的每个数据库执行此操作。在主要版本升级期间不会传输优化程序统计数据，因此您需要重新生成所有统计数据，避免出现性能问题。运行不带任何参数的命令，为当前数据库中的所有常规表生成统计数据，如下所示：

```
ANALYZE VERBOSE;
```

  VERBOSE 标记为可选项，可用于显示进度。有关更多信息，请参阅 PostgreSQL 文档中的 ANALYZE。

  > **Note**
  >
  > 升级后在系统上运行 ANALYZE，避免出现性能问题。

- 如果您升级到 PostgreSQL 版本 10，请在您拥有的任何哈希索引上运行 REINDEX。哈希索引在版本 10 中已更改，必须重新构建。要查找无效的哈希索引，请针对包含哈希索引的每个数据库运行以下 SQL。

```
SELECT idx.indrelid::regclass AS table_name,
   idx.indexrelid::regclass AS index_name
FROM pg_catalog.pg_index idx
   JOIN pg_catalog.pg_class cls ON cls.oid = idx.indexrelid
   JOIN pg_catalog.pg_am am ON am.oid = cls.relam
WHERE am.amname = 'hash'
AND NOT idx.indisvalid;
```

- 建议您在升级的数据库上使用类似的工作负载测试应用程序，以验证是否一切正常。验证升级之后，您可以删除此测试实例。

将 Aurora PostgreSQL 引擎升级到新的主要版本

当您启动升级到新的主要版本的过程时，Aurora PostgreSQL 会在对集群进行任何更改之前拍摄 Aurora DB 集群的快照。此快照仅针对主要版本升级（而不是次要版本升级）而创建。升级过程完成后，您可以在 RDS 控制台的 Snapshots（快照）下列出的手动快照中找到此快照。快照名称包括 `preupgrade`（作为其前缀）、Aurora PostgreSQL 数据库集群的名称、源版本、目标版本以及日期和时间戳，如以下示例所示。

```
preupgrade-docs-lab-apg-global-db-12-8-to-13-6-2022-05-19-00-19
```

升级完成后，如有必要，您可以使用 Aurora 创建并存储在手动快照列表中的快照，将数据库集群还原到其以前的版本。

> **Tip**
>
> 一般来说，快照提供了许多将 Aurora 数据库集群还原到不同时间点的方法。要了解更多信息，请参阅 从数据库集群快照还原 和 将数据库集群还原到指定时间。但是，Aurora PostgreSQL 不支持使用快照还原到以前的次要版本。

在主要版本升级过程中，Aurora 将分配卷并克隆源 Aurora PostgreSQL 数据库集群。如果升级由于任何原因而失败，Aurora PostgreSQL 将使用克隆回滚升级。当分配的源卷克隆超过 15 个之后，后续克隆将成为完整副本并且需要更长的时间。这可能导致升级过程也要花费更长的时间。如果 Aurora PostgreSQL 回滚升级，请注意以下事项：

- 您可能会看到在升级期间分配的原始卷和克隆卷的账单条目和指标。在集群备份保留时段超过升级时间之后，Aurora PostgreSQL 会清理额外的卷。

- 此集群中的下一个跨区域快照副本将为完整副本，而不是增量副本。

为了安全地升级构成集群的数据库实例，Aurora PostgreSQL 使用 pg_upgrade 实用程序。写入器升级完成后，每个读取器实例在升级到新的主要版本时都会出现短暂中断。要了解有关此 PostgreSQL 实用程序的更多信息，请参阅 PostgreSQL 文档中的 pg_upgrade。

您可以使用以下方法将 Aurora PostgreSQL 数据库集群升级到新版本：AWS Management Console、AWS CLI 或 RDS API。

控制台

升级数据库集群的引擎版本

1. 登录AWS Management Console并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Databases (数据库)，然后选择要升级的数据库集群。
3. 选择修改。此时会显示修改数据库集群页面。
4. 对于数据库引擎版本，选择新版本。
5. 选择继续，查看修改摘要。
6. 要立即应用更改，请选择立即应用。选择此选项在某些情况下可能导致中断。有关更多信息，请参阅“修改 Amazon Aurora 数据库集群”。
7. 在确认页面上，检查您的更改。如果更改正确无误，请选择修改集群以保存更改。

   也可以选择 Back 编辑您的更改，或选择 Cancel 取消更改。

AWS CLI

要升级数据库集群的引擎版本，请使用 AWS CLI 命令 modify-db-cluster。指定以下参数：

- `--db-cluster-identifier` – 数据库集群的名称。
- `--engine-version` – 数据库引擎要升级到的版本号。有关有效的引擎版本的信息，请使用 AWS CLI describe-db-engine-versions 命令。
- `--allow-major-version-upgrade` – 当 `--engine-version` 参数是不同于数据库集群当前主要版本的主要版本时，这是必需的标志。
- `--no-apply-immediately` – 在下一维护时段内应用更改。要立即应用更改，请使用 `--apply-immediately`。

Example

对于 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --engine-version new_version \
    --allow-major-version-upgrade \
    --no-apply-immediately
```

对于 Windows：

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --engine-version new_version ^
    --allow-major-version-upgrade ^
    --no-apply-immediately
```

RDS API

要升级数据库集群的引擎版本，请使用 ModifyDBCluster 操作。指定以下参数：

- `DBClusterIdentifier` – 数据库集群的名称，例如 *mydbcluster*。
- `EngineVersion` – 数据库引擎要升级到的版本号。有关有效的引擎版本的信息，请使用 DescribeDBEngineVersions 操作。
- `AllowMajorVersionUpgrade` – 当 `EngineVersion` 参数是不同于数据库集群当前主要版本的主要版本时，这是必需的标志。
- `ApplyImmediately` – 是立即应用更改还是在下一个维护时段内应用更改。要立即应用更改，请将该值设置为 `true`。要在下一个维护时段内应用更改，请将该值设置为 `false`。

全局数据库的主要版本升级

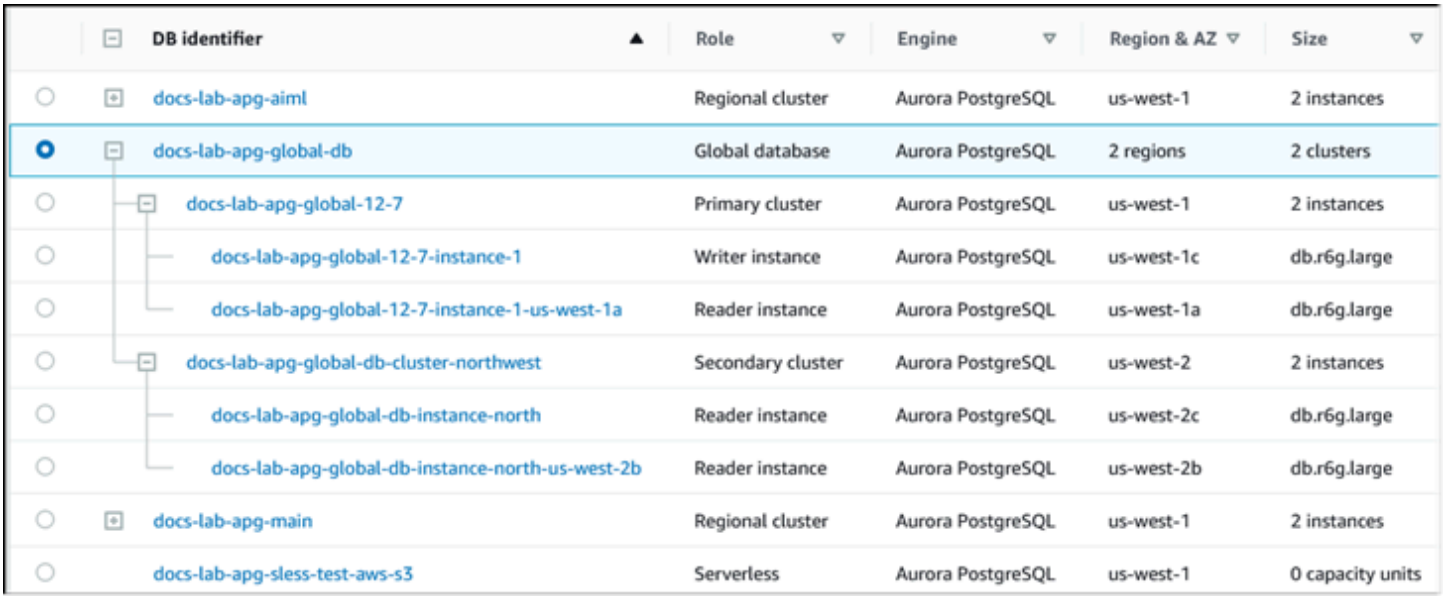
对于 Aurora 全局数据库集群，升级过程会同时升级构成 Aurora 全局数据库的所有数据库集群。这样做是为了确保每个数据库集群都运行相同的 Aurora PostgreSQL 版本。它还可确保对系统表、数据文件格式等所做的任何更改都会自动复制到所有辅助集群。

要将全局数据库集群升级到 Aurora PostgreSQL 的新主要版本，我们建议您在升级的版本上测试应用程序，如测试将生产数据库集群升级到新的主要版本中所述。请确保在升级之前为 Aurora 全局数据库

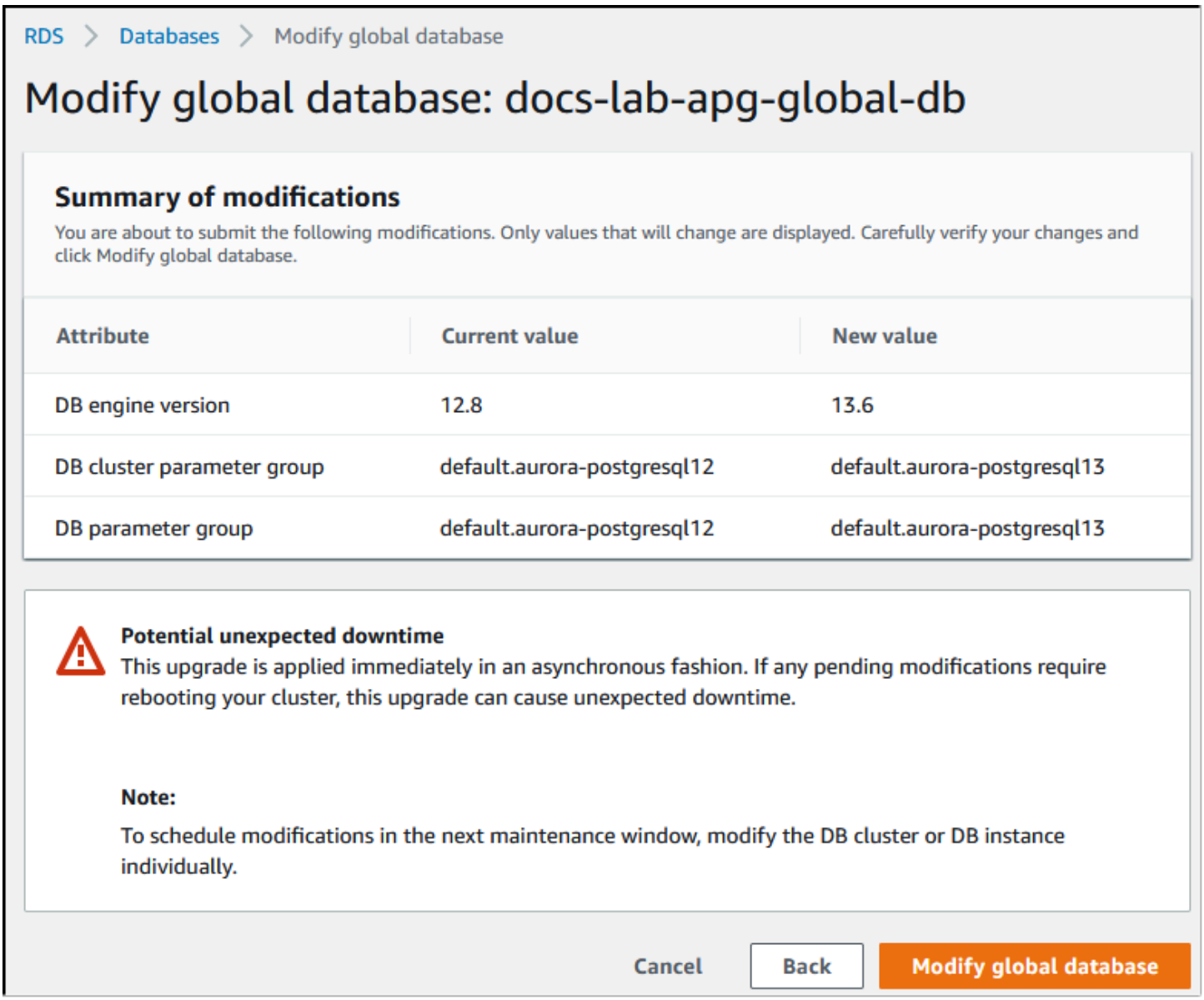
中的每个 AWS 区域准备好数据库集群参数组和数据库参数组设置，详见测试将生产数据库集群升级到新的主要版本中的step 1.。

如果 Aurora PostgreSQL 全局数据库集群为其 `rds.global_db_rpo` 参数设置了恢复点目标 (RPO)，请确保在升级之前重置此参数。如果开启了 RPO，主要版本升级过程将不起作用。默认情况下，该参数处于停用状态。有关 Aurora PostgreSQL 全局数据库和 RPO 的更多信息，请参阅管理基于 Aurora PostgreSQL 的全局数据库的 RPO。

如果您验证应用程序可以在新版本的试用部署时按预期运行，则可以启动升级过程。为此，请参阅 将 Aurora PostgreSQL 引擎升级到新的主要版本。请确保从 RDS 控制台的 Databases（数据库）列表中选择顶级项，即 Global database（全局数据库），如下图所示。



与任何修改一样，当出现提示时，您可以确认您希望继续执行该过程。



您可以使用 AWS CLI 或 RDS API 启动升级过程，而不是使用控制台。与控制台一样，您在 Aurora 全局数据库集群而不是其任何组件上运行，如下所示：

- 使用 AWS CLI 命令 modify-global-cluster，通过使用 AWS CLI 启动 Aurora 全局数据库的升级。
- 使用 ModifyGlobalCluster API 启动升级。

执行次要版本升级

可以使用以下方法升级数据库集群的次要版本或修补数据库集群：

主题

- 在执行次要版本升级之前
- 如何执行次要版本升级和应用补丁
- 次要版本升级和零停机时间修补
- 零停机时间修补的限制
- 将 Aurora PostgreSQL 引擎升级到新的次要版本

在执行次要版本升级之前

建议您执行以下操作以缩短次要版本升级期间的停机时间：

- 应在流量较低的时段执行 Aurora 数据库集群维护。使用 Performance Insights 来识别这些时间段，以便正确配置维护时段。有关 Performance Insights 的更多信息，请参阅在 Amazon RDS 上使用 Performance Insights 监控数据库负载。有关数据库集群维护时段的更多信息，请参阅调整首选数据库集群维护时段。
- 使用支持指数回退和抖动的 AWS SDK 作为最佳实践。有关更多信息，请参阅 Exponential Backoff And Jitter。

如何执行次要版本升级和应用补丁

次要版本升级和补丁只有在经过严格的测试后才会在 AWS 区域中推出。在发布升级和补丁之前，Aurora PostgreSQL 将进行测试，以确保次要社群版本发布后出现的已知安全问题、错误和其它问题不会破坏 Aurora PostgreSQL 实例集的稳定性。

如果您开启了启用自动次要版本升级，Aurora PostgreSQL 将在指定的维护时段定期升级您的数据库集群。确保对 Aurora PostgreSQL 数据库集群中的所有实例开启启用自动次要版本升级选项。有关如何设置自动次要版本升级以及该设置在集群和实例级别应用时的工作原理的信息，请参阅 Aurora 数据库集群的自动次要版本升级。

通过使用 describe-db-instances AWS CLI 命令，针对所有 Aurora PostgreSQL 数据库集群检查启用自动次要版本升级选项的值，如下所示。

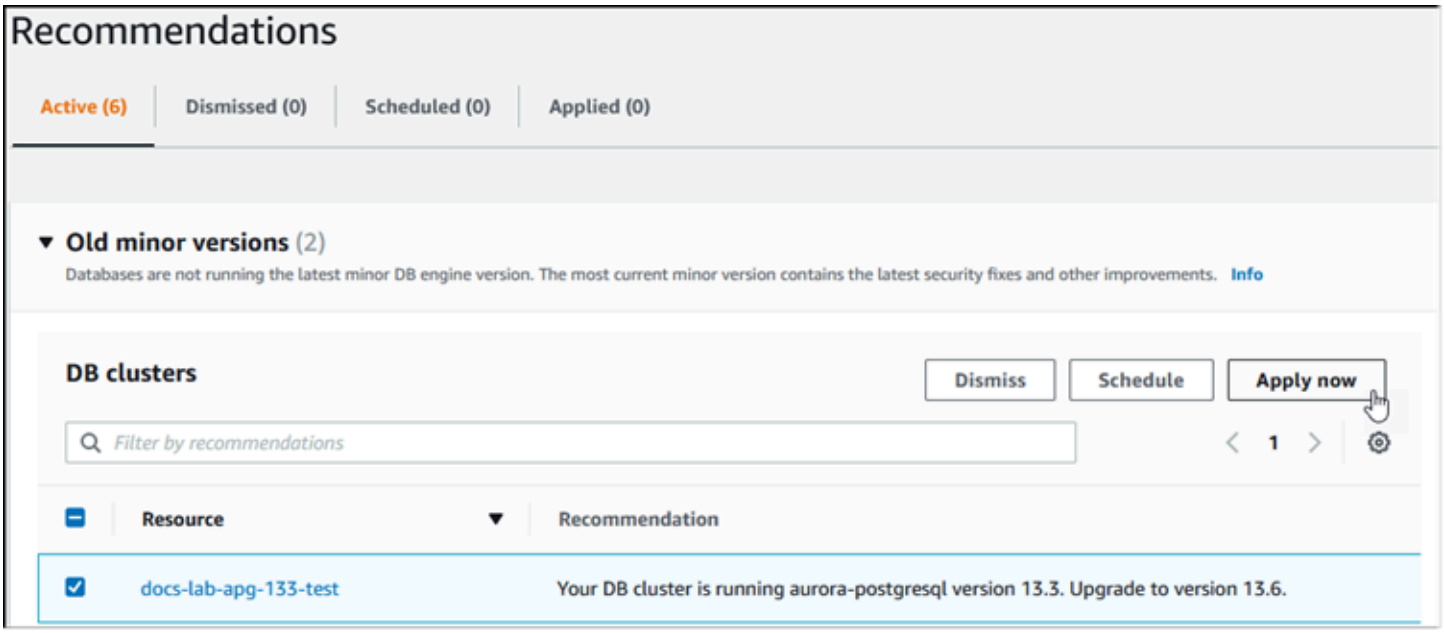
```
aws rds describe-db-instances \
  --query '*[].
{DBClusterIdentifier:DBClusterIdentifier,DBInstanceIdentifier:DBInstanceIdentifier,AutoMinorVer
```

此查询返回其 `AutoMinorVersionUpgrade` 设置的状态值为 `true` 或 `false` 的所有 Aurora DB 集群及其实例的列表。该命令假设您已将 AWS CLI 配置为以默认的 AWS 区域为目标。

有关 AmVU 选项以及如何修改 Aurora 数据库集群以使用该选项的更多信息，请参阅 Aurora 数据库集群的自动次要版本升级。

您可以通过响应维护任务或修改集群以使用新版本，将 Aurora PostgreSQL 数据库集群升级到新的次要版本。

通过使用 RDS 控制台并打开 Recommendations（建议）菜单，您可以确定 Aurora PostgreSQL 数据库集群的任何可用升级或补丁。在此处，您可以找到各种维护问题的列表，例如 Old minor versions（旧的次要版本）。根据您的生产环境，您可以选择 Schedule（计划）升级，或通过选择 Apply now（立即应用）立即采取行动，如下所示。



要了解有关如何维护 Aurora 数据库集群的更多信息，包括如何手动应用补丁和次要版本升级，请参阅维护 Amazon Aurora 数据库集群。

次要版本升级和零停机时间修补

升级 Aurora PostgreSQL 数据库集群可能会导致中断。在升级过程中，数据库会在升级时关闭。如果在数据库忙碌时开始升级，则会丢失数据库集群正在处理的所有连接和事务。如果等到数据库处于空闲状态再执行升级，则可能需要等待很长时间。

零停机修补 (ZDP) 特征改进了升级过程。使用 ZDP，可以应用次要版本升级和补丁，且对 Aurora PostgreSQL 数据库集群的影响最小。当对 Aurora PostgreSQL 版本以及这些次要版本的其他更高版本和更高的主要版本应用补丁或更高次要版本升级时，使用 ZDP。也就是说，从这些版本中的任何一个升级到新的次要版本都使用 ZDP。

下表显示提供了 ZDP 的 Aurora PostgreSQL 版本和数据库实例类：

版本	db.r* 实例类	db.t* 实例类	db.x* 实例类	db.serverless 实例类
10.21 及更高版本	支持	是	是	不适用
11.16 及更高版本	支持	是	是	不适用
11.17 及更高版本	支持	是	是	不适用
12.11 及更高版本	支持	是	是	不适用
12.12 及更高版本	支持	是	是	不适用
13.7 及更高版本	支持	是	是	不适用
13.8 及更高版本	支持	是	是	是
14.3 及更高版本	支持	是	是	不适用
14.4 及更高版本	支持	是	是	不适用
14.5 及更高版本	支持	是	是	是
15.3 及更高版本	支持	是	是	是
16.1 及更高版本	支持	是	是	是

在使用 ZDP 的升级过程中，数据库引擎会寻找一个安静点来暂停所有新事务。此操作可在应用补丁和升级期间保护数据库。为了确保您的应用程序在事务暂停的情况下平稳运行，我们建议将重试逻辑集成到您的代码中。这种方法可确保系统能够管理任何短暂的停机时间而不会出现故障，并且可以在升级后重试新事务。

当 ZDP 成功完成时，应用程序会话将保持（但断开连接的会话除外），并且数据库引擎将在升级仍在进行时重新启动。尽管数据库引擎重新启动可能会导致吞吐量临时下降，但这通常只持续几秒钟，最多约一分钟。

在某些情况下，零停机修补 (ZDP) 可能无法成功。例如，Aurora PostgreSQL 数据库集群或其实例上处于 `pending` 状态的参数更改会干扰 ZDP。

您可以在控制台的 Events（事件）页面中找到 ZDP 操作的指标和事件。这些事件包括 ZDP 升级的开始和升级完成。在这种情况下，您可以了解该过程所用的时长，以及在重新启动过程中发生的保留连接和断开连接的数量。您可以在数据库错误日志中找到详细信息。

零停机时间修补的限制

零停机时间修补存在以下限制：

- 在整个 Aurora PostgreSQL 升级过程中，ZDP 尝试保留与 Aurora PostgreSQL 写入器实例的当前客户端连接。但在以下情况下，连接将断开，以便 ZDP 完成：
  - 正在执行长时间运行的查询或事务。
  - 数据定义语言（DDL）语句正在运行。
  - 正在使用临时表或表锁定。
  - 所有会话都在侦听通知渠道。
  - 处于“WITH HOLD”状态的光标正在使用中。
  - 使用的是 TLSv1.1 连接。从 Aurora PostgreSQL 高于 16.1、15.3、14.8、13.11、12.15 和 11.20 的版本开始，TLSv1.3 连接支持 ZDP。
- 在以下情况下，不支持 ZDP：
  - 当 Aurora PostgreSQL 数据库集群配置为 Aurora Serverless v1 时。
  - 在升级任意 Aurora 读取器实例期间。
  - 在升级辅助区域中属于 Aurora Global Database 集群的任何 Aurora 读取器实例期间。
  - 在操作系统修补和操作系统升级期间。

将 Aurora PostgreSQL 引擎升级到新的次要版本

您可以使用以下方法将 Aurora PostgreSQL 数据库集群升级到新的次要版本：控制台、AWS CLI 或 RDS API。在执行升级之前，我们建议您遵循我们针对主要版本升级推荐的最佳实践。与新的主要版本一样，新的次要版本也可以对优化器进行改进（例如修复），这可能会导致查询计划回归。为确保计划稳定性，我们建议您使用查询计划管理（QPM）扩展，详情请参阅确保主要版本升级后的计划稳定性。

控制台

升级 Aurora PostgreSQL 数据库集群的引擎版本

1. 登录AWS Management Console并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Databases (数据库)，然后选择要升级的数据库集群。
3. 选择修改。此时会显示修改数据库集群页面。
4. 对于数据库引擎版本，选择新版本。
5. 选择继续，查看修改摘要。
6. 要立即应用更改，请选择立即应用。选择此选项在某些情况下可能导致中断。有关更多信息，请参阅“修改 Amazon Aurora 数据库集群”。
7. 在确认页面上，检查您的更改。如果更改正确无误，请选择修改集群以保存更改。

   也可以选择 Back 编辑您的更改，或选择 Cancel 取消更改。

AWS CLI

要升级数据库集群的引擎版本，请结合使用 AWS CLI 命令 modify-db-cluster 与以下参数：

- `--db-cluster-identifier` – Aurora PostgreSQL 数据库集群的名称。
- `--engine-version` – 数据库引擎要升级到的版本号。有关有效的引擎版本的信息，请使用 AWS CLI describe-db-engine-versions 命令。
- `--no-apply-immediately` – 在下一维护时段内应用更改。要立即应用更改，请改用 `--apply-immediately`。

对于 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --engine-version new_version \
    --no-apply-immediately
```

对于 Windows：

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
```

```
    --engine-version new_version ^
    --no-apply-immediately
```

RDS API

要升级数据库集群的引擎版本，请使用 ModifyDBCluster 操作。指定以下参数：

- `DBClusterIdentifier` – 数据库集群的名称，例如 *`mydbcluster`*。
- `EngineVersion` – 数据库引擎要升级到的版本号。有关有效的引擎版本的信息，请使用 DescribeDBEngineVersions 操作。
- `ApplyImmediately` – 是立即应用更改还是在下一个维护时段内应用更改。要立即应用更改，请将该值设置为 `true`。要在下一个维护时段内应用更改，请将该值设置为 `false`。

升级 PostgreSQL 扩展

将 Aurora PostgreSQL 数据库集群升级到新的主要或次要版本不会同时升级 PostgreSQL 扩展。对于大多数扩展，您可以在主要或次要版本升级完成后升级扩展。但是，在某些情况下，应在升级 Aurora PostgreSQL 数据库引擎之前升级扩展。有关更多信息，请参阅测试将生产数据库集群升级到新的主要版本中的list of extensions to update。

安装 PostgreSQL 扩展需要具备 `rds_superuser` 权限。通常，`rds_superuser` 将对于特定扩展的权限委派给相关用户（角色），以便于管理给定扩展。这意味着，升级 Aurora PostgreSQL 数据库集群中所有扩展的任务可能涉及许多不同用户（角色）。如果要使用脚本自动执行升级过程，请尤其注意这一点。有关 PostgreSQL 权限和角色的更多信息，请参阅 使用 Amazon Aurora PostgreSQL 实现高安全性。

> **Note**
>
> 有关更新 PostGIS 扩展的信息，请参阅使用 PostGIS 扩展管理空间数据（步骤 6：升级 PostGIS 扩展）。
> 要升级 `pg_repack` 扩展，先删除该扩展，然后在升级后的数据库实例中创建新版本。有关更多信息，请参阅 `pg_repack` 文档中的安装 pg_repack。

要在引擎升级后更新扩展，请使用 `ALTER EXTENSION UPDATE` 命令。

```
ALTER EXTENSION extension_name UPDATE TO 'new_version';
```

要列出当前安装的扩展，请在以下命令中使用 PostgreSQL pg_extension 目录。

```
SELECT * FROM pg_extension;
```

要查看可用于安装的特定扩展版本的列表，请在以下命令中使用 PostgreSQL pg_available_extension_versions 视图。

```
SELECT * FROM pg_available_extension_versions;
```

备选的蓝绿升级技术

在某些情况下，您的首要任务是立即从旧集群切换到升级后的集群。在此类情况下，您可以使用多步骤流程，并排运行新旧集群。此处，您可以将数据从旧集群复制到新集群，直到您准备好接管新集群。有关详细信息，请参阅使用蓝绿部署进行数据库更新。

使用 Aurora PostgreSQL 长期支持（LTS）版本

每个新的 Aurora PostgreSQL 版本在一定时间内保持可用状态，以便在创建或升级数据库集群时使用。在此期间后，您必须升级使用该版本的任何集群。您可以在支持期限结束之前手动升级集群，或者 Aurora 可以在不再支持集群的 Aurora PostgreSQL 版本时自动升级集群。

Aurora 将某些 Aurora PostgreSQL 版本指定为长期支持（LTS）版本。与使用非 LTS 版本的数据库集群相比，使用 LTS 版本的集群可以将同一版本保留更长时间，并经历更短的升级周期。LTS 次要版本仅包含错误修复（通过补丁版本）；LTS 版本不包括其推出后发布的新功能。

每年一次，在 LTS 次要版本上运行的数据库集群都会被修补为该 LTS 版本的最新补丁版本。我们进行这种修补，是为了帮助您享受到更高的安全性与稳定性修复。如有关键补丁（例如安全补丁）需要更新，我们可能会更频繁地修补 LTS 次要版本。

> **Note**
>
> 如果您希望在 LTS 次要版本的生命周期内保持在该版本中，请确保禁用数据库实例的自动次要版本升级功能。要避免数据库集群从 LTS 次要版本自动进行升级，请对于 Aurora 集群中的任何数据库实例清除允许自动次要版本升级复选框。

对于大多数 Aurora PostgreSQL 集群，我们建议您升级到最新的版本，而不是使用 LTS 版本。这样做可以利用 Aurora 以作为托管服务，并使您可以获取最新的功能和错误修复。LTS 版本适用于具有以下特征的集群：

- 除了在极少数情况下应用关键补丁以外，您无法承受 Aurora PostgreSQL 应用程序停机带来的损失。
- 对于 Aurora PostgreSQL 数据库引擎的每次更新，集群和关联的应用程序的测试周期需要很长的时间。
- Aurora PostgreSQL 集群的数据库版本具有应用程序所需的所有数据库引擎功能和错误修复。

Aurora PostgreSQL 的当前 LTS 版本如下：

- PostgreSQL 15.10。该版本于 2024 年 12 月 27 日发布。有关更多信息，请参阅《Aurora PostgreSQL 发布说明》中的 PostgreSQL 15.10。
- PostgreSQL 14.6。已于 2023 年 1 月 20 日发布。有关更多信息，请参阅《Aurora PostgreSQL 发布说明》中的 PostgreSQL 14.6。
- PostgreSQL 13.9。已于 2023 年 1 月 20 日发布。有关更多信息，请参阅《Aurora PostgreSQL 发布说明》中的 PostgreSQL 13.9。
- PostgreSQL 12.9。该版本于 2022 年 2 月 25 日发布。有关更多信息，请参阅《Aurora PostgreSQL 发布说明》中的 PostgreSQL 12.9。
- PostgreSQL 11.9（Aurora PostgreSQL 版本 3.4）。该产品于 2020 年 12 月 11 日发布。有关此版本的更多信息，请参阅《Aurora PostgreSQL 发布说明》中的 PostgreSQL 11.9、Aurora PostgreSQL 版本 3.4。

有关 LTS 版本的支持时间表和发布周期的详细信息，请参阅 Release calendars for Aurora PostgreSQL。

有关如何确定 Aurora 和数据库引擎版本的信息，请参阅 确定 Amazon Aurora PostgreSQL 版本。

使用 Amazon Aurora PostgreSQL Limitless Database

Amazon Aurora PostgreSQL Limitless Database 提供自动水平扩缩，每秒可处理数百万个写入事务，管理数 PB 的数据，同时保持在单个数据库内操作的简便性。借助 Aurora PostgreSQL Limitless Database，您可以专注于构建大规模应用程序，而无需为支持工作负载而构建和维护用于跨多个数据库实例扩展数据的复杂解决方案。

主题

- Aurora PostgreSQL Limitless Database 架构
- Aurora PostgreSQL Limitless Database 入门
- Aurora PostgreSQL Limitless Database 要求和注意事项
- 使用 Aurora PostgreSQL Limitless Database 的先决条件
- 创建使用 Aurora PostgreSQL Limitless Database 的数据库集群
- 使用数据库分片组
- 创建 Aurora PostgreSQL Limitless Database 表
- 将数据加载到 Aurora PostgreSQL Limitless Database
- 查询 Aurora PostgreSQL Limitless Database
- 管理 Aurora PostgreSQL Limitless Database
- 监控 Aurora PostgreSQL Limitless Database
- 备份和还原 Aurora PostgreSQL Limitless Database
- 升级 Amazon Aurora PostgreSQL Limitless Database
- Aurora PostgreSQL Limitless Database 参考

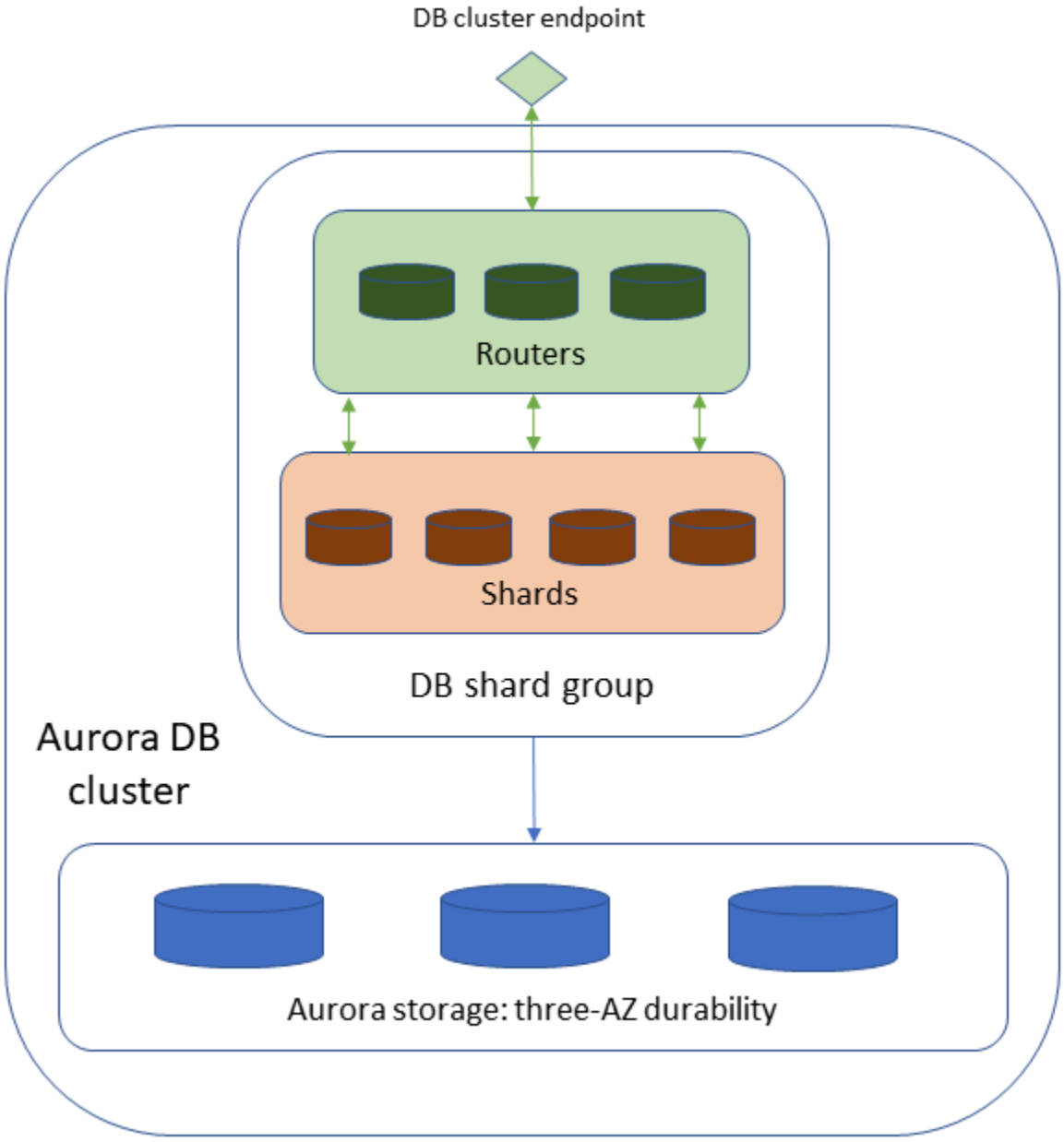
Aurora PostgreSQL Limitless Database 架构

Limitless Database 通过由多个数据库节点组成的双层架构实现扩展。节点要么是路由器，要么是分片。

- 分片是 Aurora PostgreSQL 数据库实例，每个实例都存储数据库的数据子集，允许同时处理以实现更高的写入吞吐量。
- 路由器管理数据库的分布式特性，并向数据库客户端提供单个数据库映像。路由器维护有关数据存储位置的元数据，解析传入的 SQL 命令并将这些命令发送到分片。然后，它们聚合来自分片的数据以将单个结果返回给客户端，并管理分布式事务以保持整个分布式数据库的一致性。

Aurora PostgreSQL Limitless Database 与标准 Aurora 数据库集群 的不同之处在于，它使用的是数据库分片组，而不是写入器数据库实例和读取器数据库实例。构成 Limitless Database 架构的所有节点都包含在数据库分片组中。数据库分片组中的各个分片和路由器在您的 AWS 账户中不可见。您可以使用数据库集群端点访问 Limitless Database。

下图显示了 Aurora PostgreSQL Limitless Database 的大致架构。



有关 Aurora PostgreSQL Limitless Database 架构以及如何使用它的更多信息，请观看 YouTube AWS 活动频道中的通过 Amazon Aurora PostgreSQL Limitless Database 实现扩展。

有关标准 Aurora 数据库集群架构的更多信息，请参阅 Amazon Aurora 数据库集群。

Aurora PostgreSQL Limitless Database 的关键术语

数据库分片组

Limitless Database 节点（分片和路由器）的容器。

路由器

接受来自客户端的 SQL 连接、向分片发送 SQL 命令、保持系统范围一致性并将结果返回给客户端的节点。

分片

存储分片表子集、参考表的完整副本和标准表的节点。接受来自路由器的查询，但不能由客户端直接连接。

分片表

数据在分片中被分割的表。

分片键

分片表中的一列或一组列，用于确定分片间的分区。

并置表

两个共享相同分片键并显式声明为并置的分片表。相同分片键值的所有数据都将发送到同一个分片。

引用表

一个表，其数据已全部复制到每个分片上。

标准表

Limitless Database 中的默认表类型。您可以将标准表转换为分片表和引用表。

所有标准表都存储在系统选择的同一个分片上，这样就可以在单个分片中执行标准表之间的联接。但是，标准表受分片的最大容量（128 TiB）的限制。该分片还存储来自分片表和引用表的数据，因此标准表的有效限制低于 128 TiB。

Aurora PostgreSQL Limitless Database 的表类型

Aurora PostgreSQL Limitless Database 支持三种类型的表：分片表、引用表和标准表。

分片表的数据分布在数据库分片组中的所有分片上。Limitless Database 使用分片键自动执行此操作，分片键是您在对表进行分区时指定的一列或一组列。拥有相同分片键值的所有数据都会发送到同一个分片。分片基于哈希，不基于范围或列表。

以下是分片表的良好用例：

- 该应用程序使用不同的数据子集。
- 该表非常大。
- 该表的增长速度可能比其他表快。

分片表可以并置，这意味着它们共享相同的分片键，并且两个表中具有相同分片键值的所有数据都将发送到同一个分片。如果您配置表并使用分片键将其联接，则可以在单个分片上执行联接，因为所有必需的数据都位于该分片上。

引用表在数据库分片组中的每个分片上都有其所有数据的完整副本。引用表通常用于写入量较小的表，但这些表仍需要经常联接，且不适合分片。引用表的示例包括日期表和地理数据表，例如州/省、城市和邮政编码。

标准表是 Aurora PostgreSQL Limitless Database 中的默认表类型。它们不是分布式表。Aurora PostgreSQL Limitless Database 支持标准表与标准表、分片表和引用表之间的联接。

Aurora PostgreSQL Limitless Database 的计费

有关 Aurora PostgreSQL Limitless Database 收费方式的信息，请参阅 Aurora 的数据库实例计费。

有关 Aurora 定价信息，请参阅 Aurora 定价页面。

Aurora PostgreSQL Limitless Database 入门

要开始使用 Aurora PostgreSQL Limitless Database，您需要执行以下操作：

1. 为 Limitless Database 创建 Aurora PostgreSQL 数据库集群和数据库分片组。有关更多信息，请参阅 创建使用 Aurora PostgreSQL Limitless Database 的数据库集群。
2. 在数据库分片组中创建分片表和引用表。有关更多信息，请参阅 创建 Aurora PostgreSQL Limitless Database 表。
3. 更改数据库分片组的容量，拆分分片并添加路由器。有关更多信息，请参阅 使用数据库分片组。
4. 将数据加载到数据库分片组。有关更多信息，请参阅 将数据加载到 Aurora PostgreSQL Limitless Database。
5. 在数据库分片组上运行查询和其他 SQL 语句。有关更多信息，请参阅 查询 Aurora PostgreSQL Limitless Database。
6. 监控 Limitless Database 的性能。有关更多信息，请参阅 监控 Aurora PostgreSQL Limitless Database。

Aurora PostgreSQL Limitless Database 要求和注意事项

Aurora PostgreSQL Limitless Database 有以下要求和注意事项。

主题

- Aurora PostgreSQL Limitless Database 的要求
- Aurora PostgreSQL Limitless Database 的注意事项
- Aurora PostgreSQL Limitless Database 不支持的功能

Aurora PostgreSQL Limitless Database 的要求

确保遵循 Aurora PostgreSQL Limitless Database 的以下要求。

- 以下 AWS 区域可用：
  - 亚太地区（香港）
  - 亚太地区（新加坡）
  - 亚太地区（悉尼）
  - 亚太地区（东京）
  - 欧洲地区（法兰克福）
  - 欧洲地区（爱尔兰）
  - 欧洲地区（斯德哥尔摩）
  - 美国东部（弗吉尼亚州北部）

    > **Note**
    >
    > 如果您在此 AWS 区域创建 Limitless Database 数据库集群，请不要在数据库子网组中包含 `us-east-1e` 可用区（AZ）。由于资源限制，`us-east-1e` 可用区不支持 Aurora Serverless v2，因此也不支持 Limitless Database。

  - 美国东部（俄亥俄州）
  - 美国西部（俄勒冈州）

本指南中的 AWS CLI 程序假设您使用的是其中一个可用的 AWS 区域。有关如何为 AWS CLI 设置默认 AWS 区域的信息，请参阅《AWS Command Line Interface 用户指南版本 2》中的如何设置环境变量。

- Aurora PostgreSQL Limitless Database 仅支持 Aurora I/O-Optimized 数据库集群存储配置。有关更多信息，请参阅 Amazon Aurora 数据库集群的存储配置。
- Aurora PostgreSQL Limitless Database 为 Aurora PostgreSQL Limitless Database 使用特殊版本的 Aurora PostgreSQL 数据库引擎：
  - `16.4-limitless`
  - `16.6-limitless`
- 您的数据库集群不能有任何写入器或读取器数据库实例。
- 您必须使用增强监控和性能详情。性能详情的保留时间必须至少为 1 个月（31 天）。
- 您必须将 PostgreSQL 日志导出到 Amazon CloudWatch Logs。

> **Note**
>
> 某些必需的功能（例如增强监控、性能详情和 CloudWatch Logs）会产生额外费用。有关 Aurora 定价信息，请参阅 Aurora 定价页面。

Aurora PostgreSQL Limitless Database 的注意事项

以下注意事项适用于 Aurora PostgreSQL Limitless Database 中的数据库分片组。

- 每个数据库集群只能有一个数据库分片组。
- 每个 AWS 区域最多可以有五个数据库分片组。

  因此，每个 AWS 区域最多可以有五个 Aurora PostgreSQL Limitless Database 数据库集群。有关更多信息，请参阅 Amazon Aurora 中的配额。
- 您可以将数据库分片组的最大容量设置为 16-6144 ACU。如果容量限制高于 6144 ACU，请联系 AWS。

  初始路由器和分片数量由您在创建数据库分片组时设置的最大容量决定。有关更多信息，请参阅 将数据库分片组的最大容量与创建的路由器和分片的数量相关联。
- 修改数据库分片组的最大容量时，路由器和分片的数量不会改变。
- 确保创建数据库分片组的数据库子网有足够的可用的 IP 地址用于连接数据库分片组。数据库分片组中的每台路由器需要一个 IP 地址，每个分片最多需要三个 IP 地址。

  有关创建数据库分片组时创建的路由器数量的更多信息，请参阅将数据库分片组的最大容量与创建的路由器和分片的数量相关联。

- 如果您将数据库分片组设置为可公开访问，请务必在您的 VPC 中设置互联网网关。
- 您可以使用 SQL 函数来拆分分片并添加路由器。
- 不支持合并分片。
- 您无法删除单个分片和路由器。
- 您不能以任何方式修改（对分片键执行 UPDATE 操作）分片键，包括更改其在表行中的值。

  要更改分片键，请先将其删除，然后重新创建。
- 支持可重复读取、已提交读取和未提交读取隔离级别。您不能将隔离级别设置为可序列化。
- 不支持一些 SQL 命令。有关更多信息，请参阅 Aurora PostgreSQL Limitless Database 参考。
- 并非所有的 PostgreSQL 扩展都受支持。有关更多信息，请参阅 扩展。
- 创建分片组或添加新的分片组节点（分片或路由器）时，这些节点将在数据库集群可用的可用区（AZ）中创建。您不能为单个节点选择特定的可用区。
- 如果您使用 2 的计算冗余（数据库分片组有两个计算备用数据库），请确保您的数据库子网组至少有三个可用区。
- Aurora PostgreSQL Limitless Database 支持最多 54 个字符作为分片表名称。

以下注意事项适用于 Aurora PostgreSQL Limitless Database 数据库集群：

- 建议您使用 AWS 托管策略将数据库和应用程序的权限限制为客户用例所需的权限。有关更多信息，请参阅 策略最佳实践。
- 创建 Aurora PostgreSQL Limitless Database 数据库集群时，您只需为数据库分片组设置扩展参数。
- 您无法停止或启动包含数据库分片组的数据库集群。
- 如果您需要删除数据库集群，则必须先删除数据库分片组。
- Aurora PostgreSQL Limitless Database 不能作为复制源。

Aurora PostgreSQL Limitless Database 不支持的功能

Aurora PostgreSQL Limitless Database 不支持以下 Aurora PostgreSQL 功能：

- Active Directory（Kerberos）身份验证
- Amazon DevOps Guru
- Amazon ElastiCache
- Amazon RDS 蓝绿部署
- Amazon RDS 代理
- Aurora Auto Scaling（自动向数据库集群添加读取器实例）
- Aurora Global Database
- Aurora 机器学习
- Aurora 建议
- Aurora Serverless v1
- Aurora 零 ETL 集成
- AWS Backup
- AWS Lambda 集成
- AWS Secrets Manager
- 适用于 Aurora PostgreSQL 的 Babelfish
- 克隆数据库集群
- 自定义端点
- 数据库活动流
- 只读副本
- RDS 数据 API

使用 Aurora PostgreSQL Limitless Database 的先决条件

要使用 Aurora PostgreSQL Limitless Database，必须先执行以下任务。

主题

- 启用数据库分片组操作

启用数据库分片组操作

在创建数据库分片组之前，必须启用数据库分片组操作。

- 将以下部分添加到访问 Aurora PostgreSQL Limitless Database 的用户 IAM 角色的 IAM 策略中：

  JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowDBShardGroup",
            "Effect": "Allow",
            "Action": [
                "rds:CreateDBShardGroup",
                "rds:DescribeDBShardGroups",
                "rds:DeleteDBShardGroup",
                "rds:ModifyDBShardGroup",
                "rds:RebootDBShardGroup"
            ],
            "Resource": [
                "arn:aws:rds:*:*:shard-group:*",
                "arn:aws:rds:*:*:cluster:*"
            ]
        }
    ]
}
```

创建使用 Aurora PostgreSQL Limitless Database 的数据库集群

您可以使用 Limitless Database 版本的 Aurora PostgreSQL 创建新的 Aurora 数据库集群，然后向该集群添加数据库分片组。添加数据库分片组时，您可以以 Aurora 容量单位（ACU）指定整个数据库分片组的最大计算容量（所有路由器和分片的容量之和）。每个 ACU 是约 2 GiB 的内存、相应的 CPU 和网络的组合。扩展会增加或减少数据库分片组的容量，具体取决于您的应用程序工作负载，与 Aurora Serverless v2 的工作原理类似。

主题

- 将数据库分片组的最大容量与创建的路由器和分片的数量相关联
- 创建数据库集群

将数据库分片组的最大容量与创建的路由器和分片的数量相关联

数据库分片组的初始路由器和分片数量由您在创建数据库分片组时设置的最大容量决定。最大容量越大，在数据库分片组中创建的路由器和分片的数量就越多。

每个节点（分片或路由器）都有自己的当前容量值，也以 ACU 来衡量。

- 当节点的当前容量太低而无法处理负载时，Limitless Database 可将其扩展到更高的容量。但是，当总容量达到最大值时，节点将无法再扩展。
- 当节点的当前容量高于所需容量时，Limitless Database 会将其缩减到更低的容量。但是，当总容量达到最小值时，节点将无法再缩减。

下表显示了以 Aurora 容量单位（ACU）为单位的数据库分片组最大容量与创建的节点（路由器和分片）数量之间的相关性。

> **Note**
>
> 这些值可能会发生变化。
> 如果将计算冗余设置为非零值，则分片总数将增加一倍或两倍。这将产生额外费用。
> 计算备用数据库中的节点会扩展和缩减到与写入器相同的容量。您无需为备用数据库单独设置容量范围。

节点总数	路由器	分片	默认最小容量（ACU）	最大容量范围（ACU）
4	2	2	16	16-400
5	2	3	20	401-500
6	2	4	24	501-600
7	3	4	28	601-700
8	3	5	32	701-800
9	3	6	36	801-900
10	4	6	40	901-1000
11	4	7	44	1001-1100
12	4	8	48	1101-1200
13	5	8	52	1201-1300
14	5	9	56	1301-1400
15	5	10	60	1401-1500
16	6	10	64	1501-1600
17	6	11	68	1601-1700
18	6	12	72	1701-1800
19	7	12	76	1801-1900
20	7	13	80	1901-2000
21	7	14	84	2001-2100
22	8	14	88	2101-2200
23	8	15	92	2201-2300

节点总数	路由器	分片	默认最小容量（ACU）	最大容量范围（ACU）
24	8	16	96	2301-6144

数据库分片组基于最大容量的动态配置仅在创建期间可用。修改最大容量后，路由器和分片的数量保持不变。有关更多信息，请参阅 更改数据库分片组的容量。

您可以使用 SQL 命令将分片和路由器添加到数据库分片组。有关更多信息，请参阅下列内容：

- 在数据库分片组中拆分一个分片
- 向数据库分片组中添加路由器

> **Note**
>
> 您无法删除分片或路由器。

创建数据库集群

使用以下流程可创建使用 Aurora PostgreSQL Limitless Database 的 Aurora PostgreSQL 数据库集群。

您可以使用 AWS Management Console或 AWS CLI 来创建使用 Aurora PostgreSQL Limitless Database 的数据库集群。您可以创建主数据库集群和数据库分片组。

控制台

使用 AWS Management Console创建主数据库集群时，数据库分片组也会在相同的流程中创建。

使用控制台创建数据库集群

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 选择创建数据库。

   将显示创建数据库页面。
3. 对于引擎类型，选择 Aurora（PostgreSQL 兼容）。
4. 对于版本，请选择下列选项之一：

   - 带 Limitless Database 的 Aurora PostgreSQL（兼容 PostgreSQL 16.4）
   - 带 Limitless Database 的 Aurora PostgreSQL（兼容 PostgreSQL 16.6）
5. 对于 Aurora PostgreSQL Limitless Database：



a. 输入数据库分片组标识符。

> **Important**
>
> 创建数据库分片组后，您将无法更改数据库集群标识符或数据库分片组标识符。

b. 对于数据库分片组容量范围：

   i. 输入最小容量（ACU）。最小值为 16 ACU。

      对于开发环境，默认值为 16 ACU。对于生产环境，默认值为 24 ACU。

   ii. 输入最大容量（ACU）。使用至少 16 ACU 或最多 6144 ACU 的值。

      对于开发环境，默认值为 64 ACU。对于生产环境，默认值为 384 ACU。

   有关更多信息，请参阅 将数据库分片组的最大容量与创建的路由器和分片的数量相关联。

c. 对于数据库分片组部署，选择是否为数据库分片组创建备用数据库：

   - 没有计算冗余 – 为每个分片创建没有备用数据库的数据库分片组。这是默认值。
   - 使用单个失效转移目标计算冗余 – 在不同的可用区（AZ）中创建带有一个计算备用数据库的数据库分片组。

- 使用两个失效转移目标计算冗余 – 在两个不同的可用区中创建带有两个计算备用数据库的数据库分片组。

> **Note**
>
> 如果将计算冗余设置为非零值，则数据库实例的分片总数将增加一倍或两倍。这些额外的数据库实例是计算备用实例，可以纵向扩展和缩减到与写入器实例相同的容量。您无需为备用数据库单独设置容量范围。因此，ACU 用量和账单相应地增加一倍和两倍。要了解计算冗余产生的确切 ACU 用量，请参阅 DBShardGroup 指标中的 `DBShardGroupComputeRedundancyCapacity` 指标。

d. 选择是否可以公开访问数据库分片组。

> **Note**
>
> 创建数据库分片组之后，您将无法修改此设置。

6. 对于连接：

a. （可选）选择连接到 EC2 计算资源，然后选择现有 EC2 实例或创建新实例。

> **Note**
>
> 如果您连接到 EC2 实例，则无法公开访问该数据库分片组。

b. 对于网络类型，选择 IPv4 或双栈模式。

c. 选择虚拟私有云（VPC）和数据库子网组，或者使用默认设置。

> **Note**
>
> 如果在美国东部（弗吉尼亚州北部）区域创建 Limitless Database 数据库集群，请不要将 `us-east-1e` 可用区（AZ）包含在数据库子网组中。由于资源限制，`us-east-1e` 可用区不支持 Aurora Serverless v2，因此也不支持 Limitless Database。

d. 选择 VPC 安全组（防火墙），或使用默认设置。

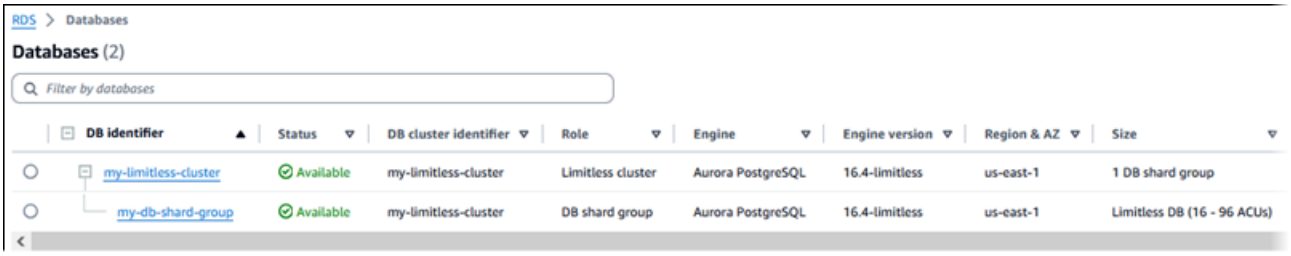
7. 对于数据库身份验证，选择密码身份验证或密码和 IAM 数据库身份验证。

8. 对于监控，请确保选中启用性能详情 和启用增强监控复选框。

对于性能详情，请选择至少 1 个月的保留时间。

9. 展开页面底部的其他配置。
10. 对于日志导出，请确保选中 PostgreSQL 日志复选框。
11. 根据需要指定其他设置。有关更多信息，请参阅 Aurora 数据库集群的设置。
12. 选择创建数据库。

创建主数据库集群和数据库分片组后，它们将显示在数据库页面上。



CLI

使用 AWS CLI 创建使用 Aurora PostgreSQL Limitless Database 的数据库集群时，您需要执行以下任务：

1. 创建主数据库集群.
2. 创建数据库分片组.

创建主数据库集群

创建数据库集群需要以下参数：

- `--db-cluster-identifier` – 数据库集群的名称。
- `--engine` – 数据库集群必须使用 `aurora-postgresql` 数据库引擎。
- `--engine-version` – 数据库集群必须使用的数据库引擎版本之一：
  - `16.4-limitless`
  - `16.6-limitless`
- `--storage-type` – 数据库集群必须使用 `aurora-iopt1` 数据库集群存储配置。
- `--cluster-scalability-type` – 指定 Aurora 数据库集群的可扩展性模式。设置为 `limitless` 时，集群将作为 Aurora PostgreSQL Limitless Database 运行。设置为 `standard`（默认）时，集群使用正常的数据库实例创建。

> **Note**
>
> 创建数据库集群之后，您将无法修改此设置。

- `--master-username` – 数据库集群的主用户名称。
- `--master-user-password` – 主用户的密码。
- `--enable-performance-insights` – 必须启用性能详情。
- `--performance-insights-retention-period` – 性能详情的保留期必须至少为 31 天。
- `--monitoring-interval` – 收集数据库集群的增强监控指标的时间点之间的间隔，以秒为单位。该值不能为 `0`。
- `--monitoring-role-arn` – 允许 RDS 将增强监控指标发送到 Amazon CloudWatch Logs 的 IAM 角色的 Amazon 资源名称（ARN）。

- `--enable-cloudwatch-logs-exports` – 您必须将 `postgresql` 日志导出到 CloudWatch 日志。

以下参数可选：

- `--db-subnet-group-name` – 要与该数据库集群关联的数据库子网组。这也决定了与数据库集群关联的 VPC。

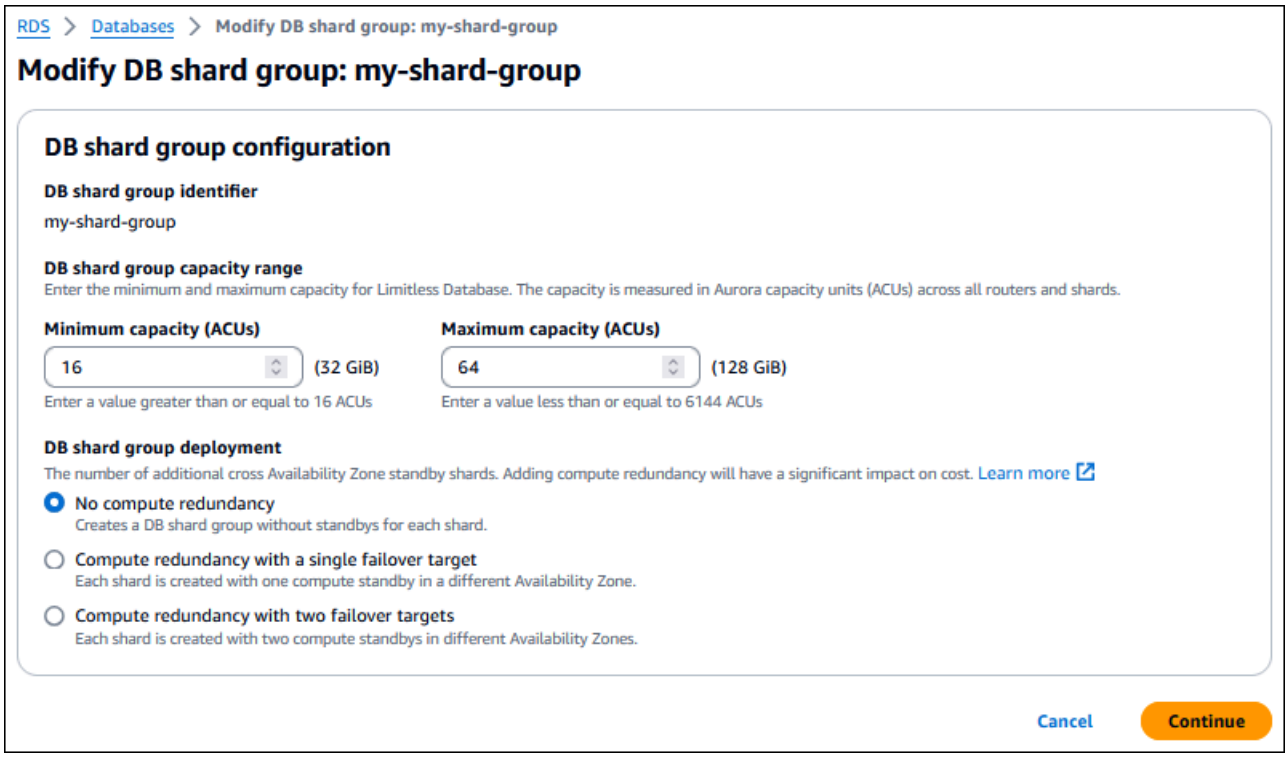
> **Note**
>
> 如果在美国东部（弗吉尼亚州北部）区域创建 Limitless Database 数据库集群，请不要将 `us-east-1e` 可用区（AZ）包含在数据库子网组中。由于资源限制，`us-east-1e` 可用区不支持 Aurora Serverless v2，因此也不支持 Limitless Database。

- `--vpc-security-group-ids` – 要与数据库集群关联的 VPC 安全组的列表。
- `--performance-insights-kms-key-id` – 用于加密性能详情数据的 AWS KMS key 标识符。如果未指定 KMS 密钥，则使用您 AWS 账户 的默认密钥。
- `--region` – 创建数据库集群的 AWS 区域。它必须支持 Aurora PostgreSQL Limitless Database。

要使用默认 VPC 和 VPC 安全组，请忽略 `--db-subnet-group-name` 和 `--vpc-security-group-ids` 选项。

创建主数据库集群

- ```
  aws rds create-db-cluster \
      --db-cluster-identifier my-limitless-cluster \
      --engine aurora-postgresql \
      --engine-version 16.6-limitless \
      --storage-type aurora-iopt1 \
      --cluster-scalability-type limitless \
      --master-username myuser \
      --master-user-password mypassword \
      --db-subnet-group-name mysubnetgroup \
      --vpc-security-group-ids sg-c7e5b0d2 \
      --enable-performance-insights \
      --performance-insights-retention-period 31 \
      --monitoring-interval 5 \
      --monitoring-role-arn arn:aws:iam::123456789012:role/EMrole \
  ```
  ```

```
    --enable-cloudwatch-logs-exports postgresql
```

有关更多信息，请参阅 create-db-cluster

创建数据库分片组

接下来，在刚创建的数据库集群中创建数据库分片组。以下参数为必需参数：

- `--db-shard-group-identifier` – 数据库分片组的名称。

  数据库分片组标识符具有以下限制：
  - 它在创建它的 AWS 区域和 AWS 账户中必须是唯一的。
  - 必须包含 1-63 个字母、数字或连字符。
  - 第一个字符必须是字母。
  - 它不能以连字符结尾，也不能包含两个连续连字符。
  -

    Important

    创建数据库分片组后，您将无法更改数据库集群标识符或数据库分片组标识符。
- `--db-cluster-identifier` – 要在其中创建数据库分片组的数据库集群的名称。
- `--max-acu` – 数据库分片组的最大容量。必须是 16-6144 ACU。如果容量限制高于 6144 ACU，请联系 AWS。

  初始路由器和分片数量由您在创建数据库分片组时设置的最大容量决定。最大容量越大，在数据库分片组中创建的路由器和分片的数量就越多。有关更多信息，请参阅 将数据库分片组的最大容量与创建的路由器和分片的数量相关联。

以下参数可选：

- `--compute-redundancy` – 是否为数据库分片组创建备用数据库。此参数可能具有以下值：
  - `0` – 为每个分片创建没有备用数据库的数据库分片组。这是默认值。
  - `1` – 在不同的可用区（AZ）中创建带有一个计算备用数据库的数据库分片组。
  - `2` – 在两个不同的可用区中创建带有两个计算备用数据库的数据库分片组。

  Note

  如果将计算冗余设置为非零值，则分片总数将增加一倍或两倍。这将产生额外费用。
  计算备用数据库中的节点会扩展和缩减到与写入器相同的容量。您无需为备用数据库单独设置容量范围。
- `--min-acu` – 数据库分片组的最小容量。它必须至少为 16 ACU，这是默认值。

- `--publicly-accessible|--no-publicly-accessible` – 是否向数据库分片组分配可公开访问的 IP 地址。对数据库分片组的访问由集群使用的安全组控制。

  默认值为 `--no-publicly-accessible`。

  > **Note**
  >
  > 创建数据库分片组之后，您将无法修改此设置。

创建数据库分片组

-

```
aws rds create-db-shard-group \
    --db-shard-group-identifier my-db-shard-group \
    --db-cluster-identifier my-limitless-cluster \
    --max-acu 1000
```

使用数据库分片组

您可以执行以下任务来添加和管理 Aurora PostgreSQL Limitless Database 的数据库分片组。

主题

- 连接到您的 Aurora PostgreSQL Limitless Database 数据库集群
- 查找数据库分片组中路由器和分片的数量
- 描述数据库分片组
- 重新启动数据库分片组
- 更改数据库分片组的容量
- 在数据库分片组中拆分一个分片
- 向数据库分片组中添加路由器
- 删除数据库分片组
- 向现有 Aurora PostgreSQL Limitless Database 数据库集群添加数据库分片组

连接到您的 Aurora PostgreSQL Limitless Database 数据库集群

要使用 Aurora PostgreSQL 无限制数据库，您需要连接到集群写入器或读取器端点。您可以使用 `psql` 或任何其他与 PostgreSQL 配合使用的连接实用程序：

```
$ psql -h DB_cluster_endpoint -p port_number -U database_username -d postgres_limitless
```

以下示例使用您在 CLI 中创建的数据库集群的端点。

```
$ psql -h my-limitless-cluster.cluster-ckifpdyyyxxx.us-east-1.rds.amazonaws.com -p 5432
 -U postgres -d postgres_limitless
```

> **Note**
>
> Aurora PostgreSQL Limitless Database 中数据库分片组的默认数据库为 `postgres_limitless`。

使用 Limitless Connection Plugin

连接到 Aurora PostgreSQL Limitless Database 时，客户端使用集群端点进行连接，并通过 Amazon Route 53 路由到事务路由器。但是，Route 53 的负载均衡能力有限，可能导致事务路由器上的工作负载不均衡。AWS JDBC 驱动程序的 Limitless Connection Plugin 通过使用负载感知执行客户端负载均衡来解决这个问题。有关 AWS JDBC 驱动程序的更多信息，请参阅使用 Amazon Web Services (AWS) JDBC 驱动程序连接到 Aurora PostgreSQL。

查找数据库分片组中路由器和分片的数量

您可以使用以下查询来查找路由器和分片的数量：

```
SELECT * FROM rds_aurora.limitless_subclusters;

 subcluster_id | subcluster_type
---------------+-----------------
 1             | router
 2             | router
 3             | shard
 4             | shard
 5             | shard
 6             | shard
```

描述数据库分片组

使用 describe-db-shard-groups AWS CLI 命令描述您的数据库分片组。以下参数为可选参数：

- --db-shard-group-identifier – 数据库分片组的名称。

以下示例描述了一个特定的数据库分片组。

```
aws rds describe-db-shard-groups --db-shard-group-identifier my-db-shard-group
```

输出与以下内容类似。

```
{
    "DBShardGroups": [
        {
            "DBShardGroupResourceId": "shardgroup-8986d309a93c4da1b1455add17abcdef",
            "DBShardGroupIdentifier": "my-shard-group",
            "DBClusterIdentifier": "my-limitless-cluster",
            "MaxACU": 1000.0,
            "ComputeRedundancy": 0,
            "Status": "available",
            "PubliclyAccessible": false,
            "Endpoint": "my-limitless-cluster.limitless-ccetp2abcdef.us-
east-1.rds.amazonaws.com"
        }
    ]
```

```
}
```

重新启动数据库分片组

有时，您必须重新启动数据库分片组，例如，当 `max_connections` 参数因最大容量变化而发生变化时。

您可以使用 AWS Management Console或 AWS CLI 来更改数据库分片组的容量。

控制台

使用以下流程。

登录 AWS Management Console并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。

1. 导航到数据库页面。
2. 选择需要重新启动的数据库分片组。
3. 对于操作，选择重启。
4. 选择确认。

CLI

要重新启动数据库分片组，请使用 `reboot-db-shard-group` AWS CLI 命令及以下参数：

- `--db-shard-group-identifier` – 数据库分片组的名称。

以下示例重新启动数据库分片组。

```
aws rds reboot-db-shard-group --db-shard-group-identifier my-db-shard-group
```

更改数据库分片组的容量

您可以使用 AWS Management Console或 AWS CLI 来更改数据库分片组的容量。

控制台

使用以下流程。

登录 AWS Management Console并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。

1. 导航到数据库页面。
2. 选择需要修改的数据库分片组。
3. 对于 Actions（操作），选择 Modify（修改）。

   将显示修改数据库分片组页面。



4. 输入新的最小容量（ACU）值，例如 **100**。
5. 输入新的最大容量（ACU）值，例如 **1000**。
6. 选择继续。

   将显示确认页面上，其中包含更改摘要。

7. 查看您的更改，然后选择修改数据库分片组。

CLI

结合下列参数使用 `modify-db-shard-group` AWS CLI 命令：

- `--db-shard-group-identifier` – 数据库分片组的名称。
- `--max-acu` – 数据库分片组新的最大容量。您可以将数据库分片组的最大容量设置为 16-6144 ACU。如果容量限制高于 6144 ACU，请联系 AWS。

  路由器和分片的数量不会变化。
- `--min-acu` – 数据库分片组新的最小容量。它必须至少为 16 ACU，这是默认值。

以下 CLI 示例将数据库分片组的容量范围更改为 100-1000 ACU。

```
aws rds modify-db-shard-group \
    --db-shard-group-identifier my-db-shard-group \
    --min-acu 100 \
    --max-acu 1000
```

在数据库分片组中拆分一个分片

您可以手动将数据库分片组中的分片拆分为两个较小的分片。这称为用户发起的分片拆分。

当分片的数据量非常大或使用率非常高时，Aurora PostgreSQL Limitless Database 也可以对其进行拆分。这称为系统发起的分片拆分。

主题

- 先决条件
- 拆分分片
- 跟踪分片拆分
- 结束分片拆分
- 取消分片拆分

先决条件

用户发起的分片拆分有以下先决条件：

- 您必须拥有一个数据库分片组。
- 数据库分片组不能为空：它必须至少包含一个分片表。
- 用户必须具有 rds_aurora_limitless_cluster_admin 权限。rds_superuser 拥有此权限；因此主用户也拥有该权限。rds_superuser 可以将该权限授予其他用户：

```
/* Logged in as the master user or a user with rds_superuser privileges */
CREATE USER username;
GRANT rds_aurora_limitless_cluster_admin to username;
```

- 您必须知道要拆分的分片的子集群（节点）ID。您可以通过以下查询获取 ID：

```
SELECT * FROM rds_aurora.limitless_subclusters;

 subcluster_id | subcluster_type
---------------+-----------------
 1             | router
 2             | router
 3             | shard
 4             | shard
 5             | shard
```

```
6             | shard
```

要启用系统发起的分片拆分，请在与数据库集群相关联的自定义数据库集群参数组中设置以下数据库集群参数：

参数	值
rds_aurora.limitless_enable_auto_scale	on
rds_aurora.limitless_auto_scale_options	split_shard 或 add_router,split_shard
rds_aurora.limitless_finalize_split_shard_mode	该参数决定系统发起的分片拆分如何结束。值可以是以下之一： • user_initiated – 由您决定何时结束分片拆分。这是默认值。 • immediate – 立即结束分片拆分。 有关更多信息，请参阅 结束分片拆分。 **Note** 此参数仅适用于系统发起的分片拆分。

有关更多信息，请参阅 Amazon Aurora 数据库集群的数据库集群参数组。

拆分分片

要拆分数据库分片组中的分片，请使用 rds_aurora.limitless_split_shard 函数。该函数启动异步运行的分片拆分任务。

```
SELECT rds_aurora.limitless_split_shard('subcluster_id');
```

在成功提交任务后，等待返回的任务 ID，例如：

```
SELECT rds_aurora.limitless_split_shard('3');

    job_id
---------------
 1691300000000
(1 row)
```

> **Note**
>
> 不支持并发分片拆分操作。按顺序执行每个操作并完成每个操作，然后再启动另一个添加操作。

跟踪分片拆分

您可以使用任务 ID 跟踪分片拆分任务。要描述特定任务并获取有关它的更多详细信息，请运行以下查询：

```
SELECT * FROM rds_aurora.limitless_list_shard_scale_jobs(job_id);
```

例如：

```
SELECT * FROM rds_aurora.limitless_list_shard_scale_jobs(1691300000000);

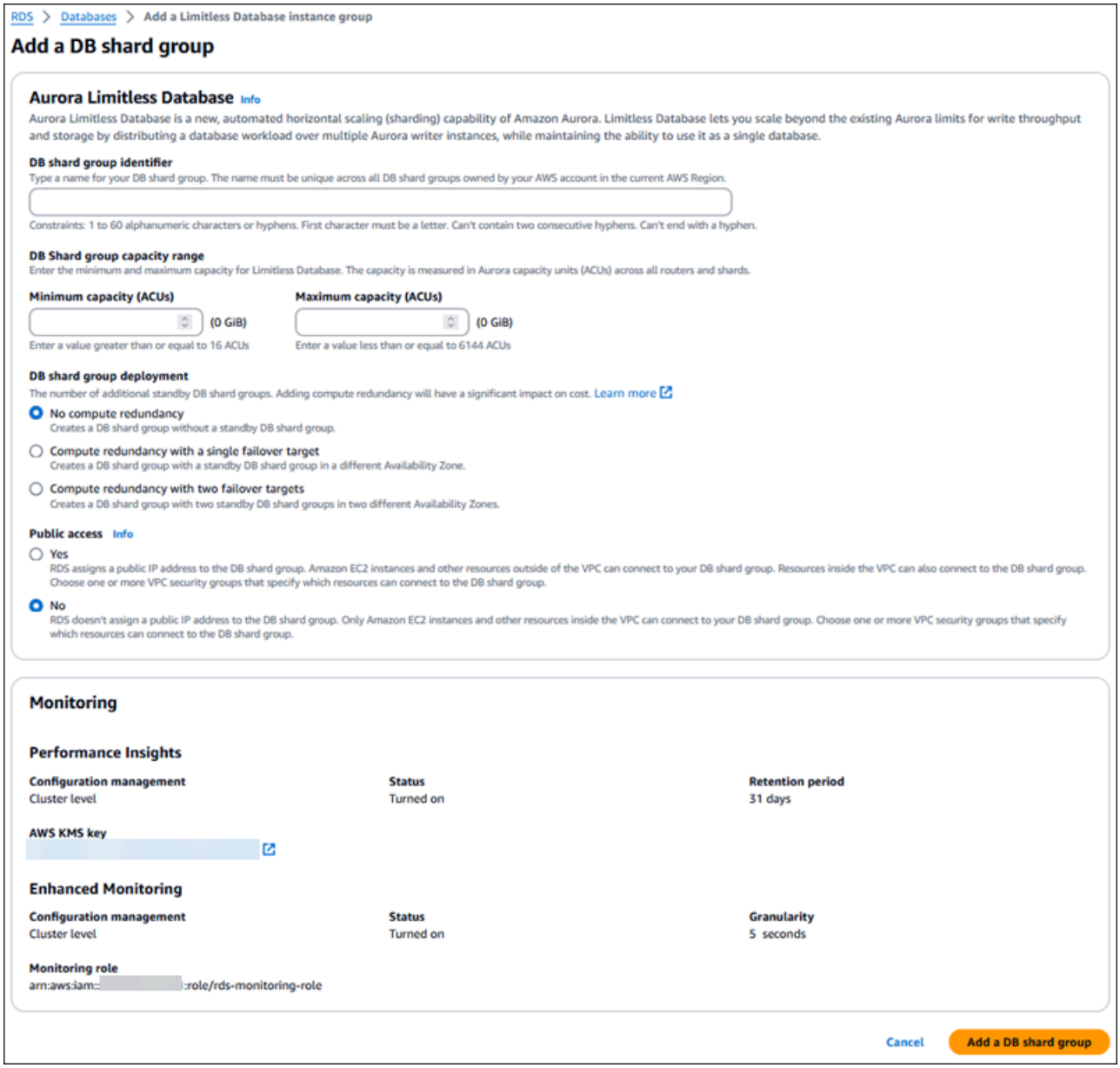
    job_id     |    action   |      job_details      | status  |    submission_time
 |                  message
---------------+-------------+-----------------------+---------
+------------------------+-------------------------------------------
 1691300000000 | SPLIT_SHARD | Split Shard 3 by User | SUCCESS | 2023-08-06 05:33:20+00
 | Scaling job succeeded.                 +
               |             |                       |         |
 | New shard instance with ID 7 was created.
(1 row)
```

当您将不存在的任务作为输入传递时，查询会返回错误。

```
SELECT * from rds_aurora.limitless_list_shard_scale_jobs(1691300000001);

ERROR:  no job found with the job ID provided
```

您可以使用不带任务 ID 的相同查询来跟踪所有分片拆分任务的状态，例如：

```
SELECT * FROM rds_aurora.limitless_list_shard_scale_jobs();

    job_id     |   action    |  job_details          |   status    |    submission_time
     |                  message
---------------+-------------+-----------------------+-------------
+------------------------
+-------------------------------------------------------------
 1691200000000 | SPLIT_SHARD | Split Shard 3 by User | IN_PROGRESS | 2023-08-05
 01:46:40+00 |
 1691300000000 | SPLIT_SHARD | Split Shard 4 by User | SUCCESS     | 2023-08-06
 05:33:20+00 | Scaling job succeeded. +
               |             |                       |             |
     | New shard instance with ID 7 was created.
 1691400000000 | SPLIT_SHARD | Split Shard 5 by User | FAILED      | 2023-08-07
 09:20:00+00 | Error occurred for the add shard job 1691400000000.
               |             |                       |             |
     | Retry the command. If the issue persists, contact AWS Support.
 1691500000000 | SPLIT_SHARD | Split Shard 5 by User | CANCELED    | 2023-08-07
 09:20:00+00 | Scaling job was cancelled.
(4 rows)
```

任务状态可以是下列项之一：

- IN_PROGRESS – 分片拆分任务已提交且正在进行中。同一时间只能有一个任务在进行中。
- PENDING – 分片拆分任务正等待您结束。有关更多信息，请参阅 结束分片拆分。
- CANCELLATION_IN_PROGRESS – 用户正在取消分片拆分任务。
- CANCELED – 用户或系统已成功取消分片拆分任务。
- SUCCESS – 分片拆分任务已成功完成。该 message 字段包含新分片的实例 ID。
- FAILED – 分片拆分任务失败。该 message 字段包含失败的详细信息以及可对失败任务采取的任何后续操作。

结束分片拆分

结束是分片拆分流程的最后一步。它会产生一些停机时间。如果启动分片拆分任务，则会在任务成功完成后立即结束。

有时，当您使用 rds_aurora.limitless_enable_auto_scale 参数启用系统发起的分片拆分时，系统会根据工作负载拆分分片。

在这种情况下，您可以选择是立即结束还是按您选择的时间结束。您可以使用 rds_aurora.limitless_finalize_split_shard_mode 数据库集群参数来选择何时结束：

- 如果将该值设置为 immediate，则会立即结束。
- 如果将该值设置为 user_initiated，则必须手动结束分片拆分任务。

  系统会向您发送一个 RDS 事件，并将分片拆分任务的状态设置为 PENDING。

如果设置为 user_initiated，则使用 rds_aurora.limitless_finalize_split_shard 函数结束分片拆分任务：

```
SELECT * FROM rds_aurora.limitless_finalize_split_shard(job_id);
```

> **Note**
>
> 此功能仅适用于由系统发起的分片拆分，不适用于用户发起的分片拆分。

取消分片拆分

您可以取消用户发起或系统发起的分片拆分，即 IN_PROGRESS 或 PENDING。您需要任务 ID 才能将其取消。

```
SELECT * from rds_aurora.limitless_cancel_shard_scale_jobs(job_id);
```

除非出现错误，否则不会返回任何输出。您可以使用任务跟踪查询来跟踪取消情况。

向数据库分片组中添加路由器

您可以向数据库分片组中添加路由器。

主题

- 先决条件
- 添加路由器
- 追踪路由器添加
- 取消路由器添加

先决条件

添加路由器具有以下先决条件：

- 您必须拥有一个数据库分片组。
- 用户必须具有 rds_aurora_limitless_cluster_admin 权限。rds_superuser 拥有此权限；因此主用户也拥有该权限。rds_superuser 可以将该权限授予其他用户：

```
/* Logged in as the master user or a user with rds_superuser privileges */
CREATE USER username;
GRANT rds_aurora_limitless_cluster_admin to username;
```

> **Note**
>
> 如果您在创建数据库分片组后更改了 AWS 账户 的默认 CA 证书，新路由器将使用与现有路由器的 CA 证书不同的新 CA 证书。根据信任存储的不同，某些连接可能会失败。

- 要启用系统启动的路由器添加功能，请在与数据库集群相关联的自定义数据库集群参数组中设置以下数据库集群参数：

参数	值
rds_aurora.limitless_enable_auto_scale	on
rds_aurora.limitless_auto_scale_options	add_router 或 add_router,split_shard

有关更多信息，请参阅 Amazon Aurora 数据库集群的数据库集群参数组。

添加路由器

要添加路由器，请使用 `rds_aurora.limitless_add_router` 函数。该函数启动异步运行的路由器添加任务。

```
SELECT rds_aurora.limitless_add_router();
```

在成功提交任务后，等待返回的任务 ID，例如：

```
    job_id
---------------
 1691300000000
(1 row)
```

> **Note**
>
> 不支持并发路由器添加操作。按顺序执行操作并完成每个操作，然后再启动另一个添加操作。

追踪路由器添加

您可以使用任务 ID 来跟踪路由器添加任务。要描述特定任务并获取有关它的更多详细信息，请运行以下查询：

```
SELECT * FROM rds_aurora.limitless_list_router_scale_jobs(job_id);
```

例如：

```
SELECT * FROM rds_aurora.limitless_list_router_scale_jobs(1691300000000);

    job_id     |   action   |        job_details       | status  |    submission_time
   |                  message
---------------+------------+--------------------------+---------
+------------------------+-----------------------------------------
 1691300000000 | ADD_ROUTER | Add 1 new Router by User | SUCCESS | 2023-08-06
 05:33:20+00 | Scaling job succeeded.                  +
```

```
               |            |                            |         |
   | New router instance with ID 7 was created.
(1 row)
```

当您将不存在的任务作为输入传递时，查询会返回错误。

```
SELECT * from rds_aurora.limitless_list_router_scale_jobs(1691300000001);

ERROR:  no job found with the job ID provided
```

您可以使用不带任务 ID 的相同查询来跟踪所有路由器添加任务的状态，例如：

```
SELECT * FROM rds_aurora.limitless_list_router_scale_jobs();

    job_id     |   action   |        job_details       |   status    |
 submission_time     |                  message
---------------+------------+--------------------------+-------------
+-----------------------+-------------------------------------------
 1691200000000 | ADD_ROUTER | Add 1 new Router by User | IN_PROGRESS | 2023-08-05
 01:46:40+00 |
 1691300000000 | ADD_ROUTER | Add 1 new Router by User | SUCCESS     | 2023-08-06
 05:33:20+00 | Scaling job succeeded.                 +
               |            |                            |             |
       | New router instance with ID 7 was created.
 1691400000000 | ADD_ROUTER | Add 1 new Router by User | FAILED      | 2023-08-07
 09:20:00+00 | Error occurred for the add router job 1691400000000.
               |            |                            |             |
       | Retry the command. If the issue persists, contact AWS Support.
 1691500000000 | ADD_ROUTER | Add 1 new Router by User | CANCELED    | 2023-08-07
 09:20:00+00 | Scaling job was cancelled.
(4 rows)
```

任务状态可以是下列项之一：

- IN_PROGRESS – 路由器添加任务已提交，正在进行中。同一时间只能有一个任务在进行中。
- CANCELLATION_IN_PROGRESS – 用户正在取消路由器添加任务。
- CANCELED – 用户或系统已成功取消路由器添加任务。
- SUCCESS – 路由器添加任务已成功完成。该 message 字段包含新路由器的实例 ID。
- FAILED – 路由器添加任务失败。该 message 字段包含失败的详细信息以及可对失败任务采取的任何后续操作。

> **Note**
>
> 没有 PENDING 状态，因为不需要最终确定路由器添加任务。它们不会造成停机。

取消路由器添加

您可以取消处于 IN_PROGRESS 状态的路由器添加任务。您需要任务 ID 才能将其取消。

```
SELECT * from rds_aurora.limitless_cancel_router_scale_jobs(job_id);
```

除非出现错误，否则不会返回任何输出。您可以使用任务跟踪查询来跟踪取消情况。

删除数据库分片组

如有必要，您可以删除数据库分片组。删除数据库分片组会删除计算节点（分片和路由器），但不会删除存储。

控制台

登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。

1. 导航到数据库页面。
2. 选择需要删除的数据库分片组。
3. 对于操作，选择删除。
4. 在方框中输入 **`delete me`**，然后选择删除。

数据库分片组已删除。

AWS CLI

要删除您的数据库分片组，请使用 `delete-db-shard-group` AWS CLI 命令及以下参数：

- `--db-shard-group-identifier` – 数据库分片组的名称。

以下示例将删除之前创建的 Aurora PostgreSQL 数据库集群中的数据库分片组。

```
aws rds delete-db-shard-group --db-shard-group-identifier my-db-shard-group
```

向现有 Aurora PostgreSQL Limitless Database 数据库集群添加数据库分片组

您可以在现有数据库集群中创建数据库分片组，例如，如果您正在恢复数据库集群或已删除该数据库分片组。

有关主数据库集群和数据库分片组要求的更多信息，请参阅 Aurora PostgreSQL Limitless Database 要求和注意事项。

> **Note**
>
> 每个集群只能有一个数据库分片组。
>
> 在创建数据库分片组之前，Limitless Database 数据库集群必须处于 `available` 状态。

控制台

您可以使用 AWS Management Console 向现有数据库集群添加数据库分片组。

添加数据库分片组

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 导航到数据库页面。
3. 选择要向其添加数据库分片组的 Limitless Database 数据库集群。
4. 对于操作，选择添加数据库分片组。



5. 输入数据库分片组标识符。

> **Important**
>
> 创建数据库分片组后，您将无法更改数据库集群标识符或数据库分片组标识符。

6. 输入最小容量（ACU）。最小值为 16 ACU。
7. 输入最大容量（ACU）。使用介于 16–6144 ACU 之间的值。

   有关更多信息，请参阅 将数据库分片组的最大容量与创建的路由器和分片的数量相关联。
8. 对于数据库分片组部署，选择是否为数据库分片组创建备用数据库：

- 没有计算冗余 – 为每个分片创建没有备用数据库的数据库分片组。这是默认值。
- 使用单个失效转移目标计算冗余 – 在不同的可用区（AZ）中创建带有一个计算备用数据库的数据库分片组。
- 使用两个失效转移目标计算冗余 – 在两个不同的可用区中创建带有两个计算备用数据库的数据库分片组。

9. 选择是否可以公开访问数据库分片组。

> **Note**
>
> 创建数据库分片组之后，您将无法修改此设置。

10. 选择添加数据库分片组。

AWS CLI

使用 `create-db-shard-group` AWS CLI 命令创建数据库分片组。

以下参数为必需参数：

- `--db-cluster-identifier` – 数据库分片组所属的数据库集群。
- `--db-shard-group-identifier` – 数据库分片组的名称。

  数据库分片组标识符具有以下限制：

  - 它在创建它的 AWS 区域和 AWS 账户中必须是唯一的。
  - 必须包含 1-63 个字母、数字或连字符。
  - 第一个字符必须是字母。
  - 它不能以连字符结尾，也不能包含两个连续连字符。

  > **Important**
  > 创建数据库分片组后，您将无法更改数据库集群标识符或数据库分片组标识符。

- `--max-acu` – 数据库分片组的最大容量。使用介于 16–6144 ACU 之间的值。

以下参数可选：

- `--compute-redundancy` – 是否为数据库分片组创建备用数据库。此参数可能具有以下值：
  - `0` – 为每个分片创建没有备用数据库的数据库分片组。这是默认值。
  - `1` – 在不同的可用区（AZ）中创建带有一个计算备用数据库的数据库分片组。
  - `2` – 在两个不同的可用区中创建带有两个计算备用数据库的数据库分片组。

  > **Note**
  > 如果将计算冗余设置为非零值，则节点总数将增加一倍或两倍。这将产生额外费用。

- `--min-acu` – 数据库分片组的最小容量。它必须至少为 16 ACU，这是默认值。
- `--publicly-accessible|--no-publicly-accessible` – 是否向数据库分片组分配可公开访问的 IP 地址。对数据库分片组的访问由集群使用的安全组控制。

  默认为 `--no-publicly-accessible`。

> **Note**
>
> 创建数据库分片组之后，您将无法修改此设置。

以下示例在 Aurora PostgreSQL 数据库集群中创建了一个数据库分片组。

```
aws rds create-db-shard-group \
    --db-cluster-identifier my-db-cluster \
    --db-shard-group-identifier my-new-shard-group \
    --max-acu 1000
```

输出与以下内容类似。

```
{
    "Status": "CREATING",
    "Endpoint": "my-db-cluster.limitless-ckifpdyyyxxx.us-east-1.rds.amazonaws.com",
    "PubliclyAccessible": false,
    "DBClusterIdentifier": "my-db-cluster",
    "MaxACU": 1000.0,
    "DBShardGroupIdentifier": "my-new-shard-group",
    "DBShardGroupResourceId": "shardgroup-8986d309a93c4da1b1455add17abcdef",
    "ComputeRedundancy": 0
}
```

创建 Aurora PostgreSQL Limitless Database 表

在 Aurora PostgreSQL Limitless Database 中，包含您数据的表有三种类型：

- 标准表 – 这是 Aurora PostgreSQL Limitless Database 中的默认表类型。您可以使用 CREATE TABLE 命令创建标准表，并可对其运行数据定义语言（DDL）和数据操作语言（DML）操作。

  标准表不是分布式表。它们存储在系统在内部选择的一个分片上。
- 分片表 – 这些表分布在多个分片中。数据根据表中指定列的值在分片之间进行拆分。这组列称为分片键。
- 引用表 – 这些表在所有分片上复制。它们用于不经常修改的引用数据，例如产品目录和邮政编码。

  引用表和分片表之间的联接查询可在分片上运行，从而消除分片和路由器之间不必要的数据移动。

可通过两种方式创建无限表：

- 使用变量创建无限表 – 若要创建新的分片表和引用表，请使用此方法。
- 将标准表转换为无限表 – 如果要将现有标准表转换为分片表和引用表，请使用此方法。

我们还为 Aurora PostgreSQL Limitless Database 提供了架构示例。

使用变量创建无限表

通过设置表创建模式，您可以使用变量创建分片表和引用表。然后，您创建的表将使用此模式，直到您设置不同的模式。

使用以下变量创建分片表和引用表：

- `rds_aurora.limitless_create_table_mode` – 将此会话变量设置为 `sharded` 或 `reference`。该变量的默认值为 `standard`。
- `rds_aurora.limitless_create_table_shard_key` – 将此会话变量设置为列名数组以用作分片键。如果 `rds_aurora.limitless_create_table_mode` 不是 `sharded`，则忽略此变量。

  将值格式化为 `untyped array literal`，类似于在数组列中插入字面值。有关更多信息，请参阅 PostgreSQL 文档中的数组。
- `rds_aurora.limitless_create_table_collocate_with` – 将此会话变量设置为特定的表名，以将新创建的表与该表并置。

  如果使用相同的分片键对两个或多个表进行分片，则可以显式对齐（并置）这些表。当两个或多个表并置时，这些表中具有相同分片键值的行将放在同一个分片上。并置有助于将某些操作限制在单个分片上，从而提高性能。

> **Note**
>
> 所有主键和唯一键都必须包含分片键。这意味着分片键是主键或唯一键的子集。
> 无限表有一些限制。有关更多信息，请参阅 Aurora PostgreSQL Limitless Database 的 DDL 限制和其他信息。

主题

- 使用变量创建无限表的示例
- Aurora PostgreSQL Limitless Database 表视图

使用变量创建无限表的示例

以下示例演示了如何使用这些变量创建分片表和引用表。

创建一个名为 items 的分片表，带有分片键 id。

```
BEGIN;
SET LOCAL rds_aurora.limitless_create_table_mode='sharded';
SET LOCAL rds_aurora.limitless_create_table_shard_key='{"id"}';
CREATE TABLE items(id int, val int, item text);
COMMIT;
```

创建一个名为 items 的分片表，其分片键由 item_id 和 item_cat 列组成。

```
BEGIN;
SET LOCAL rds_aurora.limitless_create_table_mode='sharded';
SET LOCAL rds_aurora.limitless_create_table_shard_key='{"item_id", "item_cat"}';
CREATE TABLE items(item_id int, item_cat varchar, val int, item text);
COMMIT;
```

创建一个名为 item_description 的分片表，其分片键由 item_id 和 item_cat 列组成，并将其与上一个示例中的 items 表并置。

```
BEGIN;
SET LOCAL rds_aurora.limitless_create_table_mode='sharded';
SET LOCAL rds_aurora.limitless_create_table_shard_key='{"item_id", "item_cat"}';
SET LOCAL rds_aurora.limitless_create_table_collocate_with='items';
CREATE TABLE item_description(item_id int, item_cat varchar, color_id int);
COMMIT;
```

创建一个名为 colors 的引用表。

```
BEGIN;
SET LOCAL rds_aurora.limitless_create_table_mode='reference';
CREATE TABLE colors(color_id int primary key, color varchar);
COMMIT;
```

要将 rds_aurora.limitless_create_table_mode 会话变量重置为 standard，请使用以下语句：

```
RESET rds_aurora.limitless_create_table_mode;
```

重置此变量后，会默认将表创建为标准表。有关标准表的更多信息，请参阅将标准表转换为无限表。

Aurora PostgreSQL Limitless Database 表视图

您可以使用以下视图查找有关 Limitless Database 表的信息。

rds_aurora.limitless_tables

该 `rds_aurora.limitless_tables` 视图包含有关无限表及其类型的信息。

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_tables;

 table_gid | local_oid | schema_name | table_name  | table_status | table_type  |
 distribution_key
-----------+-----------+-------------+-------------+--------------+-------------
+------------------
         5 |     18635 | public      | standard    | active       | standard    |
         6 |     18641 | public      | ref         | active       | reference   |
         7 |     18797 | public      | orders      | active       | sharded     |
 HASH (order_id)
         2 |     18579 | public      | customer    | active       | sharded     |
 HASH (cust_id)
(4 rows)
```

rds_aurora.limitless_table_collocations

该 `rds_aurora.limitless_table_collocations` 视图包含有关并置的分片表的信息。例如，`orders` 和 `customers` 表并置，它们具有相同的 `collocation_id`。`users` 和 `followers` 表并置，它们具有相同的 `collocation_id`。

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_table_collocations ORDER BY
 collocation_id;

 collocation_id | schema_name | table_name
----------------+-------------+------------
          16002 | public      | orders
          16002 | public      | customers
          16005 | public      | users
          16005 | public      | followers
(4 rows)
```

rds_aurora.limitless_table_collocation_distributions

`rds_aurora.limitless_table_collocation_distributions` 显示了每个并置的密钥分配几率。

```
postgres_limitless=> SELECT * FROM
 rds_aurora.limitless_table_collocation_distributions ORDER BY collocation_id,
 lower_bound;

 collocation_id | subcluster_id |     lower_bound      |     upper_bound
----------------+---------------+----------------------+----------------------
          16002 |             6 | -9223372036854775808 | -4611686018427387904
          16002 |             5 | -4611686018427387904 |                    0
          16002 |             4 |                    0 |  4611686018427387904
          16002 |             3 |  4611686018427387904 |  9223372036854775807
          16005 |             6 | -9223372036854775808 | -4611686018427387904
          16005 |             5 | -4611686018427387904 |                    0
          16005 |             4 |                    0 |  4611686018427387904
          16005 |             3 |  4611686018427387904 |  9223372036854775807
(8 rows)
```

将标准表转换为无限表

您可以将标准表转换为分片表或引用表。在转换过程中，数据从标准表移动到分布式表，然后删除源标准表。使用 INSERT INTO SELECT FROM 命令移动数据。

目录

- 创建分片表
- 创建并置表
- 创建引用表

创建分片表

您可以通过在标准表上运行 rds_aurora.limitless_alter_table_type_sharded 流程来创建分片表。此流程采用标准表和列列表，然后使用列列表作为分片键来分配给定表。该流程同步运行，并在表上获取 ACCESS EXCLUSIVE 锁。

成功完成该流程后，源标准表将被删除，具有相同名称的分片表将变为可用。

rds_aurora.limitless_alter_table_type_sharded 流程使用以下语法：

```
postgres=> CALL rds_aurora.limitless_alter_table_type_sharded('schema.table',
 ARRAY['shard_key1', 'shard_key2', ... 'shard_keyn']);
```

该流程需要以下参数：

- schema – 包含要分片的表的数据库架构。如果未指定架构，该流程将使用 search_path。
- table – 要分片的表。
- shard_key*n* – 用作分片键的表列数组。

  分片键值是字符串文字，因此要区分大小写。如果分片键包含单引号 (')，请使用另一个单引号对其进行转义。例如，如果表列被命名为 customer's id，则使用 customer''s id 作为分片键。反斜杠 (\) 和双引号 (“) 无需转义。

> **Note**
>
> 所有主键和唯一键都必须包含分片键。这意味着分片键是主键或唯一键的子集。
> 在分片表中，CHECK 约束条件不支持表达式。

有关更多信息，请参阅 约束。

创建分片表

以下示例说明如何使用分片键 customer_id 创建 customer 分片表。

1. 创建标准表。

```
CREATE TABLE customer (customer_id INT PRIMARY KEY NOT NULL, zipcode INT, email
 VARCHAR);
```

2. 将标准表转换为分片表。

```
postgres=> CALL rds_aurora.limitless_alter_table_type_sharded('public.customer',
 ARRAY['customer_id']);

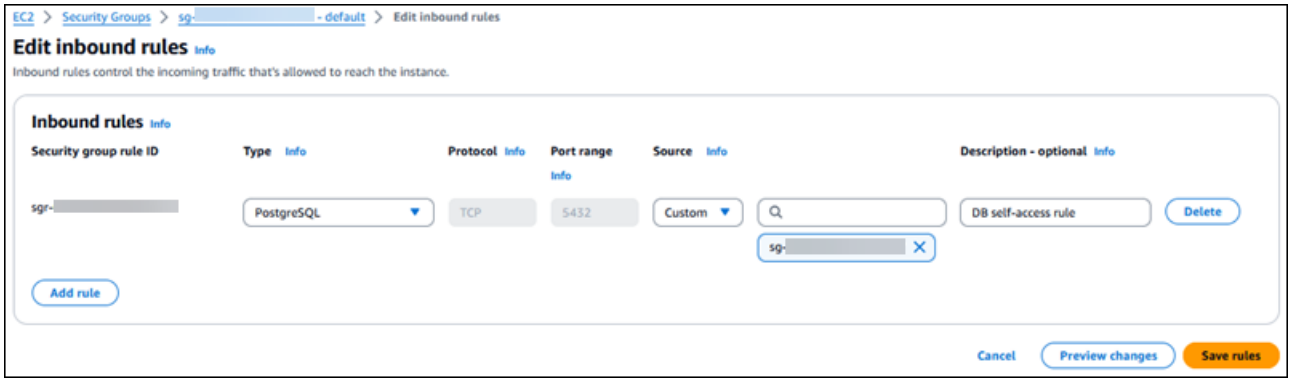
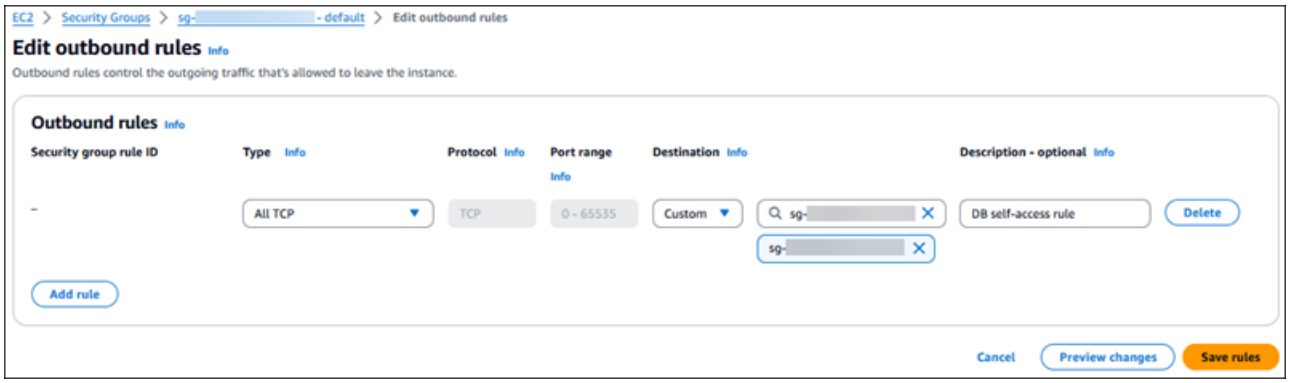
postgres=> \d

                    List of relations
 Schema |     Name     |       Type        |       Owner
--------+--------------+-------------------+--------------------
 public | customer     | partitioned table | postgres_limitless
 public | customer_fs1 | foreign table     | postgres_limitless
 public | customer_fs2 | foreign table     | postgres_limitless
 public | customer_fs3 | foreign table     | postgres_limitless
 public | customer_fs4 | foreign table     | postgres_limitless
 public | customer_fs5 | foreign table     | postgres_limitless
(6 rows)
```

创建并置表

如果使用相同的分片键对两个或多个表进行分片，则可以显式对齐（并置）这些表。当两个或多个表并置时，这些表中具有相同分片键值的行将放在同一个分片上。并置有助于将某些操作限制在单个分片上，从而提高性能。

您可以使用具有以下语法的 rds_aurora.limitless_alter_table_type_sharded 流程：

```
postgres=> CALL
 rds_aurora.limitless_alter_table_type_sharded('schema.collocated_table',
 ARRAY['shard_key1', 'shard_key2', ... 'shard_keyn'], 'schema.sharded_table');
```

该流程需要以下参数：

- schema – 包含要并置的表的数据库架构。如果未指定架构，该流程将使用 search_path。
- collocated_table – 要并置的表。
- shard_key*n* – 用作分片键的表列数组。

  必须使用与原始分片表相同的分片键，包括相同的列名和列类型。
- sharded_table – 与 collocated_table 并置的分片表。

创建并置表

1. 按照 创建分片表 中的流程创建第一个分片表。
2. 为并置表创建标准表。

```
CREATE TABLE mytable2 (customer_id INT PRIMARY KEY NOT NULL, column1 INT, column2
 VARCHAR);
```

3. 将标准表转换为并置表。

```
postgres=> CALL rds_aurora.limitless_alter_table_type_sharded('public.mytable2',
ARRAY['customer_id'], 'public.customer');

postgres=> \d

                   List of relations
 Schema |     Name     |       Type        |       Owner
--------+--------------+-------------------+--------------------
 public | customer     | partitioned table | postgres_limitless
 public | customer_fs1 | foreign table     | postgres_limitless
 public | customer_fs2 | foreign table     | postgres_limitless
 public | customer_fs3 | foreign table     | postgres_limitless
 public | customer_fs4 | foreign table     | postgres_limitless
 public | customer_fs5 | foreign table     | postgres_limitless
 public | mytable2     | partitioned table | postgres_limitless
 public | mytable2_fs1 | foreign table     | postgres_limitless
 public | mytable2_fs2 | foreign table     | postgres_limitless
 public | mytable2_fs3 | foreign table     | postgres_limitless
 public | mytable2_fs4 | foreign table     | postgres_limitless
 public | mytable2_fs5 | foreign table     | postgres_limitless
(12 rows)
```

创建引用表

可以通过在标准表上运行 rds_aurora.limitless_alter_table_type_reference 流程来创建引用表。此流程将给定表复制到数据库分片组中的所有分片，并将表类型更改为引用。该流程同步运行，并在表上获取 ACCESS EXCLUSIVE 锁。

成功完成该流程后，源标准表将被删除，具有相同名称的引用表将变为可用。

rds_aurora.limitless_alter_table_type_reference 流程使用以下语法：

```
postgres=> CALL rds_aurora.limitless_alter_table_type_reference('schema.table');
```

存储过程需要以下参数：

- schema – 包含要复制的表的数据库架构。如果未指定架构，该流程将使用 search_path。
- table – 要复制的表。

> **Note**
>
> 创建引用表的标准表必须具有主键。
> 在引用表中，CHECK 约束条件不支持表达式。
> 之前的函数 limitless_table_alter_type_reference 已被弃用。

创建引用表

以下示例演示如何创建 zipcodes 引用表。

1. 创建标准表。

```
CREATE TABLE zipcodes (zipcode INT PRIMARY KEY, details VARCHAR);
```

2. 将标准表转换为引用表。

```
CALL rds_aurora.limitless_alter_table_type_reference('public.zipcodes');

postgres=> \d

                    List of relations
 Schema |      Name      |       Type        |       Owner
```

```
--------+--------------+-------------------+--------------------
 public | customer     | partitioned table | postgres_limitless
 public | customer_fs1 | foreign table     | postgres_limitless
 public | customer_fs2 | foreign table     | postgres_limitless
 public | customer_fs3 | foreign table     | postgres_limitless
 public | customer_fs4 | foreign table     | postgres_limitless
 public | customer_fs5 | foreign table     | postgres_limitless
 public | zipcodes     | foreign table     | postgres_limitless
(7 rows)
```

输出显示了 `customer` 分片表和 `zipcodes` 引用表。

Aurora PostgreSQL Limitless Database 示例架构

我们为 Aurora PostgreSQL Limitless Database 提供了以下示例架构：

- 无限 E-Commerce 示例架构
- 无限 pgbench

您可以使用这些架构快速创建示例数据库并将数据加载到 Aurora PostgreSQL Limitless Database 表中。有关更多信息，请参阅 GitHub 存储库。

将数据加载到 Aurora PostgreSQL Limitless Database

您可以使用 COPY 命令或使用数据加载实用程序将数据加载到 Aurora PostgreSQL Limitless Database 表中。

> **Note**
>
> 您可以将数据加载到标准表、分片表和引用表中。

目录

- 在 Aurora PostgreSQL Limitless Database 中使用 COPY 命令
  - 使用 COPY 命令将数据加载到 Aurora PostgreSQL Limitless Database
    - 将数据拆分为多个文件
  - 使用 COPY 命令将 Limitless Database 数据复制到文件
- 使用 Aurora PostgreSQL Limitless Database 数据加载实用程序
  - 限制
  - 先决条件
  - 准备源数据库
  - 准备目标数据库
  - 创建数据库凭证
    - 创建源数据库凭证
    - 创建目标数据库凭证
  - 使用脚本设置数据库身份验证和资源访问权限
    - 数据加载实用程序的设置脚本
    - 数据加载实用程序设置脚本的输出
    - 清理失败的资源
  - 手动设置数据库身份验证和资源访问权限
    - 创建客户托管的 AWS KMS key
    - 创建数据库密钥
    - 创建 IAM 角色
    - 更新客户托管的 AWS KMS key
    - 添加 IAM 角色权限策略

- 从 Aurora PostgreSQL 数据库集群或 RDS for PostgreSQL 数据库实例加载数据。
- 监控数据加载
  - 列出数据加载任务
  - 使用任务 ID 查看数据加载任务的详细信息
  - 监控 Amazon CloudWatch 日志组
  - 监控 RDS 事件
- 取消数据加载

在 Aurora PostgreSQL Limitless Database 中使用 COPY 命令

您可以使用 psql 实用程序中的 \copy 功能将数据导入 Aurora PostgreSQL Limitless Database 并从中导出数据

使用 COPY 命令将数据加载到 Aurora PostgreSQL Limitless Database

Aurora PostgreSQL Limitless Database 与用于导入数据的 psql 实用程序中的 \copy 功能兼容。

与 Aurora PostgreSQL 一样，在 Limitless Database 中，不支持以下操作：

- 从 SSH 直接访问数据库实例 – 您无法将数据文件（例如 .csv 格式的文件）复制到数据库实例主机并从该文件运行 COPY。
- 在数据库实例上使用本地文件 – 使用 COPY ... FROM STDIN 和 COPY ... TO STDOUT。

PostgreSQL 中的 COPY 命令具有处理本地文件 (FROM/TO) 和使用客户端与服务器之间的连接传输数据的选项 (STDIN/STDOUT)。有关更多信息，请参阅 PostgreSQL 文档中的 COPY。

PostgreSQL psql 实用程序中的 \copy 命令适用于运行 psql 客户端的计算机上的本地文件。它会在您连接到的远程（例如，Limitless Database）服务器上调用相应的 COPY ... FROM STDIN 或 COPY ... FROM STDOUT 命令。它从本地文件向 STDIN 读取数据，或从 STDOUT 向本地文件写入数据。

将数据拆分为多个文件

数据存储在 Aurora PostgreSQL Limitless Database 中的多个分片上。要使用 \copy 加快数据加载速度，您可以将数据拆分为多个文件。然后，通过并行运行单独的 \copy 命令，独立导入每个数据文件。

例如，您有一个 CSV 格式的输入数据文件，其中有 300 万行需要导入。您可以将文件拆分成多个块，每个块容纳 200,000 行（15 个块）：

```
split -l200000 data.csv data_ --additional-suffix=.csv -d
```

这样就会生成 data_00.csv 至 data_14.csv 文件。然后，您可以使用 15 个并行 \copy 命令导入数据，例如：

```
psql -h dbcluster.limitless-111122223333.aws-region.rds.amazonaws.com -U username -c
 "\copy test_table from '/tmp/data_00.csv';" postgres_limitless &
```

```
psql -h dbcluster.limitless-111122223333.aws-region.rds.amazonaws.com -U username -c
 "\copy test_table FROM '/tmp/data_01.csv';" postgres_limitless &
...
psql -h dbcluster.limitless-111122223333.aws-region.rds.amazonaws.com -U username -c
 "\copy test_table FROM '/tmp/data_13.csv';" postgres_limitless &
psql -h dbcluster.limitless-111122223333.aws-region.rds.amazonaws.com -U username -c
 "\copy test_table FROM '/tmp/data_14.csv';" postgres_limitless
```

使用这种技术，导入相同数量数据的速度大约是使用单个 \copy 命令的 10 倍。

使用 COPY 命令将 Limitless Database 数据复制到文件

您可以使用 \copy 命令将数据从无限表复制到文件中，如以下示例所示：

```
postgres_limitless=> \copy test_table TO '/tmp/test_table.csv' DELIMITER ',' CSV
 HEADER;
```

使用 Aurora PostgreSQL Limitless Database 数据加载实用程序

Aurora 提供了一种实用程序，用于从 Aurora PostgreSQL 数据库集群或 RDS for PostgreSQL 数据库实例将数据直接加载到 Limitless Database 中。

您可以执行以下步骤来使用数据加载实用程序：

1. 先决条件
2. 准备源数据库
3. 准备目标数据库
4. 创建数据库凭证
5. 下列情况之一：
   - 使用脚本设置数据库身份验证和资源访问权限（推荐）
   - 手动设置数据库身份验证和资源访问权限
6. 从 Aurora PostgreSQL 数据库集群或 RDS for PostgreSQL 数据库实例加载数据。

限制

数据加载实用程序具有以下限制：

- 不支持以下数据类型：enum、ARRAY、BOX、CIRCLE、LINE、LSEG、PATH、PG_LSN、PG_SNAPSHOT、POLYGON、TSQUE 和 TXID_SNAPSHOT。
- 加载期间，VARBIT 数据类型的前导零 (0) 将被去除。
- 当源表上有复合主键时，数据迁移失败。
- 当目标表上有外键时，数据迁移失败。
- 不支持从 RDS for PostgreSQL 多可用区数据库集群加载数据。

先决条件

数据加载实用程序具有以下先决条件：

- 源数据库使用 Aurora PostgreSQL 或 RDS for PostgreSQL 版本 11.x 及更高版本。
- 源数据库与目标数据库分片组位于相同的 AWS 账户和 AWS 区域中。
- 源数据库集群或数据库实例处于 available 状态。

- 源数据库和 Limitless Database 上的表具有相同的表名称、列名称和列数据类型。
- 源表和目标表的主键使用相同的列和列顺序。
- 要运行数据加载命令，您必须有一个可以连接到无限数据库的环境。可用命令如下：
  - `rds_aurora.limitless_data_load_start`
  - `rds_aurora.limitless_data_load_cancel`
- 对于 CDC：
  - 源数据库和目标数据库分片组必须使用相同的数据库子网组、VPC 安全组和数据库端口。这些设置用于与源数据库和数据库分片组中路由器的网络连接。
  - 您必须在源数据库上启用逻辑复制。源数据库用户必须具有读取逻辑复制的权限。

准备源数据库

要访问源数据库进行数据加载，您必须允许传入的网络流量进入该数据库。执行以下步骤。

允许网络流量进入源数据库

1. 登录到 AWS Management Console 并打开 Amazon EC2 控制台（https://console.aws.amazon.com/ec2/）。
2. 导航到安全组页面。
3. 为源数据库集群或实例使用的安全组选择安全组 ID。

   例如，其安全组 ID 为 `sg-056a84f1712b77926`。
4. 在入站规则选项卡：

   a. 选择编辑入站规则。

   b. 为源数据库集群或实例添加新的入站规则：

      - 端口范围 – 源数据库的数据库端口，通常是 5432
      - 安全组 ID – 本例中为 `sg-056a84f1712b77926`



5. 在出站规则选项卡：

   a. 选择编辑出站规则。

   b. 为源数据库集群或实例添加新的出站规则：

      - 数据库端口 – `All traffic`（包括端口 `0-65535`）
      - 安全组 ID – 本例中为 `sg-056a84f1712b77926`



6. 登录到 AWS Management Console并打开 Amazon VPC 控制台，网址：https://console.aws.amazon.com/vpc/。
7. 导航到网络 ACL 页面。
8. 添加默认网络 ACL 中概述的默认网络 ACL 配置。

准备目标数据库

按照 创建 Aurora PostgreSQL Limitless Database 表 中的过程在数据库分片组中创建目标表。

目标表的架构、表名称和主键必须与源表相同。

创建数据库凭证

您必须在源数据库和目标数据库中创建数据库用户，然后向这些用户授予必要的权限。有关更多信息，请参阅 PostgreSQL 文档中的 CREATE USER 和 GRANT。

创建源数据库凭证

在命令中传递源数据库用户以开始加载。该用户必须拥有从源数据库执行复制的权限。

1. 使用数据库主用户（或具有 rds_superuser 角色的其他用户）创建具有 LOGIN 权限的源数据库用户。

```
CREATE USER source_db_username WITH PASSWORD 'source_db_user_password';
```

2. 将 rds_superuser 角色授予您的源数据库用户。

```
GRANT rds_superuser to source_db_username;
```

3. 如果您使用的是 full_load_and_cdc 模式，请将 rds_replication 角色授予您的源数据库用户。rds_replication 角色可授予管理逻辑时隙和使用逻辑时隙流式处理数据的权限。

```
GRANT rds_replication to source_db_username;
```

创建目标数据库凭证

目标数据库用户必须拥有写入数据库分片组中目标表的权限。

1. 使用数据库主用户（或具有 rds_superuser 角色的其他用户）创建具有 LOGIN 权限的目标数据库用户。

```
CREATE USER destination_db_username WITH PASSWORD 'destination_db_user_password';
```

2. 将 rds_superuser 角色授予您的目标数据库用户。

```
GRANT rds_superuser to destination_db_username;
```

使用脚本设置数据库身份验证和资源访问权限

设置脚本创建了一个客户托管的 AWS KMS key、一个 AWS Identity and Access Management（IAM）角色和两个 AWS Secrets Manager 密钥。

要使用设置脚本，请执行以下步骤：

1. 确保您已经安装 AWS CLI 并使用 AWS 账户 凭证进行了配置。
2. 安装 `jq` 命令行 JSON 处理器。有关更多信息，请参阅 jqlang/jq。
3. 将 data_loading_script.zip 文件复制到您的计算机上，然后从中解压缩 `data_load_aws_setup_script.sh` 文件。
4. 编辑脚本，将占位符变量替换为以下各项的相应值：
   - 您的 AWS 账户
   - 这些区域有：AWS 区域
   - 源数据库凭证
   - 目标数据库凭证
5. 打开计算机上的新终端并运行以下命令：

```
bash ./data_load_aws_setup_script.sh
```

数据加载实用程序的设置脚本

我们在此提供 `data_load_aws_setup_script.sh` 文件的文本以供参考。

```
#!/bin/bash
# Aurora Limitless data loading - AWS resources setup script #
# Set up the account credentials in advance. #
# Update the following script variables. #

###################################
#### Start of variable section ####

ACCOUNT_ID="12-digit_AWS_account_ID"
REGION="AWS_Region"
DATE=$(date +'%m%d%H%M%S')
RANDOM_SUFFIX="${DATE}"
SOURCE_SECRET_NAME="secret-source-${DATE}"
SOURCE_USERNAME="source_db_username"
```

```
SOURCE_PASSWORD="source_db_password"
DESTINATION_SECRET_NAME="secret-destination-${DATE}"
DESTINATION_USERNAME="destination_db_username"
DESTINATION_PASSWORD="destination_db_password"
DATA_LOAD_IAM_ROLE_NAME="aurora-data-loader-${RANDOM_SUFFIX}"
TMP_WORK_DIR="./tmp_data_load_aws_resource_setup/"

#### End of variable section ####
################################

# Main logic start
echo "DATE - [${DATE}]"
echo "RANDOM_SUFFIX - [${RANDOM_SUFFIX}]"
echo 'START!'

mkdir -p $TMP_WORK_DIR

# Create the symmetric KMS key for encryption and decryption.
TMP_FILE_PATH="${TMP_WORK_DIR}tmp_create_key_response.txt"
aws kms create-key --region $REGION | tee $TMP_FILE_PATH
KMS_KEY_ARN=$(cat $TMP_FILE_PATH | jq -r '.KeyMetadata.Arn')
aws kms create-alias \
    --alias-name alias/"${DATA_LOAD_IAM_ROLE_NAME}-key" \
    --target-key-id $KMS_KEY_ARN \
    --region $REGION

# Create the source secret.
TMP_FILE_PATH="${TMP_WORK_DIR}tmp_create_source_secret_response.txt"
aws secretsmanager create-secret \
    --name $SOURCE_SECRET_NAME \
    --kms-key-id $KMS_KEY_ARN \
    --secret-string "{\"username\":\"$SOURCE_USERNAME\",\"password\":\"$SOURCE_PASSWORD
\"}" \
    --region $REGION \
    | tee $TMP_FILE_PATH
SOURCE_SECRET_ARN=$(cat $TMP_FILE_PATH | jq -r '.ARN')

# Create the destination secret.
TMP_FILE_PATH="${TMP_WORK_DIR}tmp_create_destination_secret_response.txt"
aws secretsmanager create-secret \
    --name $DESTINATION_SECRET_NAME \
    --kms-key-id $KMS_KEY_ARN \
    --secret-string "{\"username\":\"$DESTINATION_USERNAME\",\"password\":
\"$DESTINATION_PASSWORD\"}" \
```

```
    --region $REGION \
    | tee $TMP_FILE_PATH
DESTINATION_SECRET_ARN=$(cat $TMP_FILE_PATH | jq -r '.ARN')

# Create the RDS trust policy JSON file.
# Use only rds.amazonaws.com for RDS PROD use cases.
TRUST_POLICY_PATH="${TMP_WORK_DIR}rds_trust_policy.json"
echo '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "rds.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}' > $TRUST_POLICY_PATH

# Create the IAM role.
TMP_FILE_PATH="${TMP_WORK_DIR}tmp_create_iam_role_response.txt"
aws iam create-role \
    --role-name $DATA_LOAD_IAM_ROLE_NAME \
    --assume-role-policy-document "file://${TRUST_POLICY_PATH}" \
    --tags Key=assumer,Value=aurora_limitless_table_data_load \
    --region $REGION \
    | tee $TMP_FILE_PATH
IAM_ROLE_ARN=$(cat $TMP_FILE_PATH | jq -r '.Role.Arn')

# Create the permission policy JSON file.
PERMISSION_POLICY_PATH="${TMP_WORK_DIR}data_load_permission_policy.json"
permission_json_policy=$(cat &lt;&lt;EOF
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Ec2Permission",
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeNetworkInterfaces",
                "ec2:CreateNetworkInterface",
```

```
                "ec2:DeleteNetworkInterface",
                "ec2:CreateNetworkInterfacePermission",
                "ec2:DeleteNetworkInterfacePermission",
                "ec2:DescribeNetworkInterfacePermissions",
                "ec2:ModifyNetworkInterfaceAttribute",
                "ec2:DescribeNetworkInterfaceAttribute",
                "ec2:DescribeAvailabilityZones",
                "ec2:DescribeRegions",
                "ec2:DescribeVpcs",
                "ec2:DescribeSubnets",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeNetworkAcls"
            ],
            "Resource": "*"
        },
        {
            "Sid": "SecretsManagerPermissions",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue",
                "secretsmanager:DescribeSecret"
            ],
            "Resource": [
                "$SOURCE_SECRET_ARN",
                "$DESTINATION_SECRET_ARN"
            ]
        },
        {
            "Sid": "KmsPermissions",
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:DescribeKey",
                "kms:GenerateDataKey"
            ],
            "Resource": "$KMS_KEY_ARN"
        },
        {
            "Sid": "RdsPermissions",
            "Effect": "Allow",
            "Action": [
                "rds:DescribeDBClusters",
                "rds:DescribeDBInstances"
            ],
```

```
            "Resource": "*"
        }
    ]
}
EOF
)
echo $permission_json_policy > $PERMISSION_POLICY_PATH

# Add the inline policy.
aws iam put-role-policy \
    --role-name $DATA_LOAD_IAM_ROLE_NAME \
    --policy-name aurora-limitless-data-load-policy \
    --policy-document "file://${PERMISSION_POLICY_PATH}" \
    --region $REGION

# Create the key policy JSON file.
KEY_POLICY_PATH="${TMP_WORK_DIR}data_load_key_policy.json"
key_json_policy=$(cat &lt;&lt;EOF
{
    "Id": "key-aurora-limitless-data-load-$RANDOM_SUFFIX",
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Enable IAM User Permissions",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::$ACCOUNT_ID:root"
            },
            "Action": "kms:*",
            "Resource": "*"
        },
        {
            "Sid": "Allow use of the key",
            "Effect": "Allow",
            "Principal": {
                "AWS": "$IAM_ROLE_ARN"
            },
            "Action": [
                "kms:Decrypt",
                "kms:DescribeKey",
                "kms:GenerateDataKey"
            ],
            "Resource": "*"
        }
```

```
    ]
}
EOF
)
echo $key_json_policy > $KEY_POLICY_PATH

# Add the key policy.
TMP_FILE_PATH="${TMP_WORK_DIR}tmp_put_key_policy_response.txt"
sleep 10 # sleep 10 sec for IAM role ready
aws kms put-key-policy \
    --key-id $KMS_KEY_ARN \
    --policy-name default \
    --policy "file://${KEY_POLICY_PATH}" \
    --region $REGION \
    | tee $TMP_FILE_PATH

echo 'DONE!'

echo "ACCOUNT_ID : [${ACCOUNT_ID}]"
echo "REGION : [${REGION}]"
echo "RANDOM_SUFFIX : [${RANDOM_SUFFIX}]"
echo "IAM_ROLE_ARN : [${IAM_ROLE_ARN}]"
echo "SOURCE_SECRET_ARN : [${SOURCE_SECRET_ARN}]"
echo "DESTINATION_SECRET_ARN : [${DESTINATION_SECRET_ARN}]"

# Example of a successful run:
# ACCOUNT_ID : [012345678912]
# REGION : [ap-northeast-1]
# RANDOM_SUFFIX : [0305000703]
# IAM_ROLE_ARN : [arn:aws:iam::012345678912:role/aurora-data-loader-0305000703]
# SOURCE_SECRET_ARN : [arn:aws:secretsmanager:ap-
northeast-1:012345678912:secret:secret-source-0305000703-yQDtow]
# DESTINATION_SECRET_ARN : [arn:aws:secretsmanager:ap-
northeast-1:012345678912:secret:secret-destination-0305000703-5d5Jy8]

# If you want to manually clean up failed resource,
# please remove them in the following order:
# 1. IAM role.
 # aws iam delete-role-policy --role-name Test-Role --policy-name ExamplePolicy --
region us-east-1
# aws iam delete-role --role-name Test-Role --region us-east-1
# 2. Source and destination secrets.
# aws secretsmanager delete-secret --secret-id MyTestSecret --force-delete-without-
recovery --region us-east-1
```

```
# 3. KDM key.
# aws kms schedule-key-deletion --key-id arn:aws:kms:us-
east-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab --pending-window-in-days 7
 --region us-east-1
```

数据加载实用程序设置脚本的输出

以下示例显示了脚本成功运行后的输出。

```
% bash ./data_load_aws_setup_script.sh
DATE - [0305000703]
RANDOM_SUFFIX - [0305000703]
START!
{
    "KeyMetadata": {
        "AWSAccountId": "123456789012",
        "KeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
        "Arn": "arn:aws:kms:ap-
northeast-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab",
        "CreationDate": "2024-03-05T00:07:49.852000+00:00",
        "Enabled": true,
        "Description": "",
        "KeyUsage": "ENCRYPT_DECRYPT",
        "KeyState": "Enabled",
        "Origin": "AWS_KMS",
        "KeyManager": "CUSTOMER",
        "CustomerMasterKeySpec": "SYMMETRIC_DEFAULT",
        "KeySpec": "SYMMETRIC_DEFAULT",
        "EncryptionAlgorithms": [
            "SYMMETRIC_DEFAULT"
        ],
        "MultiRegion": false
    }
}
{
    "ARN": "arn:aws:secretsmanager:ap-northeast-1:123456789012:secret:secret-
source-0305000703-yQDtow",
    "Name": "secret-source-0305000703",
    "VersionId": "a017bebe-a71b-4220-b923-6850c2599c26"
}
{
    "ARN": "arn:aws:secretsmanager:ap-northeast-1:123456789012:secret:secret-
destination-0305000703-5d5Jy8",
    "Name": "secret-destination-0305000703",
```

```
    "VersionId": "32a1f989-6391-46b1-9182-f65d242f5eb6"
}
{
    "Role": {
        "Path": "/",
        "RoleName": "aurora-data-loader-0305000703",
        "RoleId": "AROAYPX63ITQOYORQSC6U",
        "Arn": "arn:aws:iam::123456789012:role/aurora-data-loader-0305000703",
        "CreateDate": "2024-03-05T00:07:54+00:00",
        "AssumeRolePolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Effect": "Allow",
                    "Principal": {
                        "Service": [
                            "rds.amazonaws.com"
                        ]
                    },
                    "Action": "sts:AssumeRole"
                }
            ]
        },
        "Tags": [
            {
                "Key": "assumer",
                "Value": "aurora_limitless_table_data_load"
            }
        ]
    }
}
DONE!
ACCOUNT_ID : [123456789012]
REGION : [ap-northeast-1]
RANDOM_SUFFIX : [0305000703]
IAM_ROLE_ARN : [arn:aws:iam::123456789012:role/aurora-data-loader-0305000703]
SOURCE_SECRET_ARN : [arn:aws:secretsmanager:ap-northeast-1:123456789012:secret:secret-
source-0305000703-yQDtow]
DESTINATION_SECRET_ARN : [arn:aws:secretsmanager:ap-
northeast-1:123456789012:secret:secret-destination-0305000703-5d5Jy8]
```

清理失败的资源

如果要手动清理失败的资源，请按以下顺序将其移除：

1. IAM 角色，例如：

```
aws iam delete-role-policy \
--role-name Test-Role \
--policy-name ExamplePolicy

aws iam delete-role \
--role-name Test-Role
```

2. 源密钥和目标密钥，例如：

```
aws secretsmanager delete-secret \
--secret-id MyTestSecret \
--force-delete-without-recovery
```

3. KMS 密钥，例如：

```
aws kms schedule-key-deletion \
--key-id arn:aws:kms:us-west-2:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab
 \
--pending-window-in-days 7
```

然后即可重试脚本。

手动设置数据库身份验证和资源访问权限

手动设置数据库身份验证和资源访问权限的流程包括以下步骤：

1. 创建客户托管的 AWS KMS key
2. 添加 IAM 角色权限策略
3. 创建数据库密钥
4. 创建 IAM 角色
5. 更新客户托管的 AWS KMS key

此步骤为可选项，执行的任务与 使用脚本设置数据库身份验证和资源访问权限 中相同。建议使用脚本。

创建客户托管的 AWS KMS key

按照创建对称加密密钥中的过程创建客户托管的 KMS 密钥。如果满足这些要求，您也可以使用现有密钥。

创建客户托管的 KMS 密钥

1. 登录到 AWS Management Console，然后通过以下网址打开 AWS KMS 控制台：https://console.aws.amazon.com/kms。
2. 导航到客户托管的密钥页面。
3. 选择创建密钥。
4. 在配置密钥页面上：
   a. 对于密钥类型，选择对称。
   b. 对于密钥用途，选择加密和解密。
   c. 选择下一步。
5. 在添加标签页面上，输入别名，例如 **`limitless`**，然后选择下一步。
6. 在定义密钥管理员权限页面上，确保选中允许密钥管理员删除此密钥复选框，然后选择下一步。
7. 在定义密钥使用权限页面上，选择下一步。
8. 在检查页面上，选择完成。

   您可以稍后更新密钥策略。

记录要在 添加 IAM 角色权限策略 中使用的 KMS 密钥的 Amazon 资源名称（ARN）。

有关使用 AWS CLI 创建客户托管的 KMS 密钥的信息，请参阅 create-key 和 create-alias。

创建数据库密钥

要允许数据加载实用程序访问源数据库表和目标数据库表，请在 AWS Secrets Manager 中创建两个密钥：一个用于源数据库，另一个用于目标数据库。这些密钥存储访问源数据库和目标数据库的用户名和密码。

按照创建 AWS Secrets Manager 密钥中的过程创建键值对密钥。

创建数据库密钥

1. 通过 https://console.aws.amazon.com/secretsmanager/ 打开 Secrets Manager 控制台。
2. 选择存储新密钥。
3. 在选择密钥类型页面上：
   a. 对于密钥类型，选择其他密钥类型。
   b. 对于键/值对，选择纯文本标签。
   c. 输入以下 JSON 代码，其中 *sourcedbreader* 和 *sourcedbpassword* 是 创建源数据库凭证 中源数据库用户的凭证。

```
{
    "username":"sourcedbreader",
    "password":"sourcedbpassword"
}
```

   d. 对于加密密钥，选择您在 创建客户托管的 AWS KMS key 中创建的 KMS 密钥，例如 `limitless`。
   e. 选择下一步。
4. 在配置密钥页面上，输入密钥名称，例如 **`source_DB_secret`**，然后选择下一步。
5. 在配置旋转 - 可选页面上，选择下一步。
6. 在检查页面上，选择存储。
7. 对目标数据库密钥重复此过程：
   a. 输入以下 JSON 代码，其中 *destinationdbwriter* 和 *destinationdbpassword* 是 创建目标数据库凭证 中目标数据库用户的凭证。

```
{
    "username":"destinationdbwriter",
    "password":"destinationdbpassword"
}
```

b. 输入密钥名称，例如 **destination_DB_secret**。

记录要在 添加 IAM 角色权限策略 中使用的密钥的 ARN。

创建 IAM 角色

数据加载需要您提供对 AWS 资源的访问权限。要提供访问权限，您可以按照创建角色，为 IAM 用户授权中的流程创建 `aurora-data-loader` IAM 角色。

创建 IAM 角色

1. 登录 AWS Management Console，然后通过以下网址打开 IAM 控制台：https://console.aws.amazon.com/iam/。
2. 导航到角色页面。
3. 选择创建角色。
4. 在选择可信实体页面：

   a. 对于可信实体类型，选择自定义信任策略。

   b. 为自定义信任策略输入以下 JSON 代码：

   JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "rds.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
```

```
    ]
}
```

c. 选择下一步。

5. 在添加权限页面上，选择下一步。
6. 在名称、检查和创建页面：

a. 对于角色名称，输入 **aurora-data-loader** 或您喜欢的其他名称。

b. 选择添加标签，然后输入以下标签：

- 键：**assumer**
- 值：**aurora_limitless_table_data_load**

> **Important**
>
> Aurora PostgreSQL Limitless Database 只能担任具有此标签的 IAM 角色。

c. 选择创建角色。

更新客户托管的 AWS KMS key

按照更改密钥策略中的过程将 IAM 角色 `aurora-data-loader` 添加到默认密钥策略。

将 IAM 角色添加到密钥策略

1. 登录到 AWS Management Console，然后通过以下网址打开 AWS KMS 控制台：https://console.aws.amazon.com/kms。
2. 导航到客户托管的密钥页面。
3. 选择您在 创建客户托管的 AWS KMS key 中创建的 KMS 密钥，例如 `limitless`。
4. 在密钥策略选项卡，对于密钥用户，选择添加。
5. 在添加密钥用户窗口，选择您在 创建 IAM 角色 中创建的 IAM 角色名称，例如 aurora-data-loader。
6. 选择添加。

添加 IAM 角色权限策略

您必须将权限策略添加到您创建的 IAM 角色。这将允许 Aurora PostgreSQL Limitless Database 数据加载实用程序访问相关 AWS 资源，以建立网络连接并检索源和目标数据库凭证密钥。

有关更多信息，请参阅修改角色。

添加权限策略

1. 登录 AWS Management Console，然后通过以下网址打开 IAM 控制台：https://console.aws.amazon.com/iam/。
2. 导航到角色页面。
3. 选择您在 创建 IAM 角色 中创建的 IAM 角色，例如 aurora-data-loader。
4. 在权限选项卡，对于权限策略，选择添加权限，然后创建内联策略。
5. 在指定权限页面上，选择 JSON 编辑器。
6. 将以下模板复制并粘贴到 JSON 编辑器中，用数据库密钥和 KMS 密钥的 ARN 替换占位符。

   JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Ec2Permission",
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeNetworkInterfaces",
                "ec2:CreateNetworkInterface",
                "ec2:DeleteNetworkInterface",
                "ec2:CreateNetworkInterfacePermission",
                "ec2:DeleteNetworkInterfacePermission",
                "ec2:DescribeNetworkInterfacePermissions",
                "ec2:ModifyNetworkInterfaceAttribute",
                "ec2:DescribeNetworkInterfaceAttribute",
                "ec2:DescribeAvailabilityZones",
                "ec2:DescribeRegions",
                "ec2:DescribeVpcs",
                "ec2:DescribeSubnets",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeNetworkAcls"
```

```
            ],
            "Resource": "*"
        },
        {
            "Sid": "SecretsManagerPermissions",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue",
                "secretsmanager:DescribeSecret"
            ],
            "Resource": [
                "arn:aws:secretsmanager:us-
east-1:123456789012:secret:source_DB_secret-ABC123",
                "arn:aws:secretsmanager:us-
east-1:123456789012:secret:destination_DB_secret-456DEF"
            ]
        },        {
            "Sid": "KmsPermissions",
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:DescribeKey",
                "kms:GenerateDataKey"
            ],
            "Resource": "arn:aws:kms:us-east-1:123456789012:key/aa11bb22-
####-####-####-fedcba123456"
        },
        {
            "Sid": "RdsPermissions",
            "Effect": "Allow",
            "Action": [
                "rds:DescribeDBClusters",
                "rds:DescribeDBInstances"
            ],
            "Resource": "*"
        }
    ]
}
```

7. 检查是否有错并纠正错误。
8. 选择下一步。
9. 在检查并创建页面上，输入策略名称，例如 **data_loading_policy**，然后选择创建策略。

从 Aurora PostgreSQL 数据库集群或 RDS for PostgreSQL 数据库实例加载数据。

完成资源和身份验证设置后，连接到集群端点，然后从 Limitless Database（如 `postgres_limitless`）调用 `rds_aurora.limitless_data_load_start` 存储过程。Limitless Database 是您要将数据迁移到其中的数据库分片组上的数据库。

此函数在后台异步连接到命令中指定的源数据库，从源数据库读取数据，然后将数据加载到分片上。为了提高性能，将使用并行线程加载数据。该函数通过运行 `SELECT` 命令读取命令中提供的表的数据来检索时间点表快照。

您可以将数据加载到分片表、引用表和标准表中。

您可以在 `rds_aurora.limitless_data_load_start` 调用中加载数据库、架构或表级别的数据。

- 数据库 – 您可以在每次调用中一次加载一个数据库，对数据库中的架构或表数没有限制。
- 架构 – 每次调用最多可以加载 15 个架构，每个架构中的表数没有限制。
- 表 – 每次调用最多可以加载 15 个表。

> **Note**
>
> 此功能不使用 Amazon RDS 快照或数据库的时间点隔离。为了保持各表之间的一致性，建议克隆源数据库并指向该克隆的数据库作为源。

存储过程使用以下语法：

```
CALL rds_aurora.limitless_data_load_start('source_type',
    'source_DB_cluster_or_instance_ID',
    'source_database_name',
    'streaming_mode',
    'data_loading_IAM_role_arn',
    'source_DB_secret_arn',
    'destination_DB_secret_arn',
    'ignore_primary_key_conflict_boolean_flag',
    'is_dry_run',
    (optional parameter) schemas/tables => ARRAY['name1', 'name2', ...]);
```

以下是输入参数：

- source_type – 源类型：aurora_postgresql 或 rds_postgresql
- source_DB_cluster_or_instance_ID – 源 Aurora PostgreSQL 数据库集群标识符或 RDS for PostgreSQL 数据库实例标识符
- source_database_name – 源数据库名称，例如 *postgres*
- streaming_mode – 是否包括更改数据捕获（CDC）：full_load 或 full_load_and_cdc
- data_loading_IAM_role_arn – aurora-data-loader 的 IAM 角色 Amazon 资源名称（ARN）
- source_DB_secret_arn – 源数据库密钥 ARN
- destination_DB_secret_arn – 目标数据库密钥 ARN
- ignore_primary_key_conflict_boolean_flag – 如果发生主键冲突，是否继续：
  - 如果设置为 true，数据加载将忽略存在主键冲突的行的新更改。
  - 如果设置为 false，数据加载在遇到主键冲突时会覆盖目标表上的现有行。
- is_dry_run – 是否测试数据加载任务是否可以连接到源数据库和目标数据库：
  - 如果设置为 true，则在不加载数据的情况下测试连接
  - 如果设置为 false，则在加载数据的情况下测试连接
- （可选）schemas 或 tables – 要加载的架构或表的数组。您可以指定以下任一值：
  - 格式为 tables => ARRAY['*schema1*.*table1*', '*schema1*.*table2*', '*schema2*.*table1*', ...] 的表列表
  - 格式为 schemas => ARRAY['*schema1*', '*schema2*', ...] 的架构列表

  如果未包含该参数，则迁移整个指定的源数据库。

输出参数是带有消息的任务 ID。

以下示例说明如何使用 `rds_aurora.limitless_data_load_start` 存储过程从 Aurora PostgreSQL 数据库集群加载数据。

```
CALL rds_aurora.limitless_data_load_start('aurora_postgresql',
    'my-db-cluster',
    'postgres',
    'full_load_and_cdc',
    'arn:aws:iam::123456789012:role/aurora-data-loader-8f2c66',
    'arn:aws:secretsmanager:us-east-1:123456789012:secret:secret-source-8f2c66-EWrr0V',
    'arn:aws:secretsmanager:us-east-1:123456789012:secret:secret-destination-8f2c66-
d04fbD',
    'true',
    'false',
    tables => ARRAY['public.customer', 'public.order', 'public.orderdetails']);

INFO: limitless data load job id 1688761223647 is starting.
```

监控数据加载

Aurora PostgreSQL Limitless Database 提供了多种监控数据加载任务的方式：

- 列出数据加载任务
- 使用任务 ID 查看数据加载任务的详细信息
- 监控 Amazon CloudWatch 日志组
- 监控 RDS 事件

列出数据加载任务

您可以连接到集群端点，然后使用 `rds_aurora.limitless_data_load_jobs` 视图列出数据加载任务。

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_data_load_jobs LIMIT 6;

    job_id     |  status   | message |     source_db_identifier      | source_db_name |
 full_load_complete_time |
   progress_details                                                                 |
     start_time       |   last_updated_time    |  streaming_mode   | source_engine_type
 | ignore_primary_key_conflict | is_dryrun
--------------+-----------+---------+-------------------------------+---------------
+-------------------------
+-------------------------------------------------------------------------------------------------------
+------------------------+-------------------------+------------------
+--------------------+-----------------------------+-----------
 1725697520693 | COMPLETED |         | persistent-kdm-auto-source-01 | postgres
    | 2024-09-07 08:48:15+00  | {"FULL_LOAD": {"STATUS": "COMPLETED", "DETAILS": "9
 of 9 tables loaded", "COMPLETED_AT": "2024/09/07 08:48:15+00", "RECORDS_MIGRATED":
 600003}} | 2024-09-07 08:47:13+00 | 2024-09-07 08:48:15+00 | full_load         |
 aurora_postgresql  | t                          | f
 1725696114225 | COMPLETED |         | persistent-kdm-auto-source-01 | postgres
    | 2024-09-07 08:24:20+00  | {"FULL_LOAD": {"STATUS": "COMPLETED", "DETAILS": "3
 of 3 tables loaded", "COMPLETED_AT": "2024/09/07 08:24:20+00", "RECORDS_MIGRATED":
 200001}} | 2024-09-07 08:23:56+00 | 2024-09-07 08:24:20+00 | full_load         |
 aurora_postgresql  | t                          | f
 1725696067630 | COMPLETED |         | persistent-kdm-auto-source-01 | postgres
    | 2024-09-07 08:23:45+00  | {"FULL_LOAD": {"STATUS": "COMPLETED", "DETAILS": "6
 of 6 tables loaded", "COMPLETED_AT": "2024/09/07 08:23:45+00", "RECORDS_MIGRATED":
 400002}} | 2024-09-07 08:23:10+00 | 2024-09-07 08:23:45+00 | full_load         |
 aurora_postgresql  | t                          | f
```

```
 1725694221753 | CANCELED  |         | persistent-kdm-auto-source-01 | postgres
    |                         | {}

            | 2024-09-07 07:31:18+00 | 2024-09-07 07:51:49+00 | full_load_and_cdc |
 aurora_postgresql  | t                           | f
 1725691698210 | COMPLETED |         | persistent-kdm-auto-source-01 | postgres
    | 2024-09-07 07:10:51+00  | {"FULL_LOAD": {"STATUS": "COMPLETED", "DETAILS": "1
 of 1 tables loaded", "COMPLETED_AT": "2024/09/07 07:10:51+00", "RECORDS_MIGRATED":
 100000}} | 2024-09-07 07:10:42+00 | 2024-09-07 07:10:52+00 | full_load         |
 aurora_postgresql  | t                           | f
 1725691695049 | COMPLETED |         | persistent-kdm-auto-source-01 | postgres
    | 2024-09-07 07:10:48+00  | {"FULL_LOAD": {"STATUS": "COMPLETED", "DETAILS": "1
 of 1 tables loaded", "COMPLETED_AT": "2024/09/07 07:10:48+00", "RECORDS_MIGRATED":
 100000}} | 2024-09-07 07:10:41+00 | 2024-09-07 07:10:48+00 | full_load         |
 aurora_postgresql  | t                           | f
(6 rows)
```

任务记录将在 90 天后删除。

使用任务 ID 查看数据加载任务的详细信息

如果您知道任务 ID，则可以连接到集群端点，然后使用 `rds_aurora.limitless_data_load_job_details` 视图查看该数据加载任务的详细信息，包括表名称、任务状态和已加载的行数。您可以在对数据加载启动函数的响应中获取任务 ID，也可以从 `rds_aurora.limitless_data_load_jobs` 视图中获取。

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_data_load_job_details WHERE
 job_id='1725696114225';

job_id        | destination_table_name | destination_schema_name | start_time
      | status    | full_load_rows | full_load_total_rows | full_load_complete_time |
 cdc_insert | cdc_update | cdc_delete
--------------+------------------------+-------------------------
+-------------------------+-----------+----------------+---------------------
+-------------------------+------------+------------+------------
1725696114225 | standard_1             | public                  | 2024-09-07
 08:23:57+00 | COMPLETED | 100000         | 100000               | 2024-09-07
 08:24:08+00  | 0          | 0          | 0
1725696114225 | standard_2             | public                  | 2024-09-07
 08:24:08+00 | COMPLETED | 100000         | 100000               | 2024-09-07
 08:24:17+00  | 0          | 0          | 0
```

```
1725696114225 | standard_3             | public                  | 2024-09-07
 08:24:18+00 | COMPLETED | 1             | 1                       | 2024-09-07
 08:24:20+00  | 0          | 0          | 0
1725696114225 | standard_4             | public                  | 2024-09-07
 08:23:58+00 | PENDING   | 0             | 0                       |
    | 0          | 0          | 0
(4 rows)
```

任务记录将在 90 天后删除。

监控 Amazon CloudWatch 日志组

数据加载任务状态更改为 RUNNING 后，您可以使用 Amazon CloudWatch Logs 查看运行进度。

监控 CloudWatch 日志流

登录 AWS Management Console并打开 CloudWatch 控制台（https://console.aws.amazon.com/cloudwatch/）。

1. 导航到日志，然后导航到日志组。
2. 选择 /aws/rds/aurora-limitless-database 日志组。
3. 通过 job_id 搜索您的数据加载任务的日志流。

   日志流的模式为 Data-Load-Job-***job_id***。
4. 选择日志流以查看日志事件。

每个日志流都显示包含任务状态和已加载到 Aurora PostgreSQL Limitless Database 目标表中的行数的事件。如果数据加载任务失败，还会创建错误日志，显示失败状态和原因。

任务记录将在 90 天后删除。

监控 RDS 事件

数据加载任务还会发布 RDS 事件，例如当任务成功、失败或取消时。您可以查看目标数据库中的事件。

有关更多信息，请参阅 ???。

取消数据加载

要取消数据加载任务，请使用任务 ID 作为输入参数调用 `rds_aurora.limitless_data_load_cancel` 存储过程。您可以从启动特定数据加载任务的数据库分片组中的同一数据库中调用此存储过程。例如：

```
CALL rds_aurora.limitless_data_load_cancel(12345);

INFO: limitless data load job with id 12345 is canceling without rollback.
```

您无法取消不存在或不在同一个数据库分片组中运行的数据加载任务。

如响应所示，Aurora PostgreSQL Limitless Database 数据加载实用程序无需回滚即可将加载的数据保留在目标表中。如果您不想保留加载的数据，建议截断目标表。

查询 Aurora PostgreSQL Limitless Database

Aurora PostgreSQL Limitless Database 与 PostgreSQL 查询语法兼容。您可以使用 `psql` 或任何其他与 PostgreSQL 配合使用的连接实用程序来查询您的 Limitless Database。要运行查询，您需要连接到无限端点，如 连接到您的 Aurora PostgreSQL Limitless Database 数据库集群 所示。

Aurora PostgreSQL Limitless Database 支持所有 PostgreSQL `SELECT` 查询。然而，查询是在两个层上执行的：

1. 客户端向其发送查询的路由器
2. 数据所在的分片

性能取决于以一种允许数据库在不同分片上同时处理不同查询的方式来查询数据库。查询首先在分布式事务层（路由器）中进行解析。在计划查询执行之前，有一个分析阶段来确定参与查询的所有关系的位置。如果所有关系都是同一分片上带有已筛选分片键的分片表或引用表，则路由器层上的查询计划将被跳过，完全向下推送到分片上进行规划和执行。这一过程减少了不同节点（路由器和分片）之间的往返次数，可在大多数情况下提高性能。有关更多信息，请参阅 Aurora PostgreSQL Limitless Database 中的单分片查询。

> **Note**
>
> 在某些特定情况下，例如笛卡尔乘积（交叉联接），查询可通过从分片单独检索数据来提高性能。

有关查询执行计划更多信息，请参阅 Aurora PostgreSQL Limitless Database 参考中的 EXPLAIN。有关查询的一般信息，请参阅 PostgreSQL 文档中的 Queries。

主题

- Aurora PostgreSQL Limitless Database 中的单分片查询
- Aurora PostgreSQL Limitless Database 中的分布式查询
- Aurora PostgreSQL Limitless Database 中 PostgreSQL 日志中的分布式查询跟踪
- Aurora PostgreSQL Limitless Database 中的分布式死锁

Aurora PostgreSQL Limitless Database 中的单分片查询

单分片查询是一种可以在保持 SQL ACID 语义的同时直接在分片上运行的查询。当查询计划器在路由器上遇到这样的查询时，计划器会检测到它并继续将整个 SQL 查询向下推送到相应的分片。

此优化可减少从路由器到分片的网络往返行程数，从而提高性能。目前，此优化是针对 INSERT、SELECT、UPDATE 和 DELETE 查询执行的。

主题

- 单分片查询示例
- 单分片查询的限制
- 完全限定（显式）联接
- 设置活动分片密钥

单分片查询示例

在以下示例中，我们有带有分片键 customer_id 的分片表 customers 和引用表 zipcodes。

SELECT

```
postgres_limitless=> EXPLAIN (VERBOSE, COSTS OFF) SELECT * FROM customers WHERE
 customer_id = 100;

                       QUERY PLAN
-------------------------------------------------------------
 Foreign Scan
   Output: customer_id, other_id, customer_name, balance
   Remote SQL:  SELECT customer_id,
     other_id,
     customer_name,
     balance
    FROM public.customers
   WHERE (customer_id = 100)
 Single Shard Optimized
(9 rows)
```

```
postgres_limitless=> EXPLAIN (VERBOSE, COSTS OFF) SELECT * FROM orders
    LEFT JOIN zipcodes ON orders.zipcode_id = zipcodes.zipcode_id
    WHERE customer_id = 11;
```

```
                                                QUERY PLAN

-------------------------------------------------------------------------------------------------
 Foreign Scan
   Output: customer_id, order_id, zipcode_id, customer_name, balance,
 zipcodes.zipcode_id, zipcodes.city
   Remote SQL:  SELECT orders.customer_id,
     orders.order_id,
     orders.zipcode_id,
     orders.customer_name,
     orders.balance,
     zipcodes.zipcode_id,
     zipcodes.city
    FROM (public.orders
      LEFT JOIN public.zipcodes ON ((orders.zipcode_id = zipcodes.zipcode_id)))
   WHERE (orders.customer_id = 11)
 Single Shard Optimized
(13 rows)
```

INSERT

```
postgres_limitless=> EXPLAIN (VERBOSE, COSTS OFF) INSERT INTO customers
    (customer_id, other_id, customer_name, balance)
    VALUES (1, 10, 'saikiran', 1000);

                     QUERY PLAN
-----------------------------------------------------
 Insert on public.customers
   ->  Result
         Output: 1, 10, 'saikiran'::text, '1000'::real
 Single Shard Optimized
(4 rows)
```

UPDATE

```
postgres_limitless=> EXPLAIN (VERBOSE, COSTS OFF) UPDATE orders SET balance = balance +
 100
    WHERE customer_id = 100;

                                       QUERY PLAN

-----------------------------------------------------------------------------------------
 Update on public.orders
```

```
   Foreign Update on public.orders_fs00002 orders_1
   ->  Foreign Update
         Remote SQL:  UPDATE public.orders SET balance = (balance + (100)::double
 precision)
   WHERE (customer_id = 100)
 Single Shard Optimized
(6 rows)
```

DELETE

```
postgres_limitless=> EXPLAIN (VERBOSE, COSTS OFF) DELETE FROM orders
    WHERE customer_id = 100 and balance = 0;

                              QUERY PLAN
-----------------------------------------------------------------------
 Delete on public.orders
   Foreign Delete on public.orders_fs00002 orders_1
   ->  Foreign Delete
         Remote SQL:  DELETE FROM public.orders
   WHERE ((customer_id = 100) AND (balance = (0)::double precision))
 Single Shard Optimized
(6 rows)
```

单分片查询的限制

单分片查询有以下限制：

函数

如果单分片查询包含函数，则仅当满足以下条件之一时，该查询才有资格进行单分片优化：

- 该函数不可变。有关更多信息，请参阅 函数波动性。
- 该函数可变，但已在 `rds_aurora.limitless_distributed_functions` 视图中注册。有关更多信息，请参阅 函数分布。

视图

如果查询包含一个或多个视图，则如果查询具有以下条件之一，将禁用单分片优化：

- 任何一个视图具有 `security_barrier` 属性。
- 查询中使用的对象需要多个用户权限。例如，一个查询包含两个视图，这些视图在两个不同的用户下运行。

```
CREATE VIEW v1 AS SELECT customer_name FROM customers c WHERE c.customer_id =  1;
CREATE VIEW v2 WITH (security_barrier) AS SELECT customer_name FROM customers c
 WHERE c.customer_id =  1;

postgres_limitless=> EXPLAIN VERBOSE SELECT * FROM v1;
                                    QUERY PLAN
-------------------------------------------------------------------------------------
 Foreign Scan  (cost=100.00..101.00 rows=100 width=0)
   Output: customer_name
   Remote Plans from Shard postgres_s3:
         Seq Scan on public.customers_ts00001 c  (cost=0.00..24.12 rows=6 width=32)
           Output: c.customer_name
           Filter: (c.customer_id = 1)
         Query Identifier: -6005737533846718506
   Remote SQL:  SELECT customer_name
    FROM ( SELECT c.customer_name
             FROM public.customers c
            WHERE (c.customer_id = 1)) v1
 Query Identifier: -5754424854414896228
(12 rows)


postgres_limitless=> EXPLAIN VERBOSE SELECT * FROM v2;
                                       QUERY PLAN
-----------------------------------------------------------------------------------------
 Foreign Scan on public.customers_fs00001 c  (cost=100.00..128.41 rows=7 width=32)
   Output: c.customer_name
   Remote Plans from Shard postgres_s3:
         Seq Scan on public.customers_ts00001 customers  (cost=0.00..24.12 rows=6
 width=32)
           Output: customers.customer_name
           Filter: (customers.customer_id = 1)
         Query Identifier: 4136563775490008117
   Remote SQL: SELECT customer_name FROM public.customers WHERE ((customer_id = 1))
 Query Identifier: 5056054318010163757
(9 rows)
```

PREPARE 和 EXECUTE 语句

Aurora PostgreSQL Limitless Database 支持对已准备的 SELECT、UPDATE 和 DELETE 语句进行单分片优化。

但是，如果您为 PREPARE 和 EXECUTE 使用已准备的语句，并将 plan_cache_mode 设置为 'force_generic_plan'，则查询计划程序将拒绝该查询的单分片优化。

PL/pgSQL

带有 PL/pgSQL 变量的查询将作为隐式预处理语句运行。如果查询包含任何 PL/pgSQL 变量，则查询计划器会拒绝单分片优化。

如果语句不包含任何 PL/pgSQL 变量，则支持在 PL/pgSQL 块中进行优化。

完全限定（显式）联接

单分片优化基于分区消除。PostgreSQL 优化器会根据恒定条件消除分区。如果 Aurora PostgreSQL Limitless Database 发现所有剩余的分区和表都在同一个分片上，则它会将查询标记为符合单分片优化的条件。所有筛选条件都必须是显式的，分区消除才能起作用。如果语句中每个分片表的分片键上没有一个或多个联接谓词或筛选谓词，则 Aurora PostgreSQL Limitless Database 无法消除分区。

假设我们已经根据 customer_id 列对 customers、orders 和 order_details 表进行了分区。在此架构下，应用程序会尝试将客户的所有数据保存在单个分片上。

请考虑以下查询：

```
SELECT * FROM
    customers c, orders o, order_details od
WHERE c.customer_id = o.customer_id
    AND od.order_id = o.order_id
    AND c.customer_id = 1;
```

此查询检索客户的所有数据 (c.customer_id = 1)。该客户的数据位于单个分片上，但是 Aurora PostgreSQL Limitless Database 不能将此查询限定为单分片查询。查询的优化程序流程如下所示：

1. 优化器可以根据以下条件消除 customers 和 orders 的分区：

   ```
   c.customer_id = 1
   c.customer_id = o.customer_id
   o.customer_id =  1 (transitive implicit condition)
   ```

2. 优化器无法消除 order_details 中的任何分区，因为表中没有恒定条件。
3. 优化器得出结论，它已经从 order_details 中读取了所有分区。因此，该查询不符合单分片优化的资格。

为了使其成为单分片查询，我们添加了以下显式联接条件：

```
o.customer_id = od.customer_id
```

更改后的查询类似于以下示例：

```
SELECT * FROM
    customers c, orders o,  order_details od
WHERE c.customer_id = o.customer_id
     AND o.customer_id = od.customer_id
     AND od. order_id = o. order_id
 AND c.customer_id =  1;
```

现在，优化器可以消除 order_details 的分区。新查询变为单分片查询并有资格进行优化。

设置活动分片密钥

此功能允许您在查询数据库时设置单个分片键，从而使所有 SELECT 和 DML 查询都将分片键作为常量谓词附加到数据库中。如果您已迁移到 Aurora PostgreSQL Limitless Database，并已通过向表中添加分片键对架构进行去规范化处理，则此功能非常有用。

您可以自动将分片键谓词附加到现有 SQL 逻辑中，而不必更改查询的语义。添加活动分片键谓词仅适用于兼容表。

活动分片键功能使用 rds_aurora.limitless_active_shard_key 变量，其语法如下：

```
SET [session | local] rds_aurora.limitless_active_shard_key = '{"col1_value",
 "col2_value", ...}';
```

关于活动分片键和外键的一些注意事项：

- 如果父表和子表并置且外键是分片键的超集，则分片表可能具有外键约束。
- 分片表可以对引用表有外键约束。
- 引用表可以对另一个引用表有外键约束。

假设我们有一个在 customer_id 列上进行了分片的 customers 表。

```
BEGIN;
```

```
SET local rds_aurora.limitless_create_table_mode='sharded';
SET local rds_aurora.limitless_create_table_shard_key='{"customer_id"}';
CREATE TABLE customers(customer_id int PRIMARY KEY, name text , email text);
COMMIT;
```

如果设置了活跃的分片键，则查询会进行以下转换。

SELECT

```
SET rds_aurora.limitless_active_shard_key = '{"123"}';
SELECT * FROM customers;

-- This statement is changed to:
SELECT * FROM customers WHERE customer_id = '123'::int;
```

INSERT

```
SET rds_aurora.limitless_active_shard_key = '{"123"}';
INSERT INTO customers(name, email) VALUES('Alex', 'alex@example.com');

-- This statement is changed to:
INSERT INTO customers(customer_id, name, email) VALUES('123'::int, 'Alex',
 'alex@example.com');
```

UPDATE

```
SET rds_aurora.limitless_active_shard_key = '{"123"}';
UPDATE customers SET email = 'alex_new_email@example.com';

-- This statement is changed to:
UPDATE customers SET email = 'alex_new_email@example.com' WHERE customer_id =
 '123'::int;
```

DELETE

```
SET rds_aurora.limitless_active_shard_key = '{"123"}';
DELETE FROM customers;

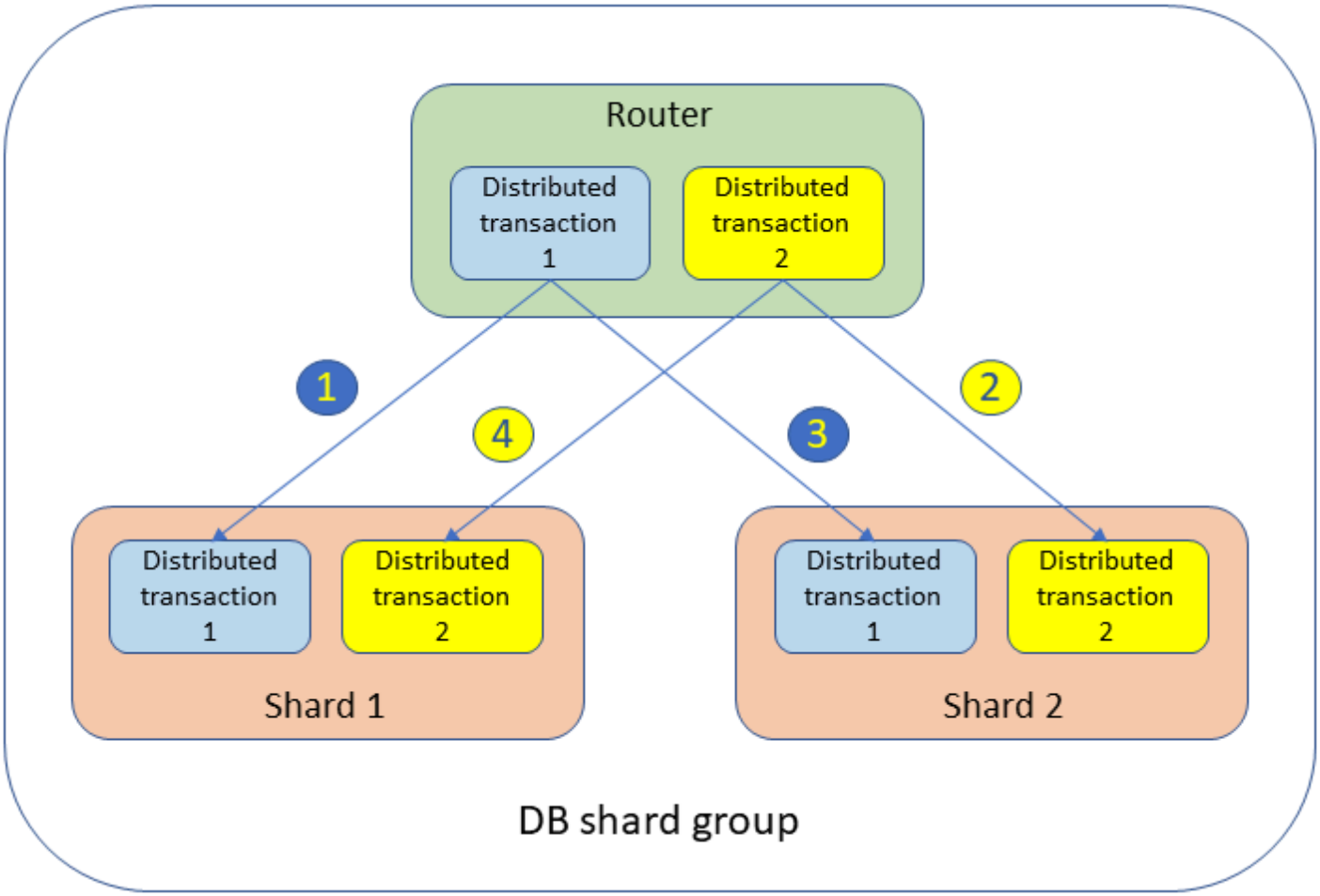
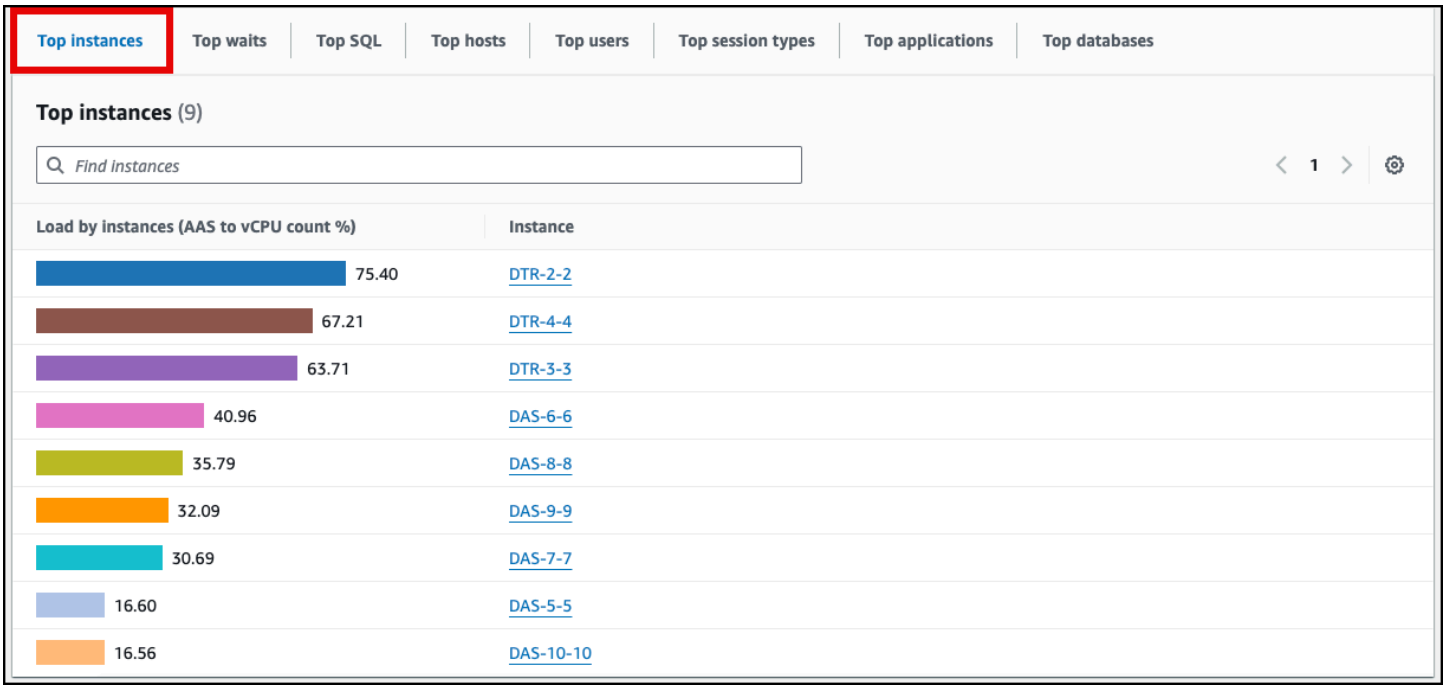
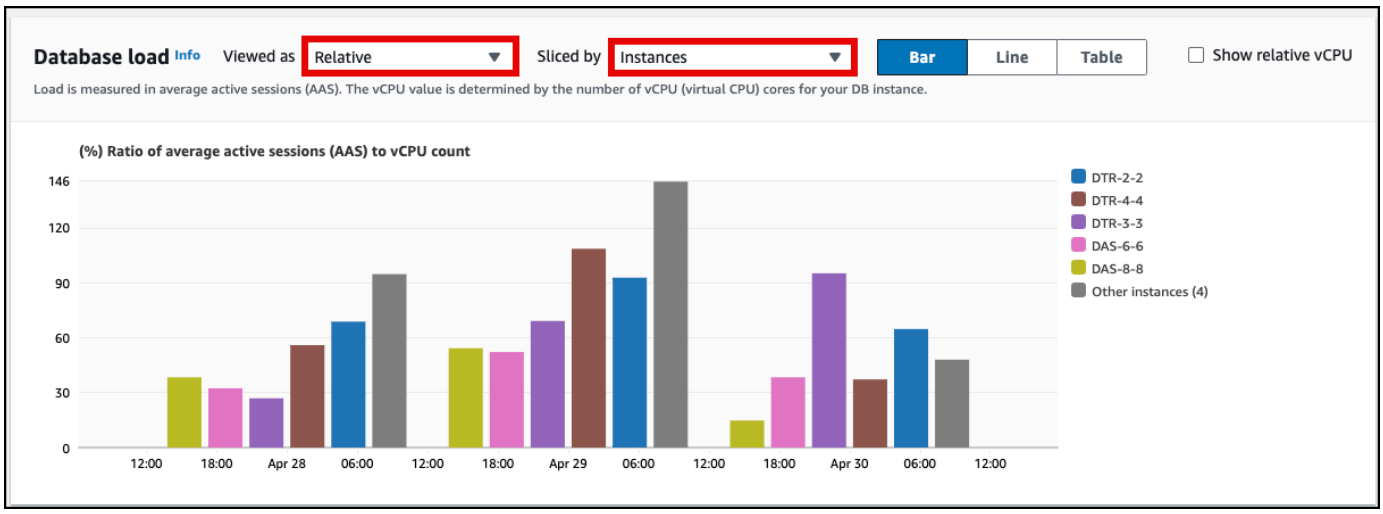
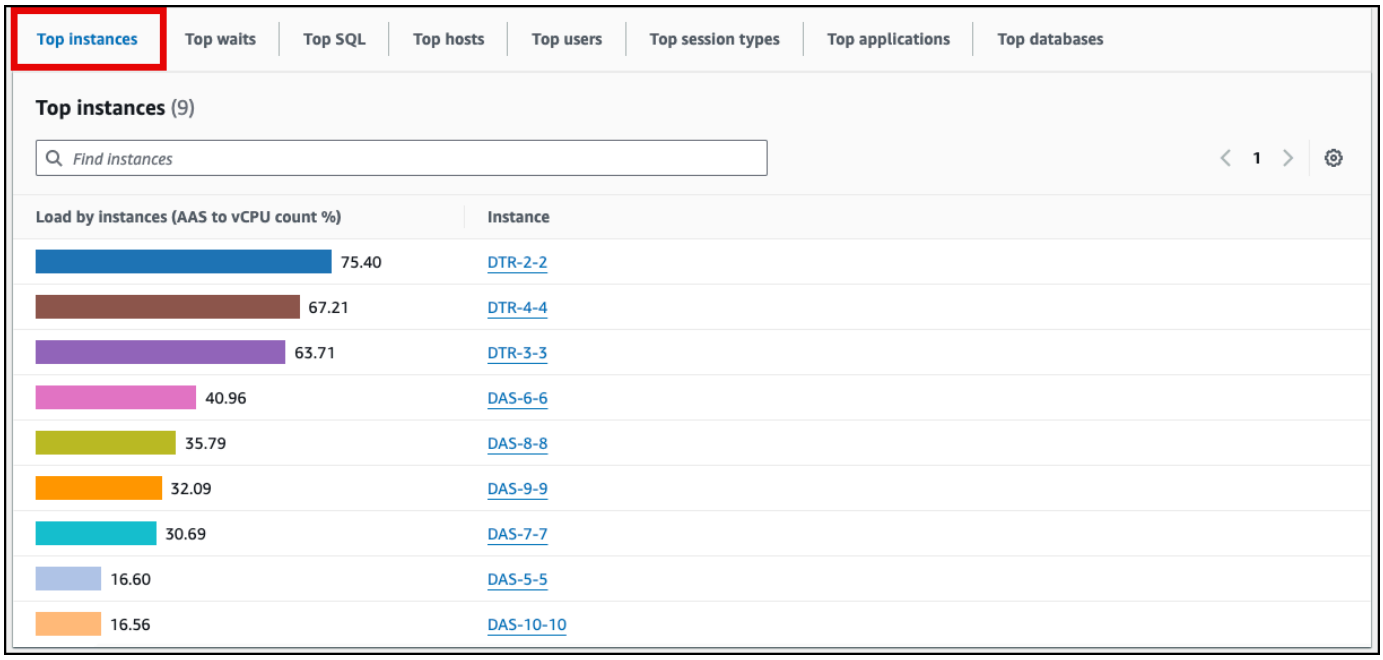
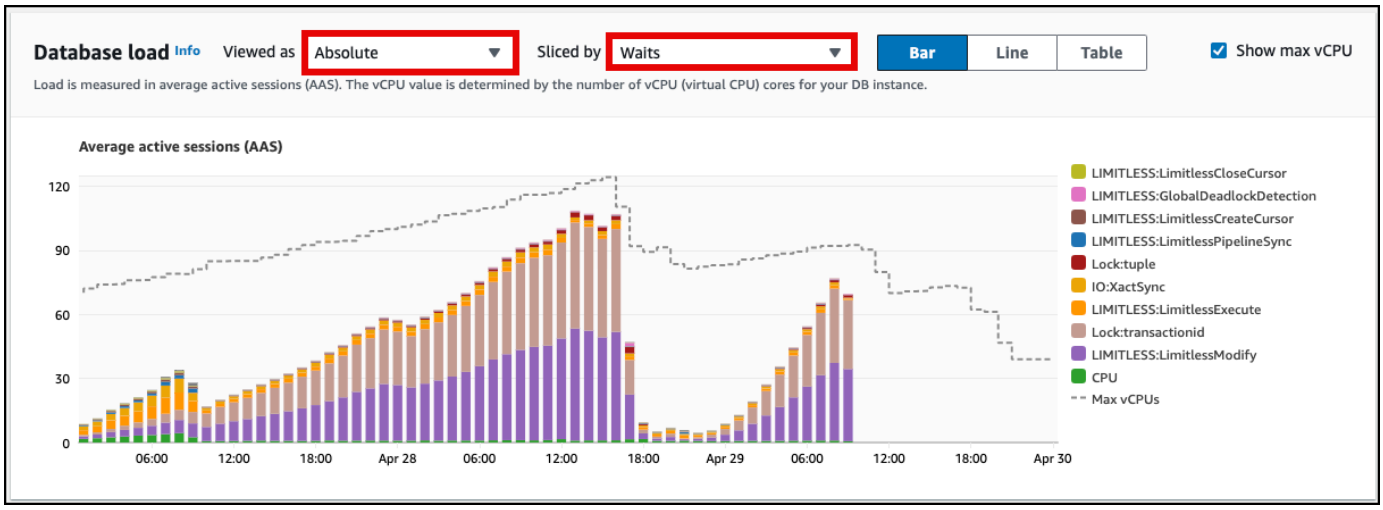
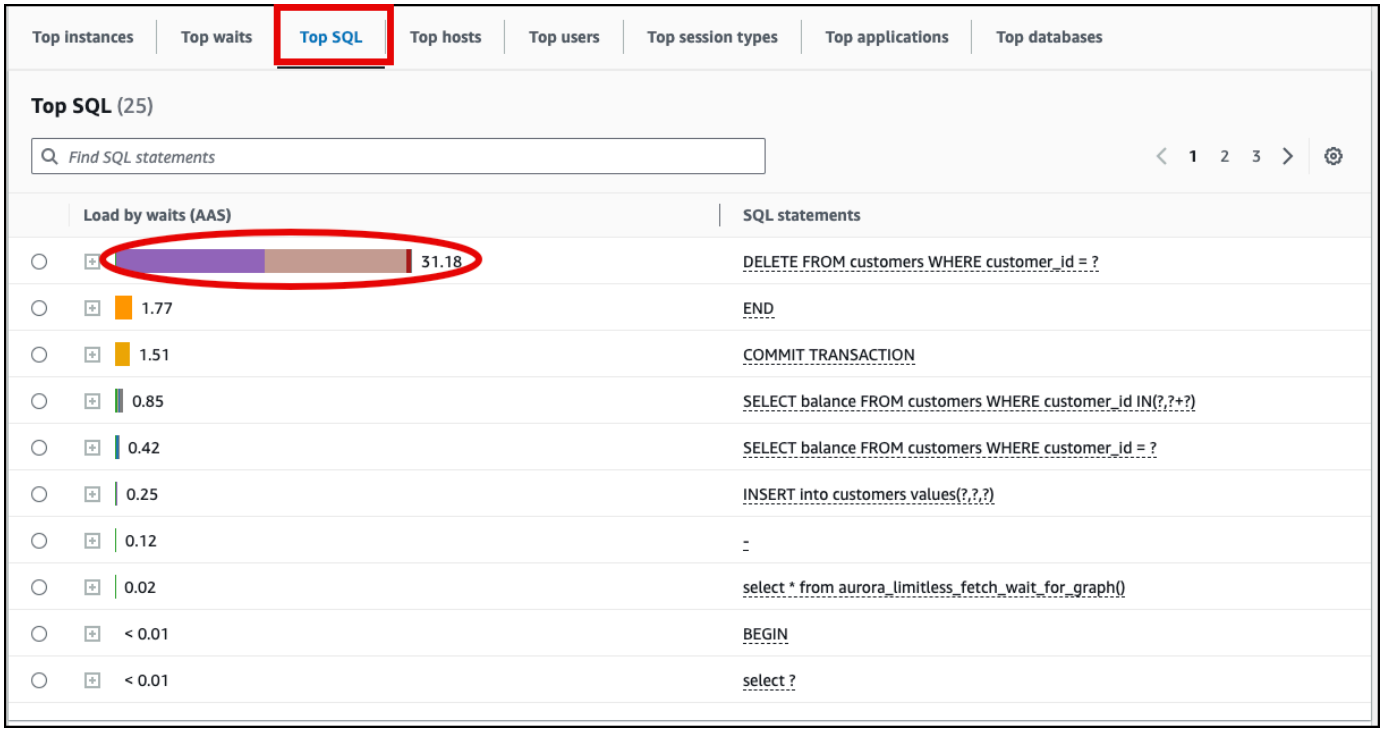
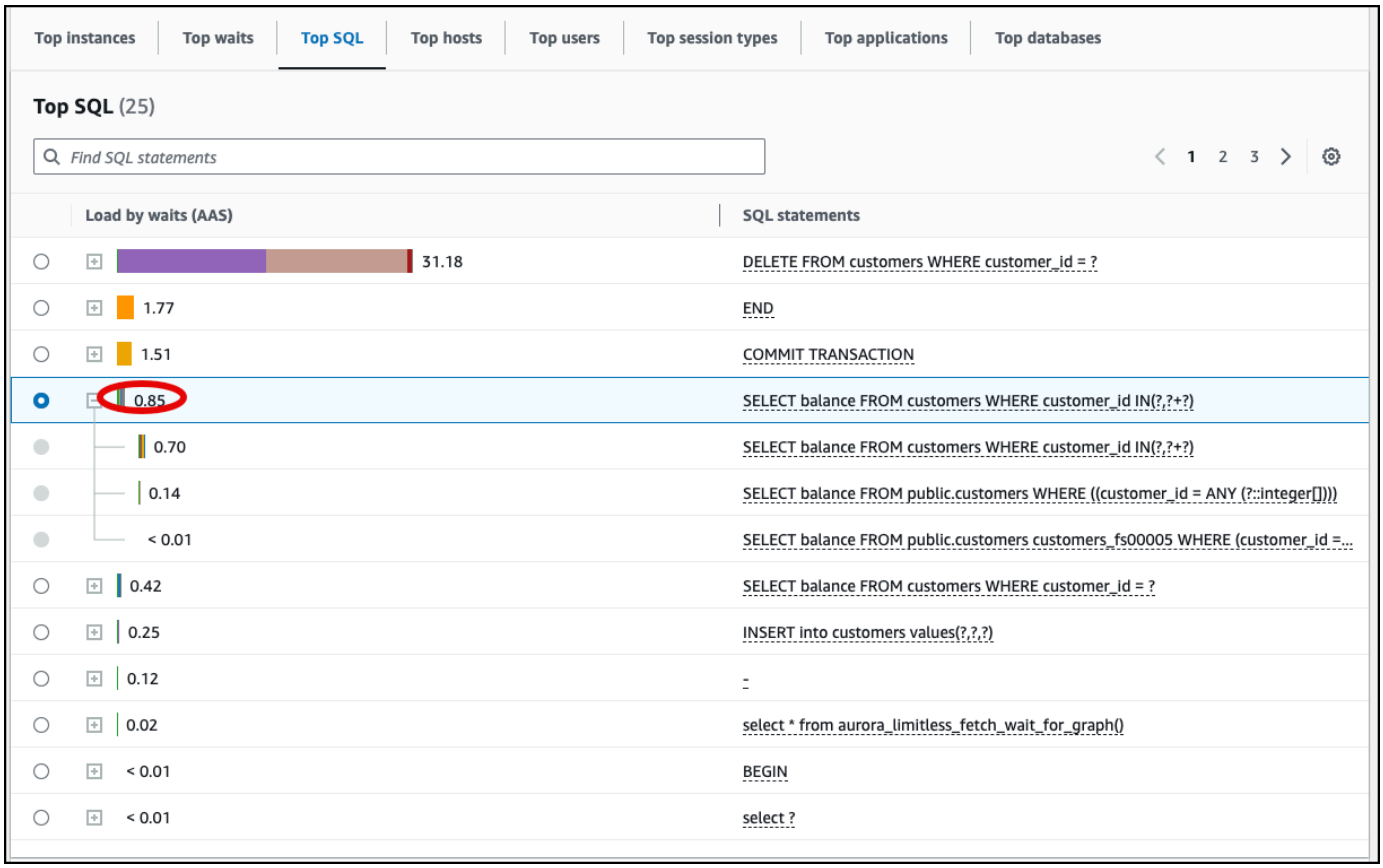
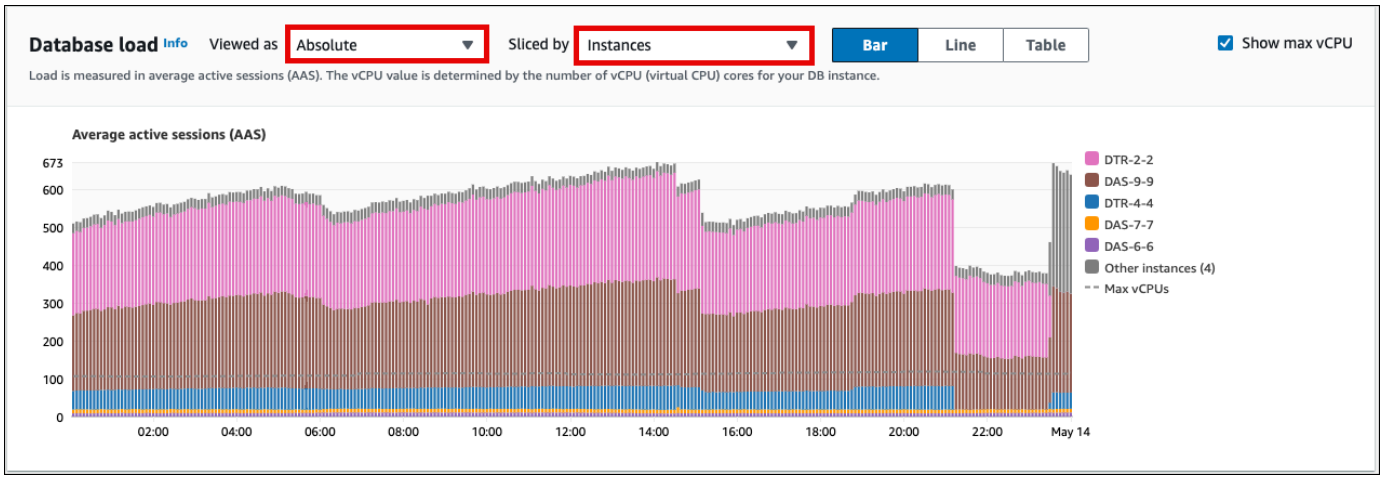
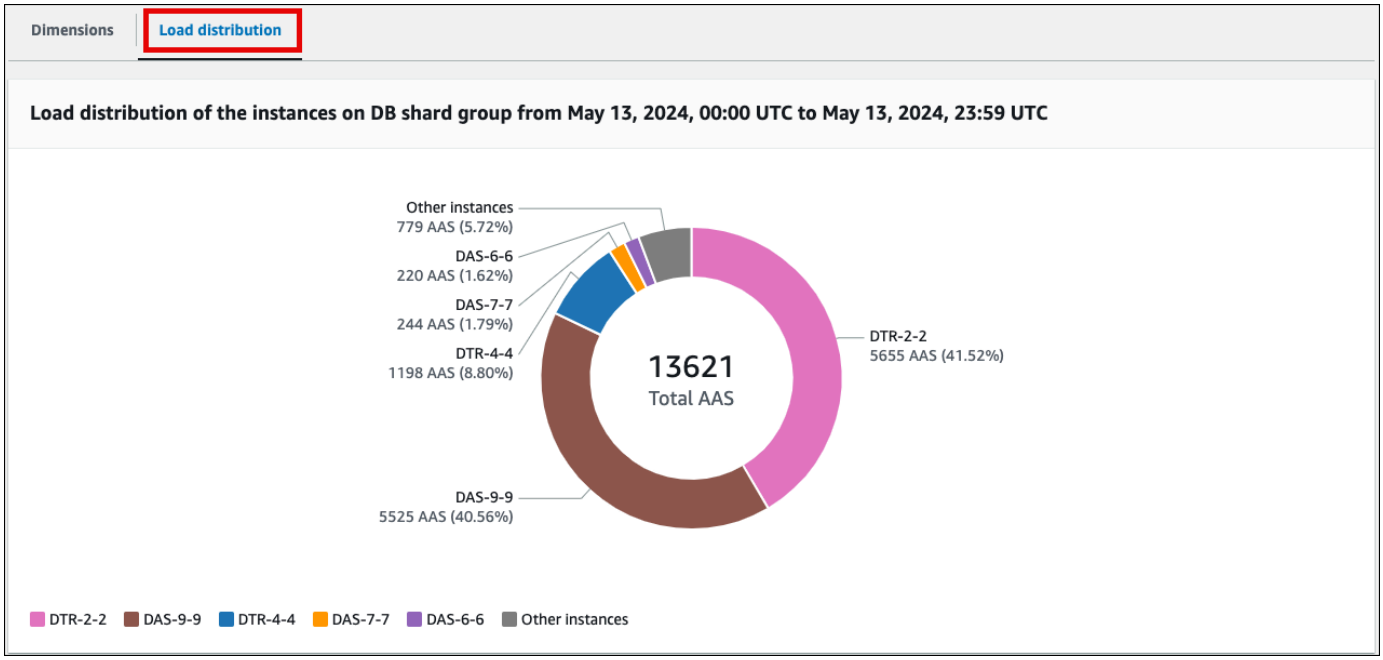
-- This statement is changed to:
DELETE FROM customers WHERE customer_id = '123'::int;
```

联接

对具有活动分片键的表执行联接操作时，分片键谓词会自动添加到联接中涉及的所有表中。只有当查询中的所有表都属于同一个搭配组时，才会自动添加分片键谓词。如果查询涉及来自不同搭配组的表，则会引发错误。

假设我们还有 orders 和 order_details 表与 customers 表并置。

```
SET local rds_aurora.limitless_create_table_mode='sharded';
SET local rds_aurora.limitless_create_table_collocate_with='customers';
SET local rds_aurora.limitless_create_table_shard_key='{"customer_id"}';
CREATE TABLE orders (id int , customer_id int, total_amount int, date date);
CREATE TABLE order_details (id int , order_id int, customer_id int, product_name
 VARCHAR(100), price int);
COMMIT;
```

检索客户 ID 为 10 的客户的最近 10 张订单发票。

```
SET rds_aurora.limitless_active_shard_key = '{"10"}';
SELECT * FROM customers, orders, order_details WHERE
    orders.customer_id = customers.customer_id AND
    order_details.order_id = orders.order_id AND
    customers.customer_id = 10
    order by order_date limit 10;
```

该查询转换为以下内容：

```
SELECT * FROM customers, orders, order_details WHERE
    orders.customer_id = customers.customer_id AND
    orders.order_id = order_details.order_id AND
    customers.customer_id = 10 AND
    order_details.customer_id = 10 AND
    orders.customer_id = 10 AND
    ORDER BY "order_date" LIMIT 10;
```

活动分片键兼容表

分片键谓词仅添加到与活动分片键兼容的表中。如果表的分片键中的列数与 rds_aurora.limitless_active_shard_key 变量中指定的列数相同，则该表被视为兼容。如果查询涉及与活动分片键不兼容的表，则系统会提示错误并停止查询。

例如：

```
-- Compatible table
SET rds_aurora.limitless_active_shard_key = '{"10"}';

-- The following query works because the customers table is sharded on one column.
SELECT * FROM customers;

-- Incompatible table
SET rds_aurora.limitless_active_shard_key = '{"10","20"}';

-- The following query raises a error because the customers table isn't sharded on
 two columns.
 SELECT * FROM customers;
```

Aurora PostgreSQL Limitless Database 中的分布式查询

分布式查询在路由器和多个分片上运行。查询由其中一台路由器接收。路由器创建并管理分布式事务，该事务将发送到参与的分片。分片使用路由器提供的上下文创建本地事务，然后运行查询。

提交事务时，如果需要，路由器会使用优化的两阶段提交协议，并使用基于时间的多版本并发控制（MVCC）在分布式数据库系统中提供 ACID 语义。

基于时间的 MVCC 记录每笔事务的提交时间，并使用事务开始时间生成数据快照时间。为了确定给定阅读器快照的事务是否已提交（可见），数据库会将其提交时间与快照时间进行比较。如果它的提交时间小于阅读器的快照时间，则它可见；否则，不可见。在此协议下，您将始终有望在 Aurora PostgreSQL Limitless Database 上看到高度一致的数据。

Aurora PostgreSQL Limitless Database 中 PostgreSQL 日志中的分布式查询跟踪

分布式查询跟踪是一种在 Aurora PostgreSQL Limitless Database 中跟踪和关联 PostgreSQL 日志中查询的工具。在 Aurora PostgreSQL 中，您可以使用事务 ID 来识别事务。但是在 Aurora PostgreSQL Limitless Database 中，事务 ID 可以在不同路由器上重复。因此，建议您在 Limitless Database 中改用跟踪 ID。

以下是一些关键用例：

- 使用 `rds_aurora.limitless_get_last_trace_id()` 函数查找当前会话中最后一次运行的查询的唯一跟踪 ID。然后使用该跟踪 ID 在 Amazon CloudWatch Logs 中搜索数据库集群日志组，以查找所有相关日志。

  您可以同时使用 `log_min_messages` 和 `log_min_error_statement` 参数来控制打印的日志量，并打印包含跟踪 ID 的语句。
- 使用 `log_min_duration_statement` 参数可确定运行时间，超过该时间所有查询都会打印其运行时长和跟踪 ID。然后可以在 CloudWatch Logs 的数据库集群日志组中搜索该运行时间，以帮助确定瓶颈节点和计划器优化工作。

  无论 `log_min_messages` 和 `log_min_error_statement` 参数的值如何，`log_min_duration_statement` 参数都会启用所有节点的跟踪 ID。

主题

- 跟踪 ID
- 使用查询跟踪
- 日志示例

跟踪 ID

此功能的核心是一个被称为跟踪 ID 的唯一标识符。跟踪 ID 是一个附加到 PostgreSQL 日志的 STATEMENT 日志行中的 31 位数字字符串，是与特定查询执行相关的日志的精确标识符。示例包括 `11262533757194085040000000000011` 和 `11262533757194084950000000000090`。

跟踪 ID 由以下内容组成：

- 事务标识符 – 前 20 位数字，事务的唯一标识。

- 命令标识符 – 前 30 位数字，表示事务中的单个查询。

  如果在显式事务块中运行的查询超过 4,294,967,294 个，则命令标识符将变为 1。发生这种情况时，PostgreSQL 日志中会显示以下 LOG 消息：

  ```
  wrapping around the tracing ID back to 1 after running 4294967294 (4.2 billion or
   2^32-2) queries inside of an explicit transaction block
  ```

- 节点类型标识符 – 最后一位数字，表示该节点是用作协调器路由器 (1) 还是参与者节点 (0)。

以下示例演示了跟踪 ID 的组成部分：

- 1126253375719408504000000000011:
  - 事务标识符 – 1126253375719408504
  - 命令标识符 – 1126253375719408504000000000001 表示事务块中的第一个命令
  - 节点类型标识符 – 1 表示协调器路由器
- 1126253375719408495000000000090:
  - 事务标识符 – 1126253375719408495
  - 命令标识符 – 1126253375719408495000000000009 表示事务块中的第九个命令
  - 节点类型标识符 – 0 表示参与者节点

使用查询跟踪

执行以下任务以使用查询跟踪：

1. 确保跟踪已启用。

   您可以使用以下命令进行检查：

   ```
   SHOW rds_aurora.limitless_log_distributed_trace_id;
   ```

   系统会默认启用该功能 (on)。如果未启用，请使用以下命令进行设置：

   ```
   SET rds_aurora.limitless_log_distributed_trace_id = on;
   ```

2. 通过配置日志严重性级别来控制打印的日志量。

日志量由 log_min_messages 参数控制。log_min_error_statement 参数用于打印带有跟踪 ID 的 STATEMENT 行。默认情况下，两者都设置为 ERROR。您可以使用以下命令进行检查：

```
SHOW log_min_messages;
SHOW log_min_error_statement;
```

要更新严重性级别并打印当前会话的 STATEMENT 行，请按照严重性级别使用以下命令：

```
SET log_min_messages = 'DEBUG5 | DEBUG4 | DEBUG3 | DEBUG2 | DEBUG1 | INFO | NOTICE
 | WARNING | ERROR | LOG | FATAL | PANIC';
SET log_min_error_statement = 'DEBUG5 | DEBUG4 | DEBUG3 | DEBUG2 | DEBUG1 | INFO
 | NOTICE | WARNING | ERROR | LOG | FATAL | PANIC';
```

例如：

```
SET log_min_messages = 'WARNING';
SET log_min_error_statement = 'WARNING';
```

3. 允许在特定运行时间以上的日志中打印跟踪 ID。

   可以将 log_min_duration_statement 参数更改为查询运行时间的最小值，超过该值时，查询将打印一条包含运行持续时间的日志行以及数据库集群中的跟踪 ID。默认情况下，此参数设置为 -1，这意味着它已被禁用。您可以使用以下命令进行检查：

   ```
   SHOW log_min_duration_statement;
   ```

   将其更改为 0，在日志中打印数据库集群中每次查询的持续时间和跟踪 ID。您可以使用以下命令在当前会话中将其设置为 0：

   ```
   SET log_min_duration_statement = 0;
   ```

4. 获取跟踪 ID。

   运行查询后（即使在显式事务块内也是如此），调用 rds_aurora.limitless_get_last_trace_id 函数以获取上次运行的查询的跟踪 ID：

   ```
   SELECT * FROM rds_aurora.limitless_get_last_trace_id();
   ```

此函数返回事务标识符和命令标识符。它不返回节点类型标识符。

```
=> SELECT * FROM customers;
 customer_id | fname | lname
-------------+-------+-------
(0 rows)

=> SELECT * FROM rds_aurora.limitless_get_last_trace_id();
 transaction_identifier |       command_identifier
------------------------+--------------------------------
 10104661421959001813   | 101046614219590018130000000001
(1 row)
```

该函数为非分布式查询返回一个空白行，因为它们没有跟踪 ID。

```
=> SET search_path = public;
SET

=> SELECT * FROM rds_aurora.limitless_get_last_trace_id();
 transaction_identifier | command_identifier
------------------------+--------------------
                        |
(1 row)
```

> **Note**
>
> 对于 VACUUM 和 ANALYZE 查询，不会使用跟踪 ID 记录持续时间语句。因此，`limitless_get_last_trace_id()` 不返回跟踪 ID。如果 VACUUM 或 ANALYZE 是一个长期运行的操作，则可以使用以下查询获取该操作的跟踪 ID：
>
> ```
> SELECT * FROM rds_aurora.limitless_stat_activity
> WHERE distributed_tracing_id IS NOT NULL;
> ```
>
> 如果服务器在您找到最后一个跟踪 ID 之前就停止了，则必须手动搜索 PostgreSQL 日志，从日志中找到失效前的跟踪 ID。

5. 使用 CloudWatch 在数据库集群日志中搜索跟踪 ID。

   使用 CloudWatch Insights 查询数据库集群的日志组，如以下示例所示。

- 查询特定的事务标识符及其内部运行的所有命令：

```
fields @timestamp, @message
| filter @message like /10104661421959001813/
| sort @timestamp desc
```

- 查询特定的命令标识符：

```
fields @timestamp, @message
| filter @message like /10104661421959001813000000000001/
| sort @timestamp desc
```

6. 检查分布式查询在数据库集群中生成的所有日志。

日志示例

以下示例演示了如何使用查询跟踪。

关联日志，查找容易出错的查询

在此示例中，当 customers 表不存在时，会在该表上运行 TRUNCATE 命令。

不使用查询跟踪

协调器路由器上的 PostgreSQL 日志文件：

```
2023-09-26 04:03:19 UTC:[local]: postgres@postgres_limitless:[27503]:ERROR: failed to
 execute remote query
2023-09-26 04:03:19 UTC:[local]: postgres@postgres_limitless:[27503]:DETAIL: relation
 "public.customers" does not exist
2023-09-26 04:03:19 UTC:[local]: postgres@postgres_limitless:[27503]:CONTEXT: remote
 SQL command: truncate public.customers;
2023-09-26 04:03:19 UTC:[local]: postgres@postgres_limitless:[27503]:STATEMENT:
 truncate customers;
```

参与者分片上的 PostgreSQL 日志文件：

```
2023-09-26 04:03:19 UTC:[local]: postgres@postgres_limitless:[ 27503]:ERROR: failed to
 execute remote query
2023-09-26 04:03:19 UTC:[local]: postgres@postgres_limitless:[ 27503]:STATEMENT:
 truncate customers;
```

这些日志具有代表性。它们缺乏在数据库集群中轻松关联查询所需的精确标识符。

使用查询跟踪

协调器路由器上的 PostgreSQL 日志文件：

```
2023-09-26 04:03:19 UTC:[local]:postgres@postgres_limitless:[27503]:ERROR: failed to
 execute remote query
2023-09-26 04:03:19 UTC:[local]:postgres@postgres_limitless:[27503]:DETAIL: relation
 "public.customers" does not exist
2023-09-26 04:03:19 UTC:[local]:postgres@postgres_limitless:[27503]:CONTEXT: remote SQL
 command: truncate public.customers;
2023-09-26 04:03:19 UTC:[local]:postgres@postgres_limitless:[27503]:STATEMENT: /* tid =
 1126253375719408502700000000011 */ truncate customers;
```

参与者分片上的 PostgreSQL 日志文件：

```
2023-09-26 04:03:19 UTC:[local]:postgres@postgres_limitless:[27503]:ERROR:  failed to
 execute remote query
2023-09-26 04:03:19 UTC:[local]:postgres@postgres_limitless:[27503]:STATEMENT:  /* tid
 = 1126253375719408502700000000010 */ truncate customers;
```

在进行查询跟踪时，每个日志行都会附加一个 31 位的唯一标识符。这里，1126253375719408502700000000011 和 1126253375719408502700000000010 分别代表协调器和参与者节点的跟踪 ID。

- 事务标识符 – 11262533757194085027
- 命令标识符 – 1126253375719408502700000000001
- 节点类型标识符 – 最后一位数字 1 或 0 分别表示协调器路由器和参与者节点。

关联日志以查找各个节点上的查询运行时间

在此示例中，log_min_duration_statement 参数已更新为 0，以打印所有查询的持续时间。

不使用查询跟踪

```
2024-01-15 07:28:46 UTC:[local]:postgres@postgres_limitless:[178322]:LOG: duration:
 12.779 ms statement: select * from customers;
```

使用查询跟踪

协调器路由器上的 PostgreSQL 日志文件：

```
2024-01-15 07:32:08 UTC:[local]:postgres@postgres_limitless:[183877]:LOG: duration:
 12.618 ms statement: /* tid = 045766956624049708840000000011 */ select * from
 customers;
```

参与者分片上的 PostgreSQL 日志文件：

```
2024-01-15 07:32:08 UTC:localhost(46358):postgres@postgres_limitless:[183944]:LOG:
 duration: 0.279 ms statement: /* tid = 045766956624049708840000000010 */ START
 TRANSACTION ISOLATION LEVEL READ COMMITTED
2024-01-15 07:32:08 UTC:localhost(46358):postgres@postgres_limitless:[183944]:LOG:
 duration: 0.249 ms parse <unnamed>: SELECT customer_id, fname, lname FROM
 public.customers
```

```
2024-01-15 07:32:08 UTC:localhost(46358):postgres@postgres_limitless:[183944]:LOG:
 duration: 0.398 ms bind <unnamed>/c1: SELECT customer_id, fname, lname FROM
 public.customers
2024-01-15 07:32:08 UTC:localhost(46358):postgres@postgres_limitless:[183944]:LOG:
 duration: 0.019 ms execute <unnamed>/c1: SELECT customer_id, fname, lname FROM
 public.customers
2024-01-15 07:32:08 UTC:localhost(46358):postgres@postgres_limitless:[183944]:LOG:
 duration: 0.073 ms statement: /* tid = 045766956624049708840000000010 */ COMMIT
 TRANSACTION
```

Aurora PostgreSQL Limitless Database 中的分布式死锁

在数据库分片组中，分布在不同路由器和分片之间的事务之间可能会发生死锁。例如，运行跨两个分片的两个并发分布式事务，如下图所示。



事务会锁定表并在两个分片中创建等待事件，如下所示：

1. 分布式事务 1：

```
UPDATE table SET value = 1 WHERE key = 'shard1_key';
```

   这会锁定分片 1。

2. 分布式事务 2：

```
UPDATE table SET value = 2 WHERE key = 'shard2_key';
```

这会锁定分片 2。

3. 分布式事务 1：

```
UPDATE table SET value = 3 WHERE key = 'shard2_key';
```

分布式事务 1 正在等待分片 2。

4. 分布式事务 2：

```
UPDATE table SET value = 4 WHERE key = 'shard1_key';
```

分布式事务 2 正在等待分片 1。

在这种情况下，分片 1 和分片 2 都看不到问题：事务 1 在等待分片 2 上的事务 2，而事务 2 在等待分片 1 上的事务 1。从全局角度来看，事务 1 在等待事务 2，事务 2 在等待事务 1。两个不同分片上的两个事务互相等待的情况称为分布式死锁。

Aurora PostgreSQL Limitless Database 可以自动检测和解决分布式死锁。当事务等待获取资源的时间过长时，数据库分片组中的路由器会收到通知。接收通知的路由器开始从数据库分片组中的所有路由器和分片收集必要的信息。然后，路由器会继续结束参与分布式死锁的事务，直到数据库分片组中的其余事务可以在不被彼此阻止的情况下继续进行。

当您的事务是分布式死锁的一部分并由路由器结束时，您会收到以下错误：

```
ERROR: aborting transaction participating in a distributed deadlock
```

`rds_aurora.limitless_distributed_deadlock_timeout` 数据库集群参数设置每个事务在通知路由器检查分布式死锁之前等待资源的时间。如果您的工作负载不太容易出现死锁情况，则可以增加该参数值。默认值为 `1000` 毫秒（1 秒）。

当发现并解决跨节点死锁时，分布式死锁周期将发布到 PostgreSQL 日志中。死锁中每个进程的信息包括以下内容：

- 启动事务的协调器节点
- 协调器节点上事务的虚拟事务 ID（xid），格式为 *backend_id*/*backend_local_xid*
- 事务的分布式会话 ID

管理 Aurora PostgreSQL Limitless Database

以下主题介绍了如何管理 Aurora PostgreSQL Limitless Database 数据库集群。

主题

- 数据库和表大小的考虑事项
- 通过清理回收存储空间

数据库和表大小的考虑事项

对于 Aurora PostgreSQL Limitless Database，在每个分片上，分片表被分成多个表切片，这些切片因数据库分片组中可用分片的数量而异。每个表切片最多可以增长到 32 TiB，但每个分片的最大容量为 128 TiB。整个数据库分片组的引用表大小限制为 32 TiB。

> **Note**
>
> 每个节点（路由器或分片）的最大容量为 128 TiB，因为这是 Aurora PostgreSQL 数据库集群的最大容量。

Aurora PostgreSQL 和 Aurora PostgreSQL Limitless Database 中每个数据库（包括表、视图和索引）的最大关系数均为 1,431,650,303。

有关更多信息，请参阅 PostgreSQL 文档中的 Appendix K. PostgreSQL limits 和 Amazon Aurora 大小限制。

通过清理回收存储空间

PostgreSQL 多版本并发控制（MVCC）通过保存已更新或已删除行的内部副本，帮助保持数据完整性，直到事务提交或回滚。这些副本也称为元组，如果不定期清理，可能会导致表膨胀。PostgreSQL 实例按事务 ID 对事务进行排序，而 PostgreSQL 使用基于事务 ID 的 MVCC 来控制元组的可见性并提供事务隔离。每个事务都会建立数据快照，每个元组都有一个版本。快照和版本均基于事务 ID。

为了清理数据，VACUUM 实用程序在 PostgreSQL 中执行了四个关键功能：

- VACUUM – 移除过期的行版本，使空间可供重复使用。
- VACUUM FULL – 通过移除死行版本并压缩表来提供完整的碎片整理，从而减小表大小并提高效率。
- VACUUM FREEZE – 通过将较旧的行版本标记为冻结，防止事务 ID 环绕问题。
- VACUUM ANALYZE – 移除死行版本并更新数据库的查询计划统计数据。它是 VACUUM 和 ANALYZE 函数的组合。有关 Aurora PostgreSQL Limitless Database 中 ANALYZE 工作原理的更多信息，请参阅ANALYZE。

与 MVCC 一样，在 Aurora PostgreSQL 中进行清理是基于事务 ID 的。如果在清理开始时有正在进行的事务，则不会移除该事务仍然可见的行。

有关 VACUUM 实用程序的更多信息，请参阅 PostgreSQL 文档中的 VACUUM。有关 Aurora PostgreSQL Limitless Database 中 VACUUM 支持的更多信息，请参阅 VACUUM。

主题

- AUTOVACUUM
- Aurora PostgreSQL Limitless Database 中基于时间的清理
- 使用数据库统计数据进行清理
- Aurora PostgreSQL 和 Aurora PostgreSQL Limitless Database 在清理行为上的差异

AUTOVACUUM

Aurora PostgreSQL 使用 VACUUM 和 AUTOVACUUM 实用程序来移除不需要的元组。AUTOVACUUM 和手动 VACUUM 的底层机制相同；唯一的区别是自动化。

Aurora PostgreSQL 和 Aurora PostgreSQL Limitless Database 中的 AUTOVACUUM 是 VACUUM 和 ANALYZE 实用程序的组合。AUTOVACUUM 根据预定义的规则（例如死元组的百分比和插入次数）确定要清理哪些数据库和表。

例如，AUTOVACUUM 会定期“唤醒”以执行清理。间隔时间由 autovacuum_naptime 参数控制。默认值为 1 分钟。Aurora PostgreSQL Limitless Database 的默认值 AUTOVACUUM 和 VACUUM 配置参数与 Aurora PostgreSQL 相同。

如果启用 AUTOVACUUM 进程守护程序，只要表的内容发生足够大的变化，它就会自动发出 ANALYZE 命令。在 Aurora PostgreSQL Limitless Database 中，AUTOVACUUM 会在路由器和分片上发出 ANALYZE。

有关 AUTOVACUUM 进程守护程序和与 AUTOVACUUM 关联的表存储参数的更多信息，请参阅 PostgreSQL 文档中的 The autovacuum daemon 和 Storage parameters。

Aurora PostgreSQL Limitless Database 中基于时间的清理

Aurora PostgreSQL Limitless Database 是一个分布式系统，这意味着一个事务可以涉及多个实例。因此，基于事务 ID 的可见性不适用。相反，Aurora PostgreSQL Limitless Database 使用基于时间的可见性，因为事务 ID 无法在不同实例之间“统一”，但时间可以在不同实例之间“统一”。每个事务快照和每个元组版本都遵循时间而不是事务 ID。更具体地说，事务快照有快照开始时间，元组有创建时间（发生 INSERT 或 UPDATE 时）和删除时间（发生 DELETE 时）。

为了保持数据库分片组中实例间的数据一致性，Aurora PostgreSQL Limitless Database 必须确保清理不会移除数据库分片组中任何活动事务仍可见的任何元组。因此，Aurora PostgreSQL Limitless Database 中的清理也是基于时间的。VACUUM 的其他方面保持不变，包括要在特定表上运行 VACUUM，用户必须有访问该表的权限。

> **Note**
>
> 我们强烈建议您不要让事务长时间处于开放状态。
> 基于时间的清理会比基于事务 ID 的清理消耗更多的内存。

以下示例说明了基于时间的清理的工作原理。

1. 客户表分布在四个分片上。
2. 事务 1 从可重复读取开始，并且仅针对一个分片（分片 1）。此事务仍处于开放状态。

   事务 1 早于其后开始的任何其他事务。
3. 事务 2 稍后开始，从表中删除所有元组，然后提交。
4. 如果 AUTOVACUUM 或手动 VACUUM 尝试清理死元组（由于事务 2 而死亡），它不会移除任何内容。

这不仅适用于分片 1，也适用于分片 2–4，因为事务 1 可能仍需要访问这些元组。由于 MVCC，它们仍然对事务 1 可见。

最后一步是通过同步实现的，因此所有分片都知道事务 1，即使事务 1 并未触及所有分片。

使用数据库统计数据进行清理

要获取有关可能需要清理的元组的信息，请使用 limitless_stat_all_tables 视图，其工作原理与 pg_stat_all_tables 类似。以下示例查询了视图。

```
SELECT * FROM rds_aurora.limitless_stat_all_tables WHERE relname LIKE '%customer%';
```

同样，对于数据库统计数据，请使用 limitless_stat_database 代替 pg_stat_database，使用 limitless_stat_activity 代替 pg_stat_activity。

要检查表磁盘使用情况，请使用 limitless_stat_relation_sizes 函数，其工作原理与 pg_relation_size 类似。以下示例查询函数。

```
SELECT * FROM rds_aurora.limitless_stat_relation_sizes('public','customer');
```

要在 Aurora PostgreSQL Limitless Database 上跟踪 VACUUM 操作的进度，请使用 limitless_stat_progress_vacuum 视图，而不是 pg_stat_progress_vacuum。以下示例查询了视图。

```
SELECT * FROM rds_aurora.limitless_stat_progress_vacuum;
```

有关更多信息，请参阅Aurora PostgreSQL Limitless Database 视图 和Aurora PostgreSQL Limitless Database 函数。

Aurora PostgreSQL 和 Aurora PostgreSQL Limitless Database 在清理行为上的差异

Aurora PostgreSQL 和 Aurora PostgreSQL Limitless Database 在清理工作原理方面的一些其他区别如下：

- Aurora PostgreSQL 对事务 ID 执行 VACUUM 操作，直到最早的正在进行的事务。如果数据库中没有正在进行的事务，则 VACUUM 执行该操作直到最后一个事务。
- Aurora PostgreSQL Limitless Database 每 10 秒同步一次最早的时间快照。因此，VACUUM 可能不会对过去 10 秒内运行的任何事务执行该操作。

有关 Aurora PostgreSQL Limitless Database 中 VACUUM 支持的信息，请参阅 Aurora PostgreSQL Limitless Database 参考中的 VACUUM。

监控 Aurora PostgreSQL Limitless Database

您可以使用 Amazon CloudWatch、增强监控和性能详情来监控 Aurora PostgreSQL Limitless Database。Aurora PostgreSQL Limitless Database 还有新的统计函数和视图以及等待事件，可用于监控和诊断。

主题

- 使用 Amazon CloudWatch 监控 Aurora PostgreSQL Limitless Database
- 使用 CloudWatch Database Insights 监控 Aurora PostgreSQL 无限制数据库
- 使用 Amazon CloudWatch Logs 监控 Aurora PostgreSQL Limitless Database
- 使用增强监控指标监控 Aurora PostgreSQL Limitless Database
- 使用性能详情监控 Aurora PostgreSQL Limitless Database
- 使用 Amazon GuardDuty RDS 保护功能监控 Aurora PostgreSQL Limitless Database
- Aurora PostgreSQL Limitless Database 的函数和视图
- Aurora PostgreSQL Limitless Database 的等待事件

使用 Amazon CloudWatch 监控 Aurora PostgreSQL Limitless Database

Aurora PostgreSQL Limitless Database 的 CloudWatch 指标按以下维度报告：

- DBShardGroup
- DBShardGroupRouterAggregation
- DBShardGroupInstance
- DBClusterIdentifier

有关 CloudWatch 指标的更多信息，请参阅使用 Amazon CloudWatch 监控 Amazon Aurora 指标。

DBShardGroup 指标

要在 CloudWatch 控制台中查看 Aurora PostgreSQL Limitless Database 的 `DBShardGroup` 指标，请选择 RDS，然后选择 DBShardGroup。

您可以跟踪以下 CloudWatch 指标：

- `DBShardGroupACUUtilization` – 根据 `DBShardGroupCapacity` 除以 `DBShardGroupMaxACU` 计算得出的 Aurora 容量单位（ACU）使用百分比。
- `DBShardGroupCapacity`：数据库分片组的写入器实例使用的 ACU 数量。
- `DBShardGroupComputeRedundancyCapacity`：数据库分片组的备用实例使用的 ACU 数量。
- `DBShardGroupMaxACU` – 为数据库分片组配置的最大 ACU 数量。
- `DBShardGroupMinACU` – 数据库分片组所需的最少 ACU 数量。

`DBShardGroupIdentifier` 维度键可用于聚合 `DBShardGroup` 指标。

DBShardGroupRouterAggregation 指标

要在 CloudWatch 控制台中查看 Aurora PostgreSQL Limitless Database 的 `DBShardGroupRouterAggregation` 指标，请选择 RDS，然后选择 DBShardGroupRouterAggregation。

您可以跟踪以下 CloudWatch 指标：

- `CommitThroughput` – 数据库分片组中所有路由器节点的每秒平均提交操作数。
- `DatabaseConnections` – 数据库分片组中所有路由器节点上所有连接的总和。

DBShardGroupInstance 指标

DBShardGroupInstance 是每个分片或路由器子集群中的单个数据库实例。

要在 CloudWatch 控制台中查看 Aurora PostgreSQL Limitless Database 的 `DBShardGroupInstance` 指标，请选择 RDS，然后选择 DBShardGroupInstance。

您可以跟踪以下 CloudWatch 指标：

- `ACUUtilization`：按 `ServerlessDatabaseCapacity` 指标除以子集群的最大分配 ACU 值计算得出的百分比。
- `AuroraReplicaLag`：对于已启用计算冗余的无限制集群，这是从子集群中的主实例中复制更新时的滞后量。
- `AuroraReplicaLagMaximum`：对于已启用计算冗余的无限制集群，这是从子集群中的主实例中复制更新时的最大滞后量。删除或重命名只读副本时，复制滞后可能会暂时达到峰值，因为旧资源会进行回收。使用此指标来确定失效转移是否因其中一个读取器的复制滞后过高而发生的。
- `AuroraReplicaLagMinimum`：对于已启用计算冗余的无限制集群，这是从子集群中的主实例中复制更新时的最小滞后量。
- `BufferCacheHitRatio` – 实例的内存缓存（相对于存储卷）提供的数据和索引的百分比。
- `CommitLatency` – 引擎和存储完成特定节点（路由器或分片）的提交操作所花费的平均持续时间。
- `CommitThroughput` – 每秒平均提交操作数量。
- `CPUUtilization`：CPU 使用率占子集群的已分配最大 ACU 值的百分比。
- `FreeableMemory`：当分片组扩展到其最大容量时可用的未使用内存量。这由分片组的已分配 ACU 决定。对于当前容量低于最大容量的每个 ACU，此值增加大约 2 GiB。因此，在数据库分片组纵向扩展到最大限制之前，此指标不会接近零。
- `MaximumUsedTransactionIDs` – 事务中最早的未执行清理操作的事务 ID 的期限。如果此值达到 2146483648 (2^31 - 1000000)，则强制数据库进入只读模式，避免事务 ID 重现。有关更多信息，请参阅 PostgreSQL 文档中的防止事务 ID 重现故障。
- `NetworkReceiveThroughput` – 数据库分片组中每个实例从客户端接收的网络吞吐量。此吞吐量不包括数据库分片组中的实例与集群卷之间的网络流量。
- `NetworkThroughput` – 客户端和路由器之间以及数据库分片组中路由器和分片之间的聚合网络吞吐量（传输和接收）。此吞吐量不包括数据库分片组中的实例与集群卷之间的网络流量。
- `NetworkTransmitThroughput` – 数据库分片组中每个实例发送到客户端的网络吞吐量。此吞吐量不包括数据库分片组中的实例与集群卷之间的网络流量。

- ReadIOPS – 磁盘读取每秒进行读写操作（IOPS）的平均次数。
- ReadLatency – 每个磁盘读取输入/输出（I/O）操作所需的平均时间。
- ReadThroughput – 每秒从磁盘读取的平均字节数。
- ServerlessDatabaseCapacity：数据库分片组中数据库分片或路由器子集群的当前容量。
- StorageNetworkReceiveThroughput – 数据库分片组中每个实例从 Aurora 存储子系统接收的网络吞吐量。
- StorageNetworkThroughput – 数据库分片组中每个实例向 Aurora 存储子系统传输和从中接收的聚合网络吞吐量。
- StorageNetworkTransmitThroughput – 数据库分片组中每个实例发送到 Aurora 存储子系统的网络吞吐量。
- SwapUsage：数据库分片组使用的交换空间量。
- TempStorageIOPS – 在附加到数据库实例的本地存储上执行的平均 I/O 操作数。它包括读写 I/O 操作。

  TempStorageIOPS 可与 TempStorageThroughput 一起使用，帮助您诊断数据库实例和本地存储设备之间传输的网络活动导致意外容量增加的罕见情况。
- TempStorageThroughput – 与路由器或分片关联的本地存储的传入或传出数据量。
- WriteIOPS – 磁盘写入 IOPS 的平均数量。
- WriteLatency – 每个磁盘读取 I/O 操作所需的平均时间。
- WriteThroughput – 每秒写入磁盘的平均字节数。

以下维度键可用于聚合 DBShardGroupInstance 指标：

- DBClusterIdentifier – Aurora PostgreSQL 数据库集群。
- DBShardGroupIdentifier – 实例所属的数据库分片组。
- DBShardGroupSubClusterType – 节点类型，可以是 Distributed Transaction Router（路由器）或 Data Access Shard（分片）。
- DBShardGroupSubClusterIdentifier – 实例所属的路由器或分片的名称。

以下是聚合 CloudWatch 指标的示例：

- 属于数据库分片组中特定分片或路由器的所有实例的总体 CPUUtilization。
- 数据库分片组中所有实例的总体 CPUUtilization。

DBClusterIdentifier 指标

要在 CloudWatch 控制台中查看 Aurora PostgreSQL Limitless Database 的 `DBClusterIdentifier` 指标，请选择 RDS，然后选择 DBClusterIdentifier。

当您使用 Aurora PostgreSQL Limitless Database 时，您的输入/输出（I/O）操作可能比使用 Aurora 数据库集群还要多。您可以跟踪 Limitless Database 集群的以下 CloudWatch 指标：

- `VolumeReadIops` – 每隔 5 分钟报告的集群卷中的计费读取 I/O 操作数量。
- `VolumeWriteIops` – 集群卷的写入磁盘 I/O 操作数，每隔 5 分钟报告一次。

Aurora PostgreSQL Limitless Database 使用 Aurora I/O-Optimized 集群存储配置。使用 Aurora I/O-Optimized，您只需每月为所有（而不是每百万次）I/O 操作支付一次费用。有关更多信息，请参阅 Amazon Aurora 数据库集群的存储配置。

与 Aurora 数据库集群相比，您使用的存储空间也可能更多。您可以跟踪以下 CloudWatch 存储指标：

- `BackupRetentionPeriodStorageUsed` – Aurora PostgreSQL Limitless Database 集群的计费持续备份存储总使用量。
- `SnapshotStorageUsed` – Aurora PostgreSQL Limitless Database 集群的计费快照存储总使用量。
- `TotalBackupStorageBilled` – 自动备份保留和数据库集群快照的成本总和。

  有关备份存储成本的更多信息，请参阅了解 Amazon Aurora 备份存储使用量。
- `VolumeBytesUsed` – Aurora PostgreSQL Limitless Database 集群使用的存储量，每 5 分钟报告一次。

使用 CloudWatch Database Insights 监控 Aurora PostgreSQL 无限制数据库

启用 Aurora PostgreSQL 无限制数据库时需要标准模式的 Database Insights。您可以使用该功能来实时监控无限制数据库实例的数据库负载。数据库负载可衡量数据库中会话活动的级别。您可以使用 Database Insights 来大规模分析 Aurora PostgreSQL 无限制数据库实例的性能并排查其故障。

有关 CloudWatch Database Insights 的更多信息，请参阅以下主题。

- 使用 CloudWatch 数据库洞察监控 Amazon Aurora 数据库
- 《Amazon CloudWatch 用户指南》中的 CloudWatch Database Insights
- 《Amazon CloudWatch 用户指南》中的开始使用 CloudWatch Database Insights
- 使用适用于 Amazon Aurora 的数据库洞察配置您的数据库以监控慢速 SQL 查询

有关启用 Database Insights 的高级模式或标准模式的信息，请参阅以下主题。

主题

- 为 Aurora PostgreSQL 无限制数据库开启 Database Insights 的高级模式
- 为 Aurora PostgreSQL 无限制数据库开启 Database Insights 的标准模式

为 Aurora PostgreSQL 无限制数据库开启 Database Insights 的高级模式

要为 Aurora PostgreSQL 无限制数据库开启 Database Insights 的高级模式，请按照以下过程操作。

为 Aurora PostgreSQL 无限制数据库创建数据库集群时开启 Database Insights 的高级模式

为 Aurora PostgreSQL 无限制数据库创建数据库时启用 Database Insights 的高级模式。

Console

在控制台中，您可以在创建数据库集群时开启数据库洞察的高级模式。数据库洞察的设置适用于数据库集群中的所有数据库实例。

在使用控制台创建数据库集群时，开启 Database Insights 的高级模式

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 选择数据库。

3. 选择创建数据库。
4. 在数据库洞察部分，选择高级模式。然后，选择以下选项：
   - 保留 – 保留性能详情数据的时间。对于数据库洞察的高级模式，保留期必须为 15-24 个月。
   - AWS KMS key - 指定您的 KMS 密钥。性能详情使用您的 KMS 密钥来加密所有潜在的敏感数据。正在传输的数据和静态数据都会被加密。有关更多信息，请参阅 加密 Amazon Aurora 资源。
5. 选择创建数据库。

AWS CLI

要在创建数据库集群时开启数据库洞察的高级模式，请调用 create-db-cluster AWS CLI 命令并提供以下值：

- `--db-cluster-identifier` – 数据库集群的标识符。
- `--database-insights-mode advanced`，开启数据库洞察的高级模式。
- `--engine` – 数据库集群必须使用 `aurora-postgresql` 数据库引擎。
- `--engine-version` – 数据库集群必须使用的数据库引擎版本之一：
  - `16.4-limitless`
  - `16.6-limitless`
- `--storage-type` – 数据库集群必须使用 `aurora-iopt1` 数据库集群存储配置。
- `--cluster-scalability-type` – 指定 Aurora 数据库集群的可扩展性模式。设置为 `limitless` 时，集群将作为 Aurora PostgreSQL Limitless Database 运行。设置为 `standard`（默认）时，集群使用正常的数据库实例创建。

  > **Note**
  >
  > 创建数据库集群之后，您将无法修改此设置。

- `--master-username` – 数据库集群的主用户名称。
- `--master-user-password` – 主用户的密码。
- `--enable-performance-insights`，开启数据库洞察的性能详情。
- `--performance-insights-retention-period` – 数据库集群数据的保留期。要开启数据库洞察，保留期必须至少为 465 天。

- `--monitoring-interval` – 收集数据库集群的增强监控指标的时间点之间的间隔，以秒为单位。该值不能为 `0`。
- `--monitoring-role-arn` – 允许 RDS 将增强监控指标发送到 Amazon CloudWatch Logs 的 IAM 角色的 Amazon 资源名称（ARN）。
- `--enable-cloudwatch-logs-exports` – 您必须将 `postgresql` 日志导出到 CloudWatch 日志。

以下示例在创建数据库集群时启用数据库洞察的高级模式。

对于 Linux、macOS 或 Unix：

```
aws rds create-db-cluster \
--db-cluster-identifier my-limitless-cluster \
--database-insights-mode advanced \
--engine aurora-postgresql \
--engine-version 16.6-limitless \
--storage-type aurora-iopt1 \
--cluster-scalability-type limitless \
--master-username myuser \
--master-user-password mypassword \
--enable-performance-insights \
--performance-insights-retention-period 465 \
--monitoring-interval 5 \
--monitoring-role-arn arn:aws:iam::123456789012:role/EMrole \
--enable-cloudwatch-logs-exports postgresql
```

对于 Windows：

```
aws rds create-db-cluster ^
--db-cluster-identifier my-limitless-cluster ^
--database-insights-mode advanced ^
--engine aurora-postgresql ^
--engine-version 16.6-limitless ^
--storage-type aurora-iopt1 ^
--cluster-scalability-type limitless ^
--master-username myuser ^
--master-user-password mypassword ^
--enable-performance-insights ^
--performance-insights-retention-period 465 ^
--monitoring-interval 5 ^
```

```
--monitoring-role-arn arn:aws:iam::123456789012:role/EMrole ^
--enable-cloudwatch-logs-exports postgresql
```

RDS API

要在创建数据库集群时开启数据库洞察的高级模式，请为 CreateDBCluster Amazon RDS API 操作指定以下参数。

- `DatabaseInsightsMode` 到 `advanced`
- `Engine` 到 `aurora-postgresql`
- `EngineVersion` 设置为无限制数据库的可用引擎版本
- `StorageType` 到 `aurora-iopt1`
- `ClusterScalabilityType` 到 `limitless`
- `MasterUsername`
- `MasterUserPassword`
- `EnablePerformanceInsights` 到 `True`
- `PerformanceInsightsRetentionPeriod` 设置为至少 `465` 天
- `MonitoringInterval` 设置为非 `0` 的值
- `MonitoringRoleArn` 设置为 IAM 角色的 Amazon 资源名称（ARN），该角色应可以让 RDS 将增强监控指标发送到 Amazon CloudWatch Logs。

为 Aurora PostgreSQL 无限制数据库修改数据库集群时开启 Database Insights 的高级模式

为 Aurora PostgreSQL 无限制数据库修改数据库时开启 Database Insights。

> **Note**
>
> 要启用 Database Insights，数据库集群中的每个数据库实例都必须具有相同的性能详情和增强监控设置。

Console

在控制台中，您可以在修改数据库集群时开启数据库洞察的高级模式。数据库洞察的设置适用于数据库集群中的所有数据库实例。

在使用控制台修改数据库集群时，开启 Database Insights 的高级模式

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 选择数据库。
3. 选择数据库集群，然后选择修改。
4. 在数据库洞察部分，选择高级模式。然后，选择以下选项：
   - 保留 – 保留性能详情数据的时间。对于数据库洞察的高级模式，保留期必须为 15-24 个月。
   - AWS KMS key - 指定您的 KMS 密钥。性能详情使用您的 KMS 密钥来加密所有潜在的敏感数据。正在传输的数据和静态数据都会被加密。有关更多信息，请参阅“加密 Amazon Aurora 资源”。
5. 选择继续。
6. 对于修改计划，选择立即应用。如果您选择在下一个计划维护时段内应用，则您的数据库将忽略此设置，并立即开启数据库洞察的高级模式。
7. 选择修改集群。

AWS CLI

要在修改数据库集群时开启数据库洞察的高级模式，请调用 modify-db-cluster AWS CLI 命令并提供以下值：

- `--database-insights-mode advanced`，开启数据库洞察的高级模式。
- `--db-cluster-identifier` – 数据库集群的标识符。
- `--enable-performance-insights`，开启数据库洞察的性能详情。
- `--performance-insights-retention-period`：数据库集群的数据的保留期。要开启数据库洞察的高级模式，保留期必须至少为 465 天。

以下示例在修改数据库集群时启用数据库洞察的高级模式。

对于 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster \
    --database-insights-mode advanced \
    --db-cluster-identifier sample-db-identifier \
    --enable-performance-insights \
```

```
    --performance-insights-retention-period 465
```

对于 Windows：

```
aws rds modify-db-cluster ^
    --database-insights-mode advanced ^
    --db-cluster-identifier sample-db-identifier ^
    --enable-performance-insights ^
    --performance-insights-retention-period 465
```

RDS API

要在修改数据库集群时开启 Database Insights 的高级模式，请为您的 ModifyDBCluster Amazon RDS API 操作指定以下参数。

- `DatabaseInsightsMode` 到 `advanced`
- `EnablePerformanceInsights` 到 `True`
- `PerformanceInsightsRetentionPeriod` 设置为至少 `465` 天

为 Aurora PostgreSQL 无限制数据库开启 Database Insights 的标准模式

要为 Aurora PostgreSQL 无限制数据库开启 Database Insights 的标准模式，请按照以下过程操作。

为 Aurora PostgreSQL 无限制数据库创建数据库集群时开启 Database Insights 的标准模式

为 Aurora PostgreSQL 无限制数据库创建数据库时启用 Database Insights 的标准模式。

Console

在控制台中，您可以在创建数据库集群时开启数据库洞察的标准模式。数据库洞察的设置适用于数据库集群中的所有数据库实例。

在使用控制台创建数据库集群时，开启 Database Insights 的标准模式

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 选择数据库。
3. 选择创建数据库。
4. 在数据库洞察部分，选择标准模式。然后，选择以下选项：

- 保留 – 保留性能详情数据的时间。要为 Aurora PostgreSQL 无限制数据库创建数据库集群，保留期必须至少为 31 天。
- AWS KMS key - 指定您的 KMS 密钥。性能详情使用您的 KMS 密钥来加密所有潜在的敏感数据。正在传输的数据和静态数据都会被加密。有关更多信息，请参阅 加密 Amazon Aurora 资源。

5. 选择创建数据库。

AWS CLI

要在创建数据库集群时打开数据库洞察的标准模式，请调用 create-db-cluster AWS CLI 命令并提供以下值：

- `--db-cluster-identifier` – 数据库集群的标识符。
- `--database-insights-mode standard`，开启数据库洞察的标准模式。
- `--engine` – 数据库集群必须使用 `aurora-postgresql` 数据库引擎。
- `--engine-version` – 数据库集群必须使用的数据库引擎版本之一：
  - `16.4-limitless`
  - `16.6-limitless`
- `--storage-type` – 数据库集群必须使用 `aurora-iopt1` 数据库集群存储配置。
- `--cluster-scalability-type` – 指定 Aurora 数据库集群的可扩展性模式。设置为 `limitless` 时，集群将作为 Aurora PostgreSQL Limitless Database 运行。设置为 `standard`（默认）时，集群使用正常的数据库实例创建。

> **Note**
>
> 创建数据库集群之后，您将无法修改此设置。

- `--master-username` – 数据库集群的主用户名称。
- `--master-user-password` – 主用户的密码。
- `--enable-performance-insights`，开启数据库洞察的性能详情。
- `--performance-insights-retention-period` – 数据库集群数据的保留期。要为 Aurora PostgreSQL 无限制数据库创建数据库集群，保留期必须至少为 31 天。
- `--monitoring-interval` – 收集数据库集群的增强监控指标的时间点之间的间隔，以秒为单位。该值不能为 `0`。

- `--monitoring-role-arn` – 允许 RDS 将增强监控指标发送到 Amazon CloudWatch Logs 的 IAM 角色的 Amazon 资源名称（ARN）。
- `--enable-cloudwatch-logs-exports` – 您必须将 `postgresql` 日志导出到 CloudWatch 日志。

以下示例在创建数据库集群时启用 Database Insights 的标准模式。

对于 Linux、macOS 或 Unix：

```
aws rds create-db-cluster \
--db-cluster-identifier my-limitless-cluster \
--database-insights-mode standard \
--engine aurora-postgresql \
--engine-version 16.6-limitless \
--storage-type aurora-iopt1 \
--cluster-scalability-type limitless \
--master-username myuser \
--master-user-password mypassword \
--enable-performance-insights \
--performance-insights-retention-period 31 \
--monitoring-interval 5 \
--monitoring-role-arn arn:aws:iam::123456789012:role/EMrole \
--enable-cloudwatch-logs-exports postgresql
```

对于 Windows：

```
aws rds create-db-cluster ^
--db-cluster-identifier my-limitless-cluster ^
--database-insights-mode standard ^
--engine aurora-postgresql ^
--engine-version 16.6-limitless ^
--storage-type aurora-iopt1 ^
--cluster-scalability-type limitless ^
--master-username myuser ^
--master-user-password mypassword ^
--enable-performance-insights ^
--performance-insights-retention-period 31 ^
--monitoring-interval 5 ^
--monitoring-role-arn arn:aws:iam::123456789012:role/EMrole ^
--enable-cloudwatch-logs-exports postgresql
```

RDS API

要在创建数据库集群时开启 Database Insights 的高级模式，请为 CreateDBCluster Amazon RDS API 操作指定以下参数。

- `DatabaseInsightsMode` 到 `standard`
- `Engine` 到 `aurora-postgresql`
- `EngineVersion` 设置为无限制数据库的可用引擎版本
- `StorageType` 到 `aurora-iopt1`
- `ClusterScalabilityType` 到 `limitless`
- `MasterUsername`
- `MasterUserPassword`
- `EnablePerformanceInsights` 到 `True`
- `PerformanceInsightsRetentionPeriod` 设置为至少 `31` 天
- `MonitoringInterval` 设置为非 `0` 的值
- `MonitoringRoleArn` 设置为 IAM 角色的 Amazon 资源名称（ARN），该角色应可以让 RDS 将增强监控指标发送到 Amazon CloudWatch Logs。

为 Aurora PostgreSQL 无限制数据库修改数据库集群时开启 Database Insights 的标准模式

为 Aurora PostgreSQL 无限制数据库修改数据库时开启 Database Insights。

> **Note**
>
> 要启用数据库洞察，数据库集群中的每个数据库实例都必须具有相同的性能详情和增强型监控设置。

Console

在控制台中，您可以在创建数据库集群时开启数据库洞察的标准模式。数据库洞察的设置适用于数据库集群中的所有数据库实例。

在使用控制台修改数据库集群时，开启 Database Insights 的标准模式

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。

2. 选择数据库。
3. 选择数据库集群，然后选择修改。
4. 在数据库洞察部分，选择标准模式。然后，选择以下选项：
   - 保留 – 保留性能详情数据的时间。要为 Aurora PostgreSQL 无限制数据库创建数据库集群，保留期必须至少为 31 天。
   - AWS KMS key - 指定您的 KMS 密钥。性能详情使用您的 KMS 密钥来加密所有潜在的敏感数据。正在传输的数据和静态数据都会被加密。有关更多信息，请参阅“加密 Amazon Aurora 资源”。
5. 选择继续。
6. 对于修改计划，选择立即应用。如果您选择在下一个计划维护时段内应用，则您的数据库将忽略此设置，并立即开启数据库洞察的标准模式。
7. 选择修改集群。

AWS CLI

要在修改数据库集群时开启数据库洞察的标准模式，请调用 modify-db-cluster AWS CLI 命令并提供以下值：

- `--db-cluster-identifier` – 数据库集群的标识符。
- `--database-insights-mode standard`，开启数据库洞察的标准模式。
- `--enable-performance-insights`，开启数据库洞察的性能详情。
- `--performance-insights-retention-period` – 数据库集群数据的保留期。要开启 Database Insights 的标准模式，保留期必须至少为 31 天。

以下示例在修改数据库集群时启用 Database Insights 的标准模式。

对于 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster \
    --database-insights-mode standard \
    --db-cluster-identifier sample-db-identifier \
    --enable-performance-insights \
    --performance-insights-retention-period 31
```

对于 Windows：

```
aws rds modify-db-cluster ^
    --database-insights-mode standard ^
    --db-cluster-identifier sample-db-identifier ^
    --enable-performance-insights ^
    --performance-insights-retention-period 31
```

RDS API

要在修改数据库集群时开启 Database Insights 的标准模式，请为您的 ModifyDBCluster Amazon RDS API 操作指定以下参数。

- `DatabaseInsightsMode` 到 `standard`
- `EnablePerformanceInsights` 到 `True`
- `PerformanceInsightsRetentionPeriod` 设置为至少 `31` 天

使用 Amazon CloudWatch Logs 监控 Aurora PostgreSQL Limitless Database

作为启用 Aurora PostgreSQL Limitless Database 的一部分，需要将 PostgreSQL 日志导出到 CloudWatch Logs。您可以在 CloudWatch Logs Insights 中访问和分析这些日志，类似于访问标准 Aurora PostgreSQL 数据库集群的 PostgreSQL 日志。有关更多信息，请参阅 使用 CloudWatch Logs Insights 分析 Aurora PostgreSQL 日志。

数据库集群的日志组名称与 Aurora PostgreSQL 中的名称相同：

```
/aws/rds/cluster/DB_cluster_ID/postgresql
```

数据库分片组的日志组名称采用以下格式：

```
/aws/rds/cluster/DB_cluster_ID/DB_shard_group_ID/postgresql
```

每个节点（路由器或分片）都有日志流。它们的名称具有以下形式：

```
[DistributedTransactionRouter|
DataAccessShard]/node_cluster_serial_ID-node_instance_serial_ID/n
```

例如：

- 路由器 – `DistributedTransactionRouter/6-6.2`
- 分片 – `DataAccessShard/22-22.0`

> **Note**
>
> 您无法直接在 RDS 控制台中查看数据库分片组的 PostgreSQL 日志文件 AWS CLI，也无法像数据库集群那样直接查看 RDS API。您必须使用 CloudWatch Logs Insights 进行查看。

使用增强监控指标监控 Aurora PostgreSQL Limitless Database

启用 Aurora PostgreSQL Limitless Database 需要增强监控。您可以使用该功能来实时监控 Limitless Database 数据库实例的操作系统。

Aurora 在 CloudWatch Logs 中发布增强监控指标。可用的一些关键指标包括数据库连接、存储使用情况和查询延迟。这些指标有助于确定性能瓶颈。

有关增强监控指标的更多信息，请参阅 Aurora 的操作系统指标。

使用性能详情监控 Aurora PostgreSQL Limitless Database

使用性能详情监控 Aurora PostgreSQL Limitless Database 集群。性能详情在 Aurora PostgreSQL Limitless Database 上的工作原理与在标准 Aurora 数据库集群上的工作原理类似。但是，您可以在分片组级别跟踪 Aurora PostgreSQL Limitless Database 的指标。

要跟踪的两个主要性能详情指标如下：

- 数据库负载 – 衡量数据库中的活动级别。性能详情的关键指标是 DBLoad，每秒收集一次。

  性能详情中 DBLoad 指标的单位是平均活动会话数（AAS）。为了获取平均活动会话数，性能详情会对同时运行查询的会话数进行采样。平均活动会话数等于会话总数除以特定时间段内的样本总数。有关 DBLoad 和 AAS 的更多信息，请参阅数据库负载。
- 最大 CPU – 数据库可用的最大计算能力。要查看活动会话是否超过最大 CPU，请查看它们与 Max vCPU 线的关系。Max vCPU 值由数据库实例的 vCPU（虚拟 CPU）内核数决定。有关 Max vCPU 的更多信息，请参阅最大 CPU。

此外，您还可以将 DBLoad 指标“分割”成维度，这些维度是该指标的子类别。最有用的维度如下：

- 热门实例 – 按降序显示您的实例（分片和路由器）的相对数据库负载。
- 等待事件 – 会导致 SQL 语句等待特定事件发生，然后才能继续运行。等待事件显示工作受阻的位置。
- 热门 SQL – 显示哪些查询造成的数据库负载最大。

有关性能详情维度的更多信息，请参阅尺寸。

下图显示了数据库分片组的热门实例维度。



主题

- 使用性能详情控制面板分析 Aurora PostgreSQL Limitless Database 的数据库负载

使用性能详情控制面板分析 Aurora PostgreSQL Limitless Database 的数据库负载

借助性能详情，您可以在分片组级别和实例级别跟踪 Aurora PostgreSQL Limitless Database 的指标。在分析 Aurora PostgreSQL Limitless Database 的数据库负载时，您可能需要将每个分片和路由器的数据库负载与最大 vCPU 进行比较。

> **Note**
>
> Aurora PostgreSQL Limitless Database 始终启用性能详情和增强监控。Limitless Database 的性能详情数据的最短保留期为 31 天（1 个月）。

绝对视图显示平均活动会话（AAS）数和预计 vCPU 数。相对视图显示 AAS 与预计 vCPU 的比率。

主题

- 使用性能详情控制面板分析 Aurora PostgreSQL Limitless Database 的相对数据库负载
- 使用性能详情控制面板按等待状态分析 Aurora PostgreSQL Limitless Database 的数据库负载
- 使用性能详情控制面板分析 Aurora PostgreSQL Limitless Database 的负载分配

使用性能详情控制面板分析 Aurora PostgreSQL Limitless Database 的相对数据库负载

您可能需要通过跟踪相对数据库负载来提高 Aurora PostgreSQL Limitless Database 的性能。要按实例分析 Aurora PostgreSQL Limitless Database 的相对数据库负载，请执行以下步骤。

使用控制台分析 Aurora PostgreSQL Limitless Database 的相对数据库负载

1. 通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择性能详情。
3. 选择 Aurora PostgreSQL Limitless Database。此时将显示该 Aurora PostgreSQL Limitless Database 的性能详情控制面板。
4. 在数据库负载（DB 负载）部分，为划分者选择实例。要查看 Aurora PostgreSQL Limitless Database 中所有实例的平均活动会话数（AAS）与 vCPU 内核数的比率，请为查看方式选择相对。

   平均活动会话图表显示了您的 Aurora PostgreSQL Limitless Database 中实例的数据库负载。



5. 要查看热门实例，请选择热门实例选项卡。

   在以下示例中，数据库负载最高的实例是 DTR-2-2。



6. （可选）要分析 Aurora PostgreSQL Limitless Database 中实例的数据库负载，请在实例列中选择实例名称。要查看 DTR-2-2 的数据库负载，请在实例列中选择 DTR-2-2。

> **Note**
>
> 您只能查看 Aurora PostgreSQL Limitless Database 中实例的性能详情指标。

使用性能详情控制面板按等待状态分析 Aurora PostgreSQL Limitless Database 的数据库负载

您可能需要通过跟踪等待事件来提高 Aurora PostgreSQL Limitless Database 的性能。要按等待事件分析 Aurora PostgreSQL Limitless Database 的数据库负载，请使用以下步骤。

使用控制台按等待状态分析 Aurora PostgreSQL Limitless Database 的数据库负载

1. 通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择性能详情。
3. 选择 Aurora PostgreSQL Limitless Database。此时将显示该 Aurora PostgreSQL Limitless Database 的性能详情控制面板。
4. 在数据库负载（DB 负载）部分，为划分者选择等待。要查看 AAS 和预计 vCPU 的数量，请为查看方式选择绝对。

   平均活动会话图表显示了您的 Aurora PostgreSQL Limitless Database 中实例的数据库负载。



5. 向下滚动至 Top SQL（热门 SQL）选项卡。

   在以下示例中，因等待而导致最高负载的 SQL 语句是 DELETE 语句。



6. 选择 SQL 语句，以将其扩展到其组件语句中。

   在以下示例中，SELECT 语句有 3 个组件语句。



使用性能详情控制面板分析 Aurora PostgreSQL Limitless Database 的负载分配

您可能需要在 Aurora PostgreSQL Limitless Database 上平衡实例的负载分配。要分析 Aurora PostgreSQL Limitless Database 上实例的负载分配，请使用以下步骤。

使用控制台分析 Aurora PostgreSQL Limitless Database 上实例的负载分配

1. 通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择性能详情。
3. 选择 Aurora PostgreSQL Limitless Database。此时将显示该 Aurora PostgreSQL Limitless Database 的性能详情控制面板。
4. 在数据库负载（DB 负载）部分，为划分者选择实例。要查看 Aurora PostgreSQL Limitless Database 中所有实例的 AAS 和预计 vCPU 的数量，请为查看方式选择绝对。

   平均活动会话图表显示了您的 Aurora PostgreSQL Limitless Database 中实例的数据库负载。



5. 要查看 Aurora PostgreSQL Limitless Database 中实例的负载分配图表，请选择负载分配选项卡。

   在以下示例中，数据库负载最高的实例是 DTR-2-2。


Dimensions
Load distribution
Load distribution of the instances on DB shard group from May 13, 2024, 00:00 UTC to May 13, 2024, 23:59 UTC
Other instances
779 AAS (5.72%)
DAS-6-6
220 AAS (1.62%)
DAS-7-7
244 AAS (1.79%)
DTR-4-4
1198 AAS (8.80%)
13621
Total AAS
DTR-2-2
5655 AAS (41.52%)
DAS-9-9
5525 AAS (40.56%)
DTR-2-2
DAS-9-9
DTR-4-4
DAS-7-7
DAS-6-6
Other instances

使用 Amazon GuardDuty RDS 保护功能监控 Aurora PostgreSQL Limitless Database

Amazon GuardDuty 是一项威胁检测服务，可帮助保护您的账户、容器、工作负载和 AWS 环境中的数据。GuardDuty 使用机器学习（ML）模型以及异常和威胁检测功能，持续监控不同的日志源和运行时活动，以识别环境中的潜在安全风险和恶意活动并确定其优先级。

GuardDuty RDS 保护会分析和剖析登录事件，找出对 Amazon Aurora 数据库的潜在访问威胁。当您开启 RDS 保护功能时，GuardDuty 会监控来自 Aurora 数据库的 RDS 登录事件。RDS 保护会监控这些事件并对其进行剖析，以了解潜在的内部威胁或外部行为者。

有关 Aurora 中的 GuardDuty RDS 保护功能的更多信息，请参阅使用适用于 Amazon Aurora 的 Amazon GuardDuty RDS 保护功能监控威胁。

有关启用 GuardDuty RDS 保护功能的更多信息，请参阅《Amazon GuardDuty 用户指南》中的 GuardDuty RDS 保护。

Aurora PostgreSQL Limitless Database 的函数和视图

Aurora PostgreSQL Limitless Database 添加了函数和视图。它们基于相应的 Aurora PostgreSQL 函数和视图。

> **Note**
>
> 如果有正在进行的事务，某些统计数据可能会返回不一致的结果。

主题

- Aurora PostgreSQL Limitless Database 函数
- Aurora PostgreSQL Limitless Database 视图

Aurora PostgreSQL Limitless Database 函数

下表列出了 Aurora PostgreSQL Limitless Database 的新函数。

> **Note**
>
> 此表中列出的函数位于 `rds_aurora` 架构中。使用 Limitless Database 函数时，请确保包含完全限定的对象名称：`rds_aurora.object_name`。

Aurora PostgreSQL Limitless Database 函数	对应的 Aurora PostgreSQL 函数
limitless_backend_dsid	pg_backend_pid
limitless_cancel_session	pg_cancel_backend
limitless_stat_clear_snapshot	pg_stat_clear_snapshot
limitless_stat_database_size	pg_database_size
limitless_stat_get_snapshot_timestamp	pg_stat_get_snapshot_timestamp
limitless_stat_prepared_xacts	pg_prepared_xacts

Aurora PostgreSQL Limitless Database 函数	对应的 Aurora PostgreSQL 函数
limitless_stat_relation_sizes	pg_indexes_size、pg_relation_size、pg_table_size、pg_total_relation_size
limitless_stat_reset	pg_stat_reset
limitless_stat_statements_reset	pg_stat_statements_reset
limitless_stat_system_waits	aurora_stat_system_waits
limitless_terminate_session	pg_terminate_backend
limitless_wait_report	aurora_wait_report

以下示例提供了有关 Aurora PostgreSQL Limitless Database 函数的详细信息。有关 PostgreSQL 函数的更多信息，请参阅 PostgreSQL 文档中的 Functions and operators。

limitless_backend_dsid

`limitless_backend_dsid` 函数返回当前会话的分布式会话 ID。分布式会话在数据库分片组中的路由器上运行，涉及数据库分片组中一个或多个分片上的后端进程。

以下示例演示了如何使用 `limitless_backend_dsid` 函数。

```
SELECT rds_aurora.limitless_backend_dsid();

limitless_backend_dsid
------------------------
8CACD7B04D0FC2A5
(1 row)
```

limitless_cancel_session

`limitless_cancel_session` 函数的工作原理与 `pg_cancel_backend` 类似，但它尝试通过发送 `SIGINT`（中断信号）来取消与提供的分布式会话 ID 相关的所有后端进程。

以下是输入参数：

- `distributed_session_id`（文本）– 要取消的分布式会话的 ID。

以下是输出参数：

- subcluster_id（文本）– 此进程所属的子集群 ID。
- pid（文本）– 后端进程 ID。
- success（布尔值）– 取消是否成功。

以下示例演示了如何使用 limitless_cancel_session 函数。

```
SELECT * FROM rds_aurora.limitless_cancel_session('940CD5C81E3C796B');

 subcluster_id |  pid  | success
---------------+-------+---------
             1 | 26920 | t
(1 row)
```

limitless_stat_clear_snapshot

limitless_stat_clear_snapshot 函数丢弃所有节点上的当前统计快照或缓存信息。

以下示例演示了如何使用 limitless_stat_clear_snapshot 函数。

```
SELECT rds_aurora.limitless_stat_clear_snapshot();
```

limitless_stat_database_size

limitless_stat_database_size 函数返回数据库分片组中数据库的大小。

以下是输入参数：

- dbname（名称）– 要获取其大小的数据库。

以下是输出参数：

- subcluster_id（文本）– 此进程所属的子集群 ID。
- subcluster_type（文本）– 此进程所属的子集群的类型：router 或 shard。
- db_size – 此子集群中数据库的大小（以字节为单位）。

以下示例演示了如何使用 limitless_stat_database_size 函数。

```
SELECT * FROM rds_aurora.limitless_stat_database_size('postgres_limitless');

 subcluster_id | subcluster_type | db_size
---------------+-----------------+----------
```

```
             1 | router          |  8895919
             2 | router          |  8904111
             3 | shard           | 21929391
             4 | shard           | 21913007
             5 | shard           | 21831087
(5 rows)
```

limitless_stat_get_snapshot_timestamp

`limitless_stat_get_snapshot_timestamp` 函数返回当前统计快照的时间戳，如果没有拍摄统计快照，则返回 NULL。如果 `stats_fetch_consistency` 设置为 `snapshot`，则会在事务中首次访问累积统计数据时拍摄快照。返回所有节点快照时间戳的合并视图。`subcluster_id` 和 `subcluster_type` 列显示了数据来自哪个节点。

以下示例演示了如何使用 `limitless_stat_get_snapshot_timestamp` 函数。

```
SELECT * FROM rds_aurora.limitless_stat_get_snapshot_timestamp();

 subcluster_id | subcluster_type | snapshot_timestamp
---------------+-----------------+--------------------
             1 | router          |
             2 | router          |
             3 | shard           |
             4 | shard           |
             5 | shard           |
(5 rows)
```

limitless_stat_prepared_xacts

`limitless_stat_prepared_xacts` 函数返回有关当前准备进行两阶段提交的所有节点上的事务信息。有关更多信息，请参阅 PostgreSQL 文档中的 pg_prepared_xacts。

以下示例演示了如何使用 `limitless_stat_prepared_xacts` 函数。

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_stat_prepared_xacts;

 subcluster_id | subcluster_type | transaction_id |             gid              |
         prepared            |  owner_id  |    database_id
---------------+-----------------+----------------+------------------------------
+-------------------------------+------------+--------------------
 8             | shard           |        5815978 | 7_4599899_postgres_limitless |
 2024-09-03 15:51:17.659603+00 | auroraperf | postgres_limitless
```

```
 12            | shard           |        4599138 | 7_4599899_postgres_limitless |
 2024-09-03 15:51:17.659637+00 | auroraperf | postgres_limitless
(2 rows)
```

limitless_stat_relation_sizes

`limitless_stat_relation_sizes` 函数返回数据库分片组中表的不同大小。

以下是输入参数：

- `relnspname`（名称）– 包含表的架构名称。
- `relname`（名称）– 表的名称。

以下是输出参数：

- `subcluster_id`（文本）– 此进程所属的子集群 ID。
- `subcluster_type`（文本）– 此进程所属的子集群的类型：`router` 或 `shard`。
- `main_size` – 此节点中主数据分叉的大小（以字节为单位）。
- `fsm_size` – 此节点中表的可用空间映射的大小（以字节为单位）。
- `vm_size` – 此节点中表的可见性映射的大小（以字节为单位）。
- `init_size` – 此节点中表的初始化大小（以字节为单位）。
- `toast_size` – 与该分叉中的表关联的 toast 表的大小（以字节为单位）。
- `index_size` – 此节点中表的所有索引的大小（以字节为单位）。
- `total_size` – 此节点中所有表段的大小（以字节为单位）。

以下示例演示如何使用 `limitless_stat_relation_sizes` 函数（省略了一些列）。

```
SELECT * FROM rds_aurora.limitless_stat_relation_sizes('public','customers');

 subcluster_id | subcluster_type | main_size | fsm_size | vm_size | toast_size |
 table_size | total_size
---------------+-----------------+-----------+----------+---------+------------
+------------+------------
             1 | router          |         0 |        0 |       0 |          0 |
   0 |          0
             2 | router          |         0 |        0 |       0 |          0 |
   0 |          0
             3 | shard           |   4169728 |  4177920 | 1392640 |    1392640 |
 11132928 |   11132928
```

```
             4 | shard           |   4169728 |  4177920 | 1392640 |    1392640 |
 11132928 |   11132928
             5 | shard           |   3981312 |  4227072 | 1409024 |    1409024 |
 11026432 |   11026432
(5 rows)
```

limitless_stat_reset

limitless_stat_reset 函数将当前数据库的所有统计计数器重置为零（0）。如果启用 track_functions，则 limitless_stat_database 中的 stats_reset 列将显示上次重置数据库统计信息的时间。默认情况下，limitless_stat_reset 只能由超级用户运行。其他用户可以使用 EXECUTE 权限获得权限。

以下示例演示了如何使用 limitless_stat_reset 函数。

```
SELECT tup_inserted, tup_deleted FROM pg_stat_database
WHERE datname = 'postgres_limitless';

 tup_inserted | tup_deleted
--------------+-------------
          896 |           0
(1 row)

SELECT rds_aurora.limitless_stat_reset();

limitless_stat_reset
---------------------
(1 row)

SELECT tup_inserted, tup_deleted FROM pg_stat_database
WHERE datname = 'postgres_limitless';

tup_inserted | tup_deleted
-------------+-------------
           0 |           0
(1 row)
```

limitless_stat_statements_reset

limitless_stat_statements_reset 函数丢弃 limitless_stat_statements 迄今为止收集到的与指定 username、dbname、distributed_query_id 和 queryid 参数相对应的统计数据。如果未指定任何参数，则将对每个参数使用默认值 "" 或 0（无效），并重置与其他参数匹

配的统计数据。如果未指定任何参数，或者所有指定的参数均为 "" 或 0（无效），则该函数将丢弃所有统计数据。如果 limitless_stat_statements 视图中的所有统计数据都被丢弃，则该函数还会重置 limitless_stat_statements_info 视图中的统计数据。

以下是输入参数：

- username（名称）– 查询语句的用户。
- dbname（名称）– 运行查询的数据库。
- distributed_query_id（bigint）– 来自协调器节点的父查询的查询 ID。如果是父查询，则此列为 NULL。协调器节点将分布式查询 ID 向下推送到参与者节点。因此，对于参与者节点，分布式查询 ID 和查询 ID 的值不同。
- queryid（bigint）– 语句的查询 ID。

以下示例说明如何使用 limitless_stat_statements_reset 函数重置 limitless_stat_statements 收集的所有统计数据。

```
SELECT rds_aurora.limitless_stat_statements_reset();
```

limitless_stat_system_waits

limitless_stat_system_waits 函数返回来自 aurora_stat_system_waits 的等待事件数据的合并视图，该视图报告实例中来自所有节点的系统范围的等待活动。subcluster_id 和 subcluster_type 列显示了数据来自哪个节点。

以下示例演示了如何使用 limitless_stat_system_waits 函数。

```
postgres_limitless=> SELECT *
FROM rds_aurora.limitless_stat_system_waits() lssw, pg_catalog.aurora_stat_wait_event()
 aswe
WHERE lssw.event_id=aswe.event_id and aswe.event_name='LimitlessTaskScheduler';

 subcluster_id | subcluster_type | type_id | event_id  | waits  |  wait_time   |
 event_name
---------------+-----------------+---------+-----------+--------+--------------
+------------------------
             1 | router          |      12 | 201326607 | 677068 | 616942216307 |
 LimitlessTaskScheduler
             2 | router          |      12 | 201326607 | 678586 | 616939897111 |
 LimitlessTaskScheduler
             3 | shard           |      12 | 201326607 | 756640 | 616965545172 |
 LimitlessTaskScheduler
```

```
             4 | shard           |      12 | 201326607 | 755184 | 616958057620 |
 LimitlessTaskScheduler
             5 | shard           |      12 | 201326607 | 757522 | 616963183539 |
 LimitlessTaskScheduler
(5 rows)
```

limitless_terminate_session

limitless_terminate_session 函数的工作原理与 pg_terminate_backend 类似，但它尝试通过发送 SIGTERM（结束信号）来结束与提供的分布式会话 ID 相关的所有后端进程。

以下是输入参数：

- distributed_session_id（文本）– 要结束的分布式会话的 ID。

以下是输出参数：

- subcluster_id（文本）– 此进程所属的子集群 ID。
- pid（文本）– 后端进程 ID。
- success（布尔值）– 进程是否成功结束。

以下示例演示了如何使用 limitless_terminate_session 函数。

```
SELECT * FROM rds_aurora.limitless_terminate_session('940CD5C81E3C796B');

 subcluster_id |  pid  | success
---------------+-------+---------
             1 | 26920 | t
(1 row)
```

limitless_wait_report

limitless_wait_report 函数返回一段时间内所有节点的等待事件活动。subcluster_id 和 subcluster_type 列显示了数据来自哪个节点。

以下是输出参数：

- subcluster_id（文本）– 此进程所属的子集群 ID。
- subcluster_type（文本）– 此进程所属的子集群的类型：router 或 shard。

其余各列与 aurora_wait_report 中的相同。

以下示例演示了如何使用 limitless_wait_report 函数。

```
postgres_limitless=> select * from rds_aurora.limitless_wait_report();

 subcluster_id | subcluster_type | type_name | event_name | waits | wait_time |
 ms_per_wait | waits_per_xact | ms_per_xact
---------------+-----------------+-----------+------------+-------+-----------
+-------------+--------------- +-------------
             1 | router          | Client    | ClientRead |    57 | 741550.14 |
 13009.652 |           0.19 |    2505.237
             5 | shard           | Client    | ClientRead |    54 | 738897.68 |
 13683.290 |           0.18 |    2496.276
             4 | shard           | Client    | ClientRead |    54 | 738859.53 |
 13682.584 |           0.18 |    2496.147
             2 | router          | Client    | ClientRead |    53 | 719223.64 |
 13570.257 |           0.18 |    2429.810
             3 | shard           | Client    | ClientRead |    54 | 461720.40 |
 8550.378 |           0.18 |    1559.86
```

Aurora PostgreSQL Limitless Database 视图

下表列出了 Aurora PostgreSQL Limitless Database 的新视图。

> **Note**
>
> 此表中列出的视图位于 `rds_aurora` 架构中。使用 Limitless Database 视图时，请确保包含完全限定的对象名称：`rds_aurora.object_name`。

Aurora PostgreSQL Limitless Database 视图	对应的 Aurora PostgreSQL 视图
limitless_database	pg_database
limitless_locks	pg_locks
limitless_stat_activity	pg_stat_activity
limitless_stat_all_indexes	pg_stat_all_indexes
limitless_stat_all_tables	pg_stat_all_tables
limitless_stat_database	pg_stat_database
limitless_stat_progress_vacuum	pg_stat_progress_vacuum
limitless_stat_statements	pg_stat_statements
limitless_stat_subclusters	无
limitless_stat_statements_info	pg_stat_statements_info
limitless_statio_all_indexes	pg_statio_all_indexes
limitless_statio_all_tables	pg_statio_all_tables
limitless_tables	pg_tables
limitless_table_collocations	无
limitless_table_collocation_distributions	无

以下示例提供了有关 Aurora PostgreSQL Limitless Database 视图的详细信息。有关 PostgreSQL 视图的更多信息，请参阅 PostgreSQL 文档中的 Viewing statistics。

> **Note**
>
> 如果有正在进行的事务，某些统计数据视图可能会返回不一致的结果。

limitless_database

此视图包含有关数据库分片组中可用数据库的信息。例如：

```
postgres_limitless=> SELECT subcluster_id, subcluster_type, oid, datname, datacl
 FROM rds_aurora.limitless_database;

 subcluster_id | subcluster_type |  oid  |      datname       |
                                 datacl

---------------+-----------------+-------+--------------------
+-------------------------------------------------------------------------------------------
 2             | router          |     4 | template0          | {=c/
rdsadmin,rdsadmin=CTc/rdsadmin}
 2             | router          |     5 | postgres           |
 2             | router          | 16384 | rdsadmin           |
 {rdsadmin=CTc/rdsadmin,rds_aurora_limitless_metadata_admin=c/
rdsadmin,rds_aurora_limitless_heat_mgmt_admin=c/rdsadmin}
 2             | router          | 16477 | postgres_limitless |
 2             | router          |     1 | template1          | {=c/
rdsadmin,rdsadmin=CTc/rdsadmin}
 6             | shard           |     4 | template0          | {=c/
rdsadmin,rdsadmin=CTc/rdsadmin}
```

以下是输出参数：

- `subcluster_id`（文本）– 子集群（节点）的 ID
- `subcluster_type`（文本）– 子集群（节点）、路由器或分片的类型

其余各列与 `pg_database` 中的相同。

limitless_locks

此视图包含每个节点上每个进程的一行。它提供对数据库服务器中活动进程持有的锁的相关信息的访问。

Example 使用两个事务创建锁的示例

在此示例中，我们在两台路由器上同时运行两个事务。

```
# Transaction 1 (run on router 1)
BEGIN;
SET search_path = public;
SELECT * FROM customers;
INSERT INTO customers VALUES (400,'foo','bar');

# Transaction 2 (run on router 2)
BEGIN;
SET search_path = public;
ALTER TABLE customers ADD COLUMN phone VARCHAR;
```

第一个事务已开始运行。后续事务必须等到第一个事务完成。因此，第二个事务被锁屏蔽。为了检查其根本原因，我们运行一条命令，将 `limitless_locks` 与 `limitless_stat_activity` 连接起来。

```
# Run on router 2
SELECT distributed_session_id, state, usename, query, query_start
FROM rds_aurora.limitless_stat_activity
WHERE distributed_session_id in (
SELECT distributed_session_id
FROM rds_aurora.limitless_locks
WHERE relname = 'customers'
);

 distributed_session_id | state               | usename                 | query
                                  | query_start
------------------------+---------------------+-------------------------
+---------------------------------- -------------+------------------------------
 47BDE66E9A5E8477       | idle in transaction | limitless_metadata_admin | INSERT INTO
 customers VALUES (400,'foo','bar'); | 2023-04-13 17:44:45.152244+00
 2AD7F370202D0FA9       | active              | limitless_metadata_admin | ALTER TABLE
 customers ADD COLUMN phone VARCHAR; | 2023-04-13 17:44:55.113388+00
 47BDE66E9A5E8477       |                     | limitless_auth_admin     |
 <insufficient privilege>                        |
 2AD7F370202D0FA9       |                     | limitless_auth_admin     |
 <insufficient privilege>                        |
 47BDE66E9A5E8477       |                     | limitless_auth_admin     |
 <insufficient privilege>                        |
```

```
 2AD7F370202D0FA9       |                     | limitless_auth_admin     |
 <insufficient privilege>                        |
(6 rows)
```

Example 显式创建锁示例

在此示例中，我们显式创建了一个锁，然后使用 limitless_locks 视图来查看锁（省略了某些列）。

```
BEGIN;
SET search_path = public;
LOCK TABLE customers IN ACCESS SHARE MODE;
SELECT * FROM rds_aurora.limitless_locks WHERE relname = 'customers';

 subcluster_id | subcluster_type | distributed_session_id | locktype |      datname
   | relnspname |  relname  | virtualtransaction |  pid  |      mode
---------------+-----------------+------------------------+----------
+--------------------+------------+ ----------+--------------------+-------
+-----------------
             1 | router          | 7207702F862FC937       | relation |
 postgres_limitless | public     | customers | 28/600787          | 59564 |
 AccessShareLock
             2 | router          | 7207702F862FC937       | relation |
 postgres_limitless | public     | customers | 28/600405          | 67130 |
 AccessShareLock
             3 | shard           | 7207702F862FC937       | relation |
 postgres_limitless | public     | customers | 15/473401          | 27735 |
 AccessShareLock
             4 | shard           | 7207702F862FC937       | relation |
 postgres_limitless | public     | customers | 13/473524          | 27734 |
 AccessShareLock
             5 | shard           | 7207702F862FC937       | relation |
 postgres_limitless | public     | customers | 13/472935          | 27737 |
 AccessShareLock
             6 | shard           | 7207702F862FC937       | relation |
 postgres_limitless | public     | customers | 13/473015          | 48660 |
 AccessShareLock
(6 rows)
```

limitless_stat_activity

此视图包含每个节点上每个进程的一行。它可用于跟踪整个系统的运行状况，并对耗时较长的流程进行分类。例如：

```
postgres=# SELECT
    subcluster_id,
    subcluster_type,
    distributed_session_id,
    distributed_session_state,
    datname,
    distributed_query_id,
    is_sso_query
FROM
    rds_aurora.limitless_stat_activity
WHERE
    distributed_session_id in ('D2470C97E3D07E06', '5A3CD7B8E5FD13FF')
    order by  distributed_session_id;

 subcluster_id | subcluster_type | distributed_session_id | distributed_session_state |
      datname       | distributed_query_id | is_sso_query
---------------+-----------------+------------------------+---------------------------
+--------------------+----------------------+--------------
 2             | router          | 5A3CD7B8E5FD13FF       | coordinator               |
 postgres_limitless |                      | f
 3             | shard           | 5A3CD7B8E5FD13FF       | participant               |
 postgres_limitless |  6808291725541680947 |
 4             | shard           | 5A3CD7B8E5FD13FF       | participant               |
 postgres_limitless |  6808291725541680947 |
 2             | router          | D2470C97E3D07E06       | coordinator               |
 postgres_limitless |                      | t
 3             | shard           | D2470C97E3D07E06       | participant               |
 postgres_limitless |  4058400544464210222 |
(5 rows)
```

以下是输出参数：

- subcluster_id（文本）– 此进程所属的子集群 ID。
- subcluster_type（文本）– 此进程所属的子集群的类型：router 或 shard。
- distributed_session_id（文本）– 此进程所属的分布式会话的 ID。
- distributed_session_state（文本）– 这是协调器进程、参与者进程还是独立/非分布式进程（显示为 NULL）。
- datname（文本）– 此进程所连接的数据库。
- distributed_query_id（bigint）– 来自协调器节点的父查询的查询 ID。如果是父查询，则此列为 NULL。协调器节点将分布式查询 ID 向下推送到参与者节点。因此，对于参与者节点，分布式查询 ID 和查询 ID 的值不同。

- is_sso_query（文本）：这可以让我们知道查询是否经过单分片优化。

其余各列与 pg_stat_activity 中的相同。

limitless_stat_all_indexes

此视图包含数据库分片组中索引的使用情况统计数据。例如：

```
postgres_limitless=> SELECT schemaname, relname, indexrelname, idx_scan
  FROM rds_aurora.limitless_stat_all_indexes
  WHERE relname LIKE 'orders_ts%' ORDER BY indexrelname LIMIT 10;

 schemaname |    relname     |    indexrelname     | idx_scan
------------+----------------+---------------------+----------
 ec_sample  | orders_ts00001 | orders_ts00001_pkey |   196801
 ec_sample  | orders_ts00002 | orders_ts00002_pkey |   196703
 ec_sample  | orders_ts00003 | orders_ts00003_pkey |   196376
 ec_sample  | orders_ts00004 | orders_ts00004_pkey |   197966
 ec_sample  | orders_ts00005 | orders_ts00005_pkey |   195301
 ec_sample  | orders_ts00006 | orders_ts00006_pkey |   195673
 ec_sample  | orders_ts00007 | orders_ts00007_pkey |   194475
 ec_sample  | orders_ts00008 | orders_ts00008_pkey |   191694
 ec_sample  | orders_ts00009 | orders_ts00009_pkey |   193744
 ec_sample  | orders_ts00010 | orders_ts00010_pkey |   195421
(10 rows)
```

limitless_stat_all_tables

此视图包含数据库分片组中当前数据库中所有表的统计数据。这在跟踪清理操作和数据操纵语言（DML）操作时很有用。例如：

```
postgres_limitless=> SELECT subcluster_id, subcluster_type, relname,
 n_ins_since_vacuum, n_tup_ins, last_vacuum
  FROM rds_aurora.limitless_stat_all_tables
  WHERE relname LIKE 'orders_ts%' ORDER BY relname LIMIT 10;

 subcluster_id | subcluster_type |    relname     | n_ins_since_vacuum | n_tup_ins |
 last_vacuum
---------------+-----------------+----------------+--------------------+-----------
+-------------
 5             | shard           | orders_ts00001 |              34779 |    196083 |
 5             | shard           | orders_ts00002 |              34632 |    194721 |
 5             | shard           | orders_ts00003 |              34950 |    195965 |
 5             | shard           | orders_ts00004 |              34745 |    197283 |
```

```
 5             | shard           | orders_ts00005 |              34879 |    195754 |
 5             | shard           | orders_ts00006 |              34340 |    194605 |
 5             | shard           | orders_ts00007 |              33779 |    192203 |
 5             | shard           | orders_ts00008 |              33826 |    191293 |
 5             | shard           | orders_ts00009 |              34660 |    194117 |
 5             | shard           | orders_ts00010 |              34569 |    195560 |
(10 rows)
```

以下是输出参数：

- subcluster_id（文本）– 此进程所属的子集群 ID。
- subcluster_type（文本）– 此进程所属的子集群的类型：router 或 shard。
- relname（名称）– 表的名称。

其余各列与 pg_stat_all_tables 中的相同。

limitless_stat_database

此视图包含数据库分片组中所有数据库的统计数据。每个节点的每个数据库返回一行。例如：

```
postgres_limitless=> SELECT
    subcluster_id,
    subcluster_type,
    datname,
    blks_read,
    blks_hit
FROM
    rds_aurora.limitless_stat_database
WHERE
    datname='postgres_limitless';
 subcluster_id | subcluster_type |      datname       | blks_read | blks_hit
---------------+-----------------+--------------------+-----------+----------
             1 | router          | postgres_limitless |       484 | 34371314
             2 | router          | postgres_limitless |       673 | 33859317
             3 | shard           | postgres_limitless |      1299 | 17749550
             4 | shard           | postgres_limitless |      1094 | 17492849
             5 | shard           | postgres_limitless |      1036 | 17485098
             6 | shard           | postgres_limitless |      1040 | 17437257
(6 rows)
```

以下是输出参数：

- subcluster_id（文本）– 此进程所属的子集群 ID。

- subcluster_type（文本）– 此进程所属的子集群的类型：router 或 shard。
- datname（文本）– 数据库的名称。

其余各列与 pg_stat_database 中的相同。

limitless_stat_progress_vacuum

此视图包含有关正在进行的清理操作的信息。例如：

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_stat_progress_vacuum;

-[ RECORD 1 ]----------+------------------
subcluster_id          | 3
subcluster_type        | shard
distributed_session_id | A56D96E2A5C9F426
pid                    | 5270
datname                | postgres
nspname                | public
relname                | customer_ts2
phase                  | vacuuming heap
heap_blks_total        | 130500
heap_blks_scanned      | 100036
heap_blks_vacuumed     | 0
index_vacuum_count     | 0
max_dead_tuples        | 11184810
num_dead_tuples        | 0

-[ RECORD 2 ]----------+------------------
subcluster_id          | 3
subcluster_type        | shard
distributed_session_id | 56DF26A89EC23AB5
pid                    | 6854
datname                | postgres
nspname                | public
relname                | sales_ts1
phase                  | vacuuming heap
heap_blks_total        | 43058
heap_blks_scanned      | 24868
heap_blks_vacuumed     | 0
index_vacuum_count     | 0
max_dead_tuples        | 8569523
num_dead_tuples        | 0
```

以下是输出参数：

- subcluster_id（文本）– 此进程所属的子集群 ID。
- subcluster_type（文本）– 此进程所属的子集群的类型：router 或 shard。
- distributed_session_id（文本）– 启动清理操作的会话的标识符。
- datname（名称）– 正在进行清理的数据库。
- nspname（名称）– 正在清理的表的架构名称。如果被清理的表与用户连接的表不在同一个数据库中，则为 null。
- relname（名称）– 正在清理的表的名称。如果被清理的表与用户连接的表不在同一个数据库中，则为 null。

其余各列与 pg_stat_progress_vacuum 中的相同。

limitless_stat_statements

此视图提供了一种跟踪在所有节点上运行的所有 SQL 语句的规划和运行统计数据的方法。

> **Note**
>
> 必须安装 pg_stat_statements 扩展程序才能使用 limitless_stat_statements 视图。
>
> ```
> -- CREATE EXTENSION must be run by a superuser
> CREATE EXTENSION pg_stat_statements;
>
> -- Verify that the extension is created on all nodes in the DB shard group
> SELECT distinct node_id
>     FROM rds_aurora.limitless_stat_statements
>     LIMIT 10;
> ```

以下示例说明了 limitless_stat_statements 视图的用法。

```
postgres_limitless=> SELECT
 subcluster_id,
 subcluster_type,
 distributedqueryid,
 username,
 dbname,
 sso_calls
FROM
 rds_aurora.limitless_stat_statements;
```

```
 subcluster_id | subcluster_type |  distributedqueryid  |              username
       |       dbname       | sso_calls
---------------+-----------------+----------------------
+-------------------------------------+--------------------+-----------
 2             | router          |                      | postgres
       | postgres_limitless |         0
 2             | router          |                      | postgres
       | postgres_limitless |         0
 2             | router          |                      | postgres
       | postgres_limitless |         0
 2             | router          |                      | postgres
       | postgres_limitless |         0
 2             | router          |                      | postgres
       | postgres_limitless |         0
 2             | router          |                      | postgres
       | postgres_limitless |         1
 3             | shard           | -7975178695405682176 | postgres
       | postgres_limitless |
[...]
```

以下是输出参数：

- subcluster_id（文本）– 此进程所属的子集群 ID。
- subcluster_type（文本）– 此进程所属的子集群的类型：router 或 shard。
- distributedqueryid（bigint）– 来自协调器节点的父查询的查询 ID。如果是父查询，则此列为 NULL。协调器节点将分布式查询 ID 向下推送到参与者节点。因此，对于参与者节点，分布式查询 ID 和查询 ID 的值不同。
- username（名称）– 查询语句的用户。
- dbname（名称）– 运行查询的数据库。
- sso_calls（名称）：已对语句执行单分片优化的次数。

其余列与 pg_stat_statements 中的列相同。

limitless_stat_statements_info

此视图包含 limitless_stat_statements 视图的统计数据。每行都包含来自每个节点的 pg_stat_statements_info 视图的数据。subcluster_id 列标识每个节点。

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_stat_statements_info;

 subcluster_id | subcluster_type | dealloc |          stats_reset
```

```
---------------+-----------------+---------+-------------------------------
             1 | router          |       0 | 2023-06-30 21:22:09.524781+00
             2 | router          |       0 | 2023-06-30 21:21:40.834111+00
             3 | shard           |       0 | 2023-06-30 21:22:10.709942+00
             4 | shard           |       0 | 2023-06-30 21:22:10.740179+00
             5 | shard           |       0 | 2023-06-30 21:22:10.774282+00
             6 | shard           |       0 | 2023-06-30 21:22:10.808267+00
(6 rows)
```

以下是输出参数：

- subcluster_id（文本）– 此进程所属的子集群 ID。

其余列与 pg_stat_statements_info 中的列相同。

limitless_stat_subclusters

此视图包含路由器与其他节点之间的网络统计数据。它包含每对路由器和其他节点的一行，例如：

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_stat_subclusters;

 orig_subcluster | orig_instance_az | dest_subcluster | dest_instance_az |
 latency_us |       latest_collection       | failed_requests | received_bytes |
 sent_bytes | same_az_requests | cross_az_requests |     stat_reset_timestamp
-----------------+------------------+-----------------+------------------
+------------+--------------------------------+-----------------+---------------
+------------+------------------+-------------------+------------------------------
 3               | us-west-2b       | 2               | us-west-2a       |
 847 | 2024-10-07 17:25:39.518617+00 |               0 |       35668633 |   92090171
 |               0 |           302787 | 2024-10-05 12:39:55.239675+00
 3               | us-west-2b       | 4               | us-west-2b       |
 419 | 2024-10-07 17:25:39.546376+00 |               0 |      101190464 |  248795719
 |           883478 |                0 | 2024-10-05 12:39:55.231218+00
 3               | us-west-2b       | 5               | us-west-2c       |
    1396 | 2024-10-07 17:25:39.52122+00  |               0 |       72864849 |
 172086292 |                0 |           557726 | 2024-10-05 12:39:55.196412+00
 3               | us-west-2b       | 6               | us-west-2c       |
 729 | 2024-10-07 17:25:39.54828+00  |               0 |       35668584 |   92090171
 |               0 |           302787 | 2024-10-05 12:39:55.247334+00
 3               | us-west-2b       | 7               | us-west-2a       |
    1702 | 2024-10-07 17:25:39.545307+00 |               0 |       71699576 |
 171634844 |                0 |           556278 | 2024-10-05 12:39:52.715168+00
 2               | us-west-2a       | 3               | us-west-2b       |
 868 | 2024-10-07 17:25:40.293927+00 |               0 |       35659611 |   92011872
 |               0 |           302817 | 2024-10-05 12:39:54.420758+00
```

```
2               | us-west-2a       | 4               | us-west-2b       |
786 | 2024-10-07 17:25:40.296863+00 |               0 |      102437253 |  251838024
|                0 |            895060 | 2024-10-05 12:39:54.404081+00
2               | us-west-2a       | 5               | us-west-2c       |
   1232 | 2024-10-07 17:25:40.292021+00 |               0 |       71990027 |
168828110 |                0 |            545453 | 2024-10-05 12:39:36.769549+00
```

以下是输出参数：

- orig_subcluster（文本）– 发起通信的路由器 ID
- orig_subcluster_az（文本）– 发起方路由器的可用区（AZ）
- dest_subcluster（文本）– 目标节点的 ID
- dest_subcluster_az（文本）– 目标节点上次收集的可用区
- latency_us（bigint）– 上次收集的节点间网络延迟，以微秒为单位。如果无法访问该节点，则该值为 0。
- latest_collection（时间戳）– 最新可用区集合的时间戳和目标节点的延迟
- failed_requests（bigint）– 失败的内部请求的累积计数
- received_bytes（bigint）– 从该节点接收到的估计累积字节数
- sent_bytes（bigint）– 发送到该节点的估计累积字节数
- same_az_requests（bigint）– 当该节点与发起方路由器位于同一可用区时，向该节点发出的内部数据库请求的累积数量
- cross_az_requests（bigint）– 当该节点与发起方路由器位于不同可用区时，向该节点发出的内部数据库请求的累积数量
- stat_reset_timestamp（时间戳）-上次重置此视图的累积统计数据的时间戳

limitless_statio_all_indexes

此视图包含数据库分片组中所有索引的输入/输出（I/O）统计数据。例如：

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_statio_all_indexes WHERE
 relname like'customers_ts%';

 subcluster_id | subcluster_type | schemaname |      relname      |
 indexrelname             | idx_blks_read | idx_blks_hit
---------------+-----------------+------------+-------------------
+-------------------------------------+ --------------+--------------
             3 | shard           | public     | customers_ts00002 |
 customers_ts00002_customer_name_idx |             1 |            0
```

```
             3 | shard           | public     | customers_ts00001 |
 customers_ts00001_customer_name_idx |             1 |            0
             4 | shard           | public     | customers_ts00003 |
 customers_ts00003_customer_name_idx |             1 |            0
             4 | shard           | public     | customers_ts00004 |
 customers_ts00004_customer_name_idx |             1 |            0
             5 | shard           | public     | customers_ts00005 |
 customers_ts00005_customer_name_idx |             1 |            0
             5 | shard           | public     | customers_ts00006 |
 customers_ts00006_customer_name_idx |             1 |            0
             6 | shard           | public     | customers_ts00007 |
 customers_ts00007_customer_name_idx |             1 |            0
             6 | shard           | public     | customers_ts00008 |
 customers_ts00008_customer_name_idx |             1 |            0
(8 rows)
```

limitless_statio_all_tables

此视图包含数据库分片组中所有表的输入/输出（I/O）统计数据。例如：

```
postgres_limitless=> SELECT
    subcluster_id,
    subcluster_type,
    schemaname,
    relname,
    heap_blks_read,
    heap_blks_hit
FROM
    rds_aurora.limitless_statio_all_tables
WHERE
    relname LIKE 'customers_ts%';

 subcluster_id | subcluster_type | schemaname |      relname      | heap_blks_read |
 heap_blks_hit
---------------+-----------------+------------+-------------------+----------------
+---------------
             3 | shard           | public     | customers_ts00002 |            305 |
      57780
             3 | shard           | public     | customers_ts00001 |            300 |
      56972
             4 | shard           | public     | customers_ts00004 |            302 |
      57291
             4 | shard           | public     | customers_ts00003 |            302 |
      57178
```

```
             5 | shard           | public     | customers_ts00006 |            300 |
      56932
             5 | shard           | public     | customers_ts00005 |            302 |
      57386
             6 | shard           | public     | customers_ts00008 |            300 |
      56881
             6 | shard           | public     | customers_ts00007 |            304 |
      57635
(8 rows)
```

limitless_tables

此视图包含有关 Aurora PostgreSQL Limitless Database 中表的信息。

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_tables;

 table_gid | local_oid | schema_name | table_name  | table_status | table_type  |
 distribution_key
-----------+-----------+-------------+-------------+--------------+-------------
+------------------
         5 |     18635 | public      | placeholder | active       | placeholder |
         6 |     18641 | public      | ref         | active       | reference   |
         7 |     18797 | public      | orders      | active       | sharded     | HASH
 (order_id)
         2 |     18579 | public      | customer    | active       | sharded     | HASH
 (cust_id)
(4 rows)
```

limitless_table_collocations

此视图包含有关并置的分片表的信息。

在以下示例中，orders 和 customers 表并置，users 和 followers 表并置。并置的表具有相同的 collocation_id。

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_table_collocations ORDER BY
 collocation_id;

 collocation_id | schema_name | table_name
----------------+-------------+------------
              2 | public      | orders
              2 | public      | customers
              5 | public      | users
```

```
              5 | public      | followers
(4 rows)
```

limitless_table_collocation_distributions

此视图显示了每个并置的密钥分配几率。

```
postgres_limitless=> SELECT * FROM
 rds_aurora.limitless_table_collocation_distributions ORDER BY collocation_id,
 lower_bound;

 collocation_id | subcluster_id |     lower_bound      |     upper_bound
----------------+---------------+----------------------+----------------------
              2 |             6 | -9223372036854775808 | -4611686018427387904
              2 |             5 | -4611686018427387904 |                    0
              2 |             4 |                    0 |  4611686018427387904
              2 |             3 |  4611686018427387904 |  9223372036854775807
              5 |             6 | -9223372036854775808 | -4611686018427387904
              5 |             5 | -4611686018427387904 |                    0
              5 |             4 |                    0 |  4611686018427387904
              5 |             3 |  4611686018427387904 |  9223372036854775807
(8 rows)
```

Aurora PostgreSQL Limitless Database 的等待事件

Aurora PostgreSQL 中的等待事件表示会话正在等待的资源，例如输入/输出（I/O）和锁。等待事件有助于找出会话等待资源的原因，并找出瓶颈。有关更多信息，请参阅 Aurora PostgreSQL 等待事件。

Aurora PostgreSQL Limitless Database 有自己的与路由器和分片相关的等待事件。其中许多是为等待分片完成任务的路由器准备的。分片等待事件包含正在执行的任务的详细信息。

主题

- 查询等待事件
- Limitless Database 等待事件

查询等待事件

您可以使用 limitless_stat_activity 视图来查询等待事件，如以下示例所示。

```
SELECT wait_event FROM rds_aurora.limitless_stat_activity WHERE
 wait_event_type='AuroraLimitless';

      wait_event
----------------------
 RemoteStatementSetup
 RemoteStatementSetup
(2 rows)
```

您还可以使用 `aurora_stat_system_waits` 函数列出等待次数和在每个等待事件上花费的总时间，如以下示例所示。

```
postgres_limitless=> SELECT type_name,event_name,waits,wait_time
    FROM aurora_stat_system_waits()
    NATURAL JOIN aurora_stat_wait_event()
    NATURAL JOIN aurora_stat_wait_type()
    WHERE type_name='AuroraLimitless'
    ORDER BY wait_time DESC;

    type_name    |       event_name          |  waits  |  wait_time
-----------------+---------------------------+---------+-------------
 AuroraLimitless | RemoteStatementSetup      |    7518 | 75236507897
 AuroraLimitless | RemoteStatementExecution  |      40 |      132986
 AuroraLimitless | Connect                   |       5 |        1453
```

```
(3 rows)
```

Limitless Database 等待事件

以下等待事件适用于 Aurora PostgreSQL Limitless Database。您可以监控这些等待事件，以识别 Aurora PostgreSQL Limitless Database 处理中的瓶颈。

主题

- IO:TwophaseFilePoolWrite 等待事件
- IO:TwophaseFilePoolRead 等待事件
- AuroraLimitless:Connect 等待事件
- AuroraLimitless:AsyncConnect 等待事件
- AuroraLimitless:RemoteStatementSetup 等待事件
- AuroraLimitless:RemoteDDLExecution 等待事件
- AuroraLimitless:RemoteStatementExecution 等待事件
- AuroraLimitless:FetchRemoteResults 等待事件
- AuroraLimitless:AsyncGetInitialResponse 等待事件
- AuroraLimitless:AsyncGetNextResponse 等待事件
- AuroraLimitless:AbortedCommandCleanup 等待事件
- AuroraLimitless:DistributedCommitPrepare 等待事件
- AuroraLimitless:DistributedCommit 等待事件
- AuroraLimitless:DistributedCommitPrepareThrottle 等待事件
- AuroraLimitless:PreparedTransactionResolution 等待事件
- AuroraLimitless:SendPreparedTransactionOutcome 等待事件
- AuroraLimitless:CommitClockBarrier 等待事件
- AuroraLimitless:SnapshotClockBarrier 等待事件
- AuroraLimitless:ReaderSnapshotClockBarrier 等待事件
- AuroraLimitless:GatherDistributedDeadlockGraph 等待事件
- AuroraLimitless:DistributedDeadlockDetection 等待事件
- AuroraLimitless:DistributedDeadlockAbort 等待事件
- AuroraLimitless:GatherRemoteStats 等待事件
- AuroraLimitless:GlobalSequenceRefresh 等待事件

- AuroraLimitless:GlobalVacuumTimeExchange 等待事件
- AuroraLimitless:DistributedTransactionMonitorGather 等待事件
- AuroraLimitlessActivity:AdminTaskSchedulerMain 等待事件
- AuroraLimitlessActivity:AdminTaskExecutorMain 等待事件
- AuroraLimitlessActivity:AdminTaskMonitorMain 等待事件
- AuroraLimitlessActivity:DatabaseCleanupMonitorMain 等待事件
- AuroraLimitlessActivity:TopologyCleanupMonitorMain 等待事件
- AuroraLimitlessActivity:ToplogyChangeMonitorMain 等待事件
- AuroraLimitlessActivity:DistributedTransactionMonitorMain 等待事件
- AuroraLimitlessActivity:GlobalVacuumMonitorMain 等待事件

IO:TwophaseFilePoolWrite 等待事件

等待写入两阶段状态文件池中的两阶段状态文件。这是 Aurora 特有的事件。

原因

执行 `PREPARED TRANSACTION` 命令的进程，包括 Limitless Database 分布式事务的参与者，必须将事务状态保留在两阶段文件中。Aurora 使用文件池来提高此操作的性能。

操作

这是同步写入 I/O 操作，因此该事件中的高延迟与 `IO:XactSync` 的原因类似，也可以用同样的方式进行调查。如果使用 Limitless Database，则可能需要减少正在执行的分布式事务的数量。

IO:TwophaseFilePoolRead 等待事件

等待读取两阶段状态文件池中的两阶段状态文件。

原因

对先前准备好的事务执行 `COMMIT PREPARED` 命令的进程，包括 Limitless Database 分布式事务的参与者，可能需要从两阶段文件中读取先前保留的事务状态。Aurora 使用文件池来提高此操作的性能。

操作

这是读取 I/O 操作。因此，此事件中的高延迟与 `IO:DataFileRead` 的原因类似，也可以进行同样的调查。如果使用 Limitless Database，则可能需要减少正在执行的分布式事务的数量。

AuroraLimitless:Connect 等待事件

该进程正在等待与集群中另一个节点建立连接。

原因

在进程和远程节点之间建立连接，以执行查询、分布式事务和执行 DDL。

操作

减少集群的同步连接数或调整跨分片查询的使用。

AuroraLimitless:AsyncConnect 等待事件

此事件类似于 `Connect`，但表示一个进程在等待与一组节点建立并行连接。

原因

并行连接的建立通常是在执行 DDL 语句时完成的。

操作

减少 DDL 语句的数量或在同一个会话中合并多个 DDL 以提高连接的重复使用率。

AuroraLimitless:RemoteStatementSetup 等待事件

该进程正在等待远程查询执行设置，例如打开、关闭游标或创建准备好的语句。

原因

在无法对语句进行单分片优化的分片表上，此等待事件会随着扫描次数的增加而增加。

操作

优化查询以减少扫描操作的数量或提高单分片优化的资格。

AuroraLimitless:RemoteDDLExecution 等待事件

进程正在等待远程 DDL 命令完成。

原因

在数据库分片组上发出 DDL 命令时，必须先将其分发到其他路由器和分片节点，然后才能确认操作。某些 DDL 操作可能会运行很长时间，因为数据必须适应架构的变化。

操作

识别长时间运行的 DDL 命令，以便对其进行优化。

AuroraLimitless:RemoteStatementExecution 等待事件

进程正在等待远程命令完成。

原因

SQL 命令正在远程节点上运行。此事件将频繁出现，用于内部通信，例如 auto_analyze 和心跳检查。

操作

使用 limitless_stat_statements 视图识别长时间运行的命令。在许多情况下，这是预期事件，特别是对于后台工作人员或内部流程而言，无需采取任何操作。

AuroraLimitless:FetchRemoteResults 等待事件

进程正在等待从远程节点检索行。

原因

从远程表（例如分片表或引用表）读取大量行时，此等待事件可能会增加。

操作

使用 limitless_stat_statements 视图识别未优化的 SELECT 查询。优化查询以仅检索必要的数据。您也可以调整 rds_aurora.limitless_maximum_adaptive_fetch_size 参数。

AuroraLimitless:AsyncGetInitialResponse 等待事件

在查询执行中使用管道模式时，该进程正在等待初始响应。

原因

对于使用单分片数据放置的查询，这通常会在路由器到分片执行期间出现，并且是正常执行的预期部分。

操作

无需进一步操作。

AuroraLimitless:AsyncGetNextResponse 等待事件

在查询执行中使用管道模式时，该进程正在等待额外响应。

原因

对于使用单分片数据放置的查询，这通常会在路由器到分片执行期间出现，并且是正常执行的预期部分。

操作

无需进一步操作。

AuroraLimitless:AbortedCommandCleanup 等待事件

进程正在等待远程清理查询的结果。向分片节点发出清理查询，以便在分布式事务结束时将其恢复到适当的状态。

原因

事务清理是在事务中止时完成的，要么是因为发现了错误，要么是因为用户发出了明确的 ABORT 命令或取消了正在运行的查询。

操作

调查事务被取消的原因。

AuroraLimitless:DistributedCommitPrepare 等待事件

该进程正在提交分布式事务，正在等待所有参与者确认准备命令。

原因

修改多个节点的事务必须执行分布式提交。`DistributedCommitPrepare` 中的长时间等待可能是由于参与节点上的 `IO:TwophaseFilePoolWrite` 事件等待时间过长所致。

操作

减少修改多个节点上数据的事务数量。调查集群其他节点中的 `IO:TwophaseFilePoolWrite` 事件。

AuroraLimitless:DistributedCommit 等待事件

该进程正在提交分布式事务，正在等待主要参与者确认提交命令。

原因

修改多个节点的事务必须执行分布式提交。DistributedCommit 中的长时间等待可能是由于主要参与者的 IO:XactSync 事件等待时间过长所致。

操作

减少修改多个节点上数据的事务数量。调查集群其他节点中的 IO:XactSync 事件。

AuroraLimitless:DistributedCommitPrepareThrottle 等待事件

该进程正在尝试准备分布式事务，但由于存在已准备好的事务，该进程已被限制。

原因

修改多个节点的事务必须执行分布式提交。作为提交协议的一部分，这些事务的参与者必须执行准备操作。Aurora 限制了并发准备的数量，如果超过此限制，进程将在 DistributedCommitPrepareThrottle 事件中等待。

操作

减少修改多个节点上数据的事务数量。调查 IO:TwophaseFilePoolWrite 事件，因为这些事件中的时间延长可能会导致现有的准备事务积累，从而限制新的准备尝试。

AuroraLimitless:PreparedTransactionResolution 等待事件

该进程遇到了一个由处于准备状态的分布式事务修改的元组。该进程必须确定分布式事务是否会在其快照中可见。

原因

修改多个节点的事务必须执行分布式提交，其中包括准备阶段。大量分布式事务或分布式提交延迟增加可能会导致其他进程遇到 PreparedTransactionResolution 等待事件。

操作

减少修改多个节点上数据的事务数量。调查与分布式提交相关的事件，因为这些事件中的时间延长可能会增加分布式事务提交路径的延迟。您可能还想调查网络和 CPU 负载。

AuroraLimitless:SendPreparedTransactionOutcome 等待事件

该进程正在协调分布式事务的节点上执行，而另一个进程已询问该事务的状态，或者该进程已提交分布式事务并将结果发送给参与者。

原因

遇到 PreparedTransactionResolution 等待事件的进程将查询事务协调器。事务协调器上的回复将遇到 SendPreparedTransactionOutcome。

操作

减少修改多个节点上数据的事务数量。调查与分布式提交相关事件以及 IO:TwophaseFilePoolRead 和 IO:TwophaseFilePoolWrite 事件，因为这些事件可能会增加分布式事务提交路径的延迟。您可能还想调查网络和 CPU 负载。

AuroraLimitless:CommitClockBarrier 等待事件

该进程正在提交事务，必须等待以确保集群中所有节点的分配提交时间均为过去时间。

原因

CPU 或网络饱和可能会导致时钟偏移增加，从而导致在此等待事件中耗费时间。

操作

调查集群中的 CPU 或网络饱和度。

AuroraLimitless:SnapshotClockBarrier 等待事件

该进程已收到来自另一个节点的快照时间，其时钟有一个未来时间，正在等待自己的时钟达到该时间。

原因

这通常发生在进程收到向下推到分片的函数结果并且节点之间存在时钟偏移之后。CPU 或网络饱和可能会导致时钟偏移增加，从而导致在此等待事件中耗费时间。

操作

调查集群中的 CPU 或网络饱和度。

AuroraLimitless:ReaderSnapshotClockBarrier 等待事件

此事件发生在读取节点上。该进程正在等待读取节点重放写入流，以便应用在进程快照时间之前发生的所有写入操作。

原因

在这种情况下，Aurora 副本延迟的增加可能会导致等待时间延长。

操作

调查 Aurora 副本延迟。

AuroraLimitless:GatherDistributedDeadlockGraph 等待事件

作为分布式死锁检测的一部分，该进程正在与其他节点通信以收集锁图。

原因

当一个进程等待锁定时，它将在等待的时间超过 `rds_aurora.limitless_distributed_deadlock_timeout` 之后执行分布式死锁检查。

操作

调查应用程序中争用锁的原因，并考虑调整 `rds_aurora.limitless_distributed_deadlock_timeout`。

AuroraLimitless:DistributedDeadlockDetection 等待事件

该进程正在与其他节点通信，以检测分布式死锁。

原因

当一个进程等待锁定时，它将在等待的时间超过 `rds_aurora.limitless_distributed_deadlock_timeout` 之后执行分布式死锁检查。

操作

调查应用程序中争用锁的原因，并考虑调整 `rds_aurora.limitless_distributed_deadlock_timeout`。

AuroraLimitless:DistributedDeadlockAbort 等待事件

该进程正在与另一个节点通信，以中止在分布式死锁中被选为受害者的会话。

原因

应用程序模式会导致分布式死锁。

操作

调查导致分布式死锁的应用程序模式。

AuroraLimitless:GatherRemoteStats 等待事件

该进程正在从集群中的其他节点收集统计数据。

原因

监控或活动查询和视图（例如 `limitless_stat_activity`）将从其他节点检索统计数据。

操作

无需进一步操作。

AuroraLimitless:GlobalSequenceRefresh 等待事件

该进程正在生成一个新的序列值，必须从全局序列中请求一个新的区块。

原因

如果 `rds_aurora.limitless_sequence_chunk_size` 不足，序列值生成速度过快可能导致该事件停滞。

操作

这是正常现象。如果您在此事件中看到时间过长，请考虑调整 `rds_aurora.limitless_sequence_chunk_size`。请参阅 Limitless Database 中有关序列的文档。

AuroraLimitless:GlobalVacuumTimeExchange 等待事件

该进程正在交换快照数据以支持清理。

原因

Limitless Database 中的节点与其他节点交换最旧的活动快照时间数据，以计算清理执行的正确截止时间。

操作

无需进一步操作。

AuroraLimitless:DistributedTransactionMonitorGather 等待事件

该进程正在从其他节点收集事务元数据以支持分布式事务清理。

原因

Limitless Database 中的节点与其他节点交换事务元数据，以确定何时可以清除分布式事务状态。

操作

无需进一步操作。

AuroraLimitlessActivity:AdminTaskSchedulerMain 等待事件

在任务调度程序进程的主循环中等待。

AuroraLimitlessActivity:AdminTaskExecutorMain 等待事件

在任务执行进程的主循环中等待。

AuroraLimitlessActivity:AdminTaskMonitorMain 等待事件

在任务监控进程的主循环中等待。

AuroraLimitlessActivity:DatabaseCleanupMonitorMain 等待事件

在数据库清理监控进程的主循环中等待。

AuroraLimitlessActivity:TopologyCleanupMonitorMain 等待事件

在拓扑清理监控进程的主循环中等待。

AuroraLimitlessActivity:ToplogyChangeMonitorMain 等待事件

在拓扑更改监控进程的主循环中等待。

AuroraLimitlessActivity:DistributedTransactionMonitorMain 等待事件

在分布式事务监控进程的主循环中等待。

AuroraLimitlessActivity:GlobalVacuumMonitorMain 等待事件

在全局清理监控进程的主循环中等待。

备份和还原 Aurora PostgreSQL Limitless Database

您可以备份和还原使用 Aurora PostgreSQL Limitless Database 的数据库集群。

目录

- 备份使用 Aurora PostgreSQL Limitless Database 的数据库集群
  - 创建数据库集群快照
- 还原使用 Aurora PostgreSQL Limitless Database 的数据库集群
  - 从数据库快照还原数据库集群
  - 使用时间点故障恢复还原数据库集群
- 不支持 PostgreSQL 备份和还原实用程序

备份使用 Aurora PostgreSQL Limitless Database 的数据库集群

与备份标准 Aurora 数据库集群相比，使用 Aurora PostgreSQL Limitless Database 备份数据库集群在功能上有相似之处，也有不同之处。

- 当您为使用 Limitless Database 的 Aurora 数据库集群手动拍摄数据库集群快照时，快照会包含来自数据库分片组的数据。
- 持续备份包含来自数据库分片组的数据。
- 自动每日快照包含来自数据库分片组的数据。
- 支持复制数据库集群快照。有关更多信息，请参阅数据库集群快照复制。
- 支持分享数据库集群快照。有关更多信息，请参阅 共享数据库集群快照。
- 您不能使用 `pg_dump` 或 `pg_dumpall` 实用程序备份数据库分片组中的数据库。
- Aurora PostgreSQL Limitless Database 支持在删除数据库集群时拍摄最终快照。
- Aurora PostgreSQL Limitless Database 不支持在删除数据库集群时保留自动备份。

创建数据库集群快照

创建 Aurora PostgreSQL Limitless Database 数据库集群快照的方式与创建标准 Aurora 数据库集群的方式相同，如以下 AWS CLI 示例所示：

```
aws rds create-db-cluster-snapshot \
    --db-cluster-identifier my-db-cluster \
    --db-cluster-snapshot-identifier my-db-cluster-snapshot
```

有关备份数据库集群的更多信息，请参阅备份和还原 Aurora 数据库集群的概述。

还原使用 Aurora PostgreSQL Limitless Database 的数据库集群

与还原标准 Aurora 数据库集群相比，使用 Aurora PostgreSQL Limitless Database 还原数据库集群在功能上有相似之处，也有不同之处。

- 您只能从使用与 Limitless Database 兼容的数据库引擎版本的源数据库集群中还原 Limitless Database 数据库集群，例如 `16.4-limitless`。
- 从使用 Limitless Database 的数据库集群的手动快照还原数据库集群时，整个数据库集群存储空间都将还原。这包括数据库分片组的存储空间。

  您必须创建数据库分片组才能访问 Limitless Database 的存储空间。
- 您可以使用时间点故障恢复（PITR）将数据库集群还原到保留期内的任意时间点。恢复的数据库集群包括数据库分片组的存储空间。

  您必须创建数据库分片组才能访问 Limitless Database 的存储空间。
- 已删除的 Aurora PostgreSQL Limitless Database 数据库集群不支持 PITR。
- 从自动每日快照还原数据库集群，数据库分片组的存储空间也会还原。
- 还原 Aurora PostgreSQL Limitless Database 数据库集群时，必须启用“增强监控”和“性能详情”。务必要包含性能详情 KMS 密钥 ID。

还原 Aurora PostgreSQL Limitless Database 数据库集群后，请务必通过对其运行查询来验证其功能。

从数据库快照还原数据库集群

以下 AWS CLI 示例说明如何从数据库集群快照中还原 Aurora PostgreSQL Limitless Database 数据库集群。

您必须使用 `16.4-limitless` 数据库引擎版本。

从数据库集群快照中还原 Limitless Database 数据库集群。

1. 还原数据库集群：

```
aws rds restore-db-cluster-from-snapshot \
    --db-cluster-identifier my-new-db-cluster \
    --snapshot-identifier my-db-cluster-snapshot \
    --engine aurora-postgresql \
    --engine-version 16.4-limitless \
    --enable-performance-insights \
    --performance-insights-retention-period 31 \
    --performance-insights-kms-key-id arn:aws:kms:us-
east-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab \
    --monitoring-interval 5 \
    --monitoring-role-arn arn:aws:iam::123456789012:role/EMrole
```

2. 创建数据库分片组：

```
aws rds create-db-shard-group \
    --db-cluster-identifier my-new-db-cluster \
    --db-shard-group-identifier my-new-DB-shard-group \
    --max-acu 1000
```

   有关更多信息，请参阅 向现有 Aurora PostgreSQL Limitless Database 数据库集群添加数据库分片组。

有关从数据库集群快照中还原 Aurora 数据库集群的更多信息，请参阅从数据库集群快照还原。

使用时间点故障恢复还原数据库集群

以下 AWS CLI 示例说明如何使用时间点故障恢复（PITR）还原 Aurora PostgreSQL Limitless Database 数据库集群。

使用 PITR 还原 Limitless Database 数据库集群

1. 还原数据库集群：

```
aws rds restore-db-cluster-to-point-in-time \
    --source-db-cluster-identifier my-db-cluster \
    --db-cluster-identifier my-new-db-cluster \
    --use-latest-restorable-time \
    --enable-performance-insights \
    --performance-insights-retention-period 31 \
    --performance-insights-kms-key-id arn:aws:kms:us-
east-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab \
    --monitoring-interval 5 \
    --monitoring-role-arn arn:aws:iam::123456789012:role/EMrole
```

2. 创建数据库分片组：

```
aws rds create-db-shard-group \
    --db-cluster-identifier my-new-db-cluster \
    --db-shard-group-identifier my-new-DB-shard-group \
    --max-acu 1000
```

   有关更多信息，请参阅 向现有 Aurora PostgreSQL Limitless Database 数据库集群添加数据库分片组。

有关 PITR 的更多信息，请参阅将数据库集群还原到指定时间。

不支持 PostgreSQL 备份和还原实用程序

主数据库集群或数据库分片组均不支持以下 PostgreSQL 实用程序：

- `pg_dump`
- `pg_dumpall`
- `pg_restore`

虽然您可以通过开源二进制文件或其他方法使用它们，但这样做可能会产生不一致的结果。

升级 Amazon Aurora PostgreSQL Limitless Database

以下内容适用于升级 Aurora PostgreSQL Limitless Database：

- 支持次要版本升级。
- 支持修补 Aurora PostgreSQL Limitless Database。补丁显示为待维护状态，将在维护时段内处理。

升级使用 Amazon Aurora PostgreSQL Limitless Database 的数据库集群

您可以通过修改数据库集群并选择新的数据库引擎版本来升级数据库集群。您可以使用 AWS Management Console或 AWS CLI。

控制台

升级您的 Limitless Database 数据库集群

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 导航到数据库页面。
3. 选择要升级的 Limitless Database 数据库集群。
4. 选择修改。

   此时会显示修改数据库集群页面。
5. 对于数据库引擎版本，选择新的数据库引擎版本，例如带 Limitless Database 的 Aurora PostgreSQL（兼容 PostgreSQL 16.6）。
6. 选择继续。
7. 在摘要页面上，选择是立即应用更改，还是在下一个计划维护时段内应用更改，然后选择修改集群。

CLI

请使用 modify-db-cluster AWS CLI 命令，如以下示例所示。

```
aws rds modify-db-cluster \
    --db-cluster-identifier my-sv2-cluster \
    --engine-version 16.6-limitless \
    --apply-immediately
```

Aurora PostgreSQL Limitless Database 参考

我们为 Aurora PostgreSQL Limitless Database 提供了以下参考主题。

主题

- 支持和不支持的数据定义语言（DDL）SQL 命令
- Aurora PostgreSQL Limitless Database 的 DDL 限制和其他信息
- 支持和不支持的数据操作语言（DML）和查询处理 SQL 命令
- Aurora PostgreSQL Limitless Database 的 DML 限制和其他信息
- Aurora PostgreSQL Limitless Database 中的变量
- Aurora PostgreSQL Limitless Database 中的数据库集群参数

支持和不支持的数据定义语言（DDL）SQL 命令

下表列出了 Aurora PostgreSQL Limitless Database 支持和不支持的 DDL 命令，以及限制参考或更多信息。

命令	是否支持？	限制或更多信息
ALTER AGGREGATE	否	不适用
ALTER COLLATION	是	无
ALTER CONVERSION	是	无
ALTER DATABASE	否	不适用
ALTER DEFAULT PRIVILEGES	否	不适用
ALTER DOMAIN	否	不适用
ALTER EVENT TRIGGER	否	不适用
ALTER EXTENSION	是	扩展
ALTER FOREIGN DATA WRAPPER	否	不适用

命令	是否支持？	限制或更多信息
ALTER FOREIGN TABLE	否	不适用
ALTER FUNCTION	是	函数
ALTER GROUP	是	无
ALTER INDEX	是	无
ALTER LANGUAGE	否	不适用
ALTER LARGE OBJECT	否	不适用
ALTER MATERIALIZED VIEW	否	不适用
ALTER OPERATOR	是	无
ALTER OPERATOR CLASS	是	无
ALTER OPERATOR FAMILY	是	无
ALTER POLICY	否	不适用
ALTER PROCEDURE	是	无
ALTER PUBLICATION	否	不适用
ALTER ROLE	是	无
ALTER ROUTINE	否	不适用
ALTER RULE	否	不适用
ALTER SCHEMA	是	无
ALTER SEQUENCE	是	序列
ALTER SERVER	否	不适用
ALTER STATISTICS	否	不适用

命令	是否支持？	限制或更多信息
ALTER SUBSCRIPTION	否	不适用
ALTER SYSTEM	否	不适用
ALTER TABLE	是	ALTER TABLE
ALTER TABLESPACE	否	不适用
ALTER TEXT SEARCH CONFIGURATION	否	不适用
ALTER TEXT SEARCH DICTIONARY	否	不适用
ALTER TEXT SEARCH PARSER	否	不适用
ALTER TEXT SEARCH TEMPLATE	否	不适用
ALTER TRIGGER	否	不适用
ALTER TYPE	是	无
ALTER USER	是	无
ALTER USER MAPPING	否	不适用
ALTER VIEW	是	无
COMMENT	否	不适用
CREATE ACCESS METHOD	否	不适用
CREATE AGGREGATE	否	不适用
CREATE CAST	是	无
CREATE COLLATION	是	无

命令	是否支持？	限制或更多信息
CREATE CONVERSION	是	无
CREATE DATABASE	是	CREATE DATABASE
CREATE DOMAIN	否	不适用
CREATE EVENT TRIGGER	否	不适用
CREATE EXTENSION	是	扩展
CREATE FOREIGN DATA WRAPPER	否	不适用
CREATE FOREIGN TABLE	否	不适用
CREATE FUNCTION	是	函数
CREATE GROUP	是	无
CREATE INDEX	是	CREATE INDEX
CREATE LANGUAGE	否	不适用
CREATE MATERIALIZED VIEW	否	不适用
CREATE OPERATOR	是	无
CREATE OPERATOR CLASS	是	无
CREATE OPERATOR FAMILY	是	无
CREATE POLICY	是	无
CREATE PROCEDURE	是	无
CREATE PUBLICATION	否	不适用
CREATE ROLE	是	无

命令	是否支持？	限制或更多信息
CREATE RULE	否	不适用
CREATE SCHEMA	是	CREATE SCHEMA
CREATE SEQUENCE	是	序列
CREATE SERVER	否	不适用
CREATE STATISTICS	否	不适用
CREATE SUBSCRIPTION	否	不适用
CREATE TABLE	是	CREATE TABLE
CREATE TABLE AS	是	CREATE TABLE AS
CREATE TABLESPACE	否	不适用
CREATE TEMPORARY TABLE	否	不适用
CREATE TEMPORARY TABLE AS	否	不适用
CREATE TEXT SEARCH CONFIGURATION	否	不适用
CREATE TEXT SEARCH DICTIONARY	否	不适用
CREATE TEXT SEARCH PARSER	否	不适用
CREATE TEXT SEARCH TEMPLATE	否	不适用
CREATE TRANSFORM	否	不适用
CREATE TRIGGER	否	不适用

命令	是否支持？	限制或更多信息
CREATE TYPE	是	无
CREATE USER	是	无
CREATE USER MAPPING	否	不适用
CREATE VIEW	是	无
DROP ACCESS METHOD	否	不适用
DROP AGGREGATE	是	无
DROP CAST	是	无
DROP COLLATION	是	无
DROP CONVERSION	是	无
DROP DATABASE	是	DROP DATABASE
DROP DOMAIN	否	不适用
DROP EVENT TRIGGER	否	不适用
DROP EXTENSION	是	扩展
DROP FOREIGN DATA WRAPPER	否	不适用
DROP FOREIGN TABLE	否	不适用
DROP FUNCTION	是	函数
DROP GROUP	是	无
DROP INDEX	是	无
DROP LANGUAGE	否	不适用
DROP MATERIALIZED VIEW	否	不适用

命令	是否支持？	限制或更多信息
DROP OPERATOR	是	无
DROP OPERATOR CLASS	是	无
DROP OPERATOR FAMILY	是	无
DROP OWNED	否	不适用
DROP POLICY	否	不适用
DROP PROCEDURE	是	无
DROP PUBLICATION	否	不适用
DROP ROLE	是	无
DROP ROUTINE	否	不适用
DROP RULE	否	不适用
DROP SCHEMA	是	无
DROP SEQUENCE	是	无
DROP SERVER	否	不适用
DROP STATISTICS	否	不适用
DROP SUBSCRIPTION	否	无
DROP TABLE	是	无
DROP TABLESPACE	否	不适用
DROP TEXT SEARCH CONFIGURATION	否	不适用
DROP TEXT SEARCH DICTIONARY	否	不适用

命令	是否支持？	限制或更多信息
DROP TEXT SEARCH PARSER	否	不适用
DROP TEXT SEARCH TEMPLATE	否	不适用
DROP TRANSFORM	否	不适用
DROP TRIGGER	否	不适用
DROP TYPE	是	无
DROP USER	是	无
DROP USER MAPPING	否	不适用
DROP VIEW	是	无
GRANT	是	无
REASSIGN OWNED	否	不适用
REVOKE	是	无
SECURITY LABEL	否	不适用
SELECT INTO	是	SELECT INTO
SET	是	无
SET CONSTRAINTS	否	不适用
SET ROLE	是	无
SET SESSION AUTHORIZATION	是	无
SET TRANSACTION	是	无
TRUNCATE	是	无

Aurora PostgreSQL Limitless Database 的 DDL 限制和其他信息

以下主题描述了 Aurora PostgreSQL Limitless Database 中的 DDL SQL 命令的限制或提供了更多信息。

主题

- ALTER TABLE
- CREATE DATABASE
- CREATE INDEX
- CREATE SCHEMA
- CREATE TABLE
- CREATE TABLE AS
- DROP DATABASE
- SELECT INTO
- 约束
- 默认值
- 扩展
- 外键
- 函数
- 序列

ALTER TABLE

Aurora PostgreSQL Limitless Database 通常支持 ALTER TABLE 命令。有关更多信息，请参阅 PostgreSQL 文档中的 ALTER TABLE。

限制

对于受支持的选项，ALTER TABLE 具有以下限制。

移除一列

- 在分片表上，您无法移除属于分片键的列。
- 在引用表上，您无法移除主键列。

更改列的数据类型

- 不支持 USING 表达式。
- 在分片表上，您无法更改属于分片键的列类型。

添加或移除约束

有关不支持的内容的详细信息，请参阅约束。

更改列的默认值

支持默认值。有关更多信息，请参阅 默认值。

不支持的选项

不支持某些选项，因为它们依赖于不支持的功能，例如触发器。

不支持以下 ALTER TABLE 的表级选项：

- ALL IN TABLESPACE
- ATTACH PARTITION
- DETACH PARTITION
- ONLY 标志
- RENAME CONSTRAINT

不支持以下 ALTER TABLE 的列级选项：

- ADD GENERATED

- DROP EXPRESSION [IF EXISTS]
- DROP IDENTITY [IF EXISTS]
- RESET
- RESTART
- SET
- SET COMPRESSION
- SET STATISTICS

CREATE DATABASE

在 Aurora PostgreSQL Limitless Database 中，仅支持无限数据库。

运行 CREATE DATABASE 时，在一个或多个节点中成功创建的数据库可能会在其他节点中失效，因为数据库创建是一项非事务性操作。在这种情况下，成功创建的数据库对象将在预定时间内自动从所有节点中移除，以保持数据库分片组的一致性。在此期间，重新创建同名数据库可能会导致错误，提示该数据库已存在。

支持以下选项：

- 排序规则：

```
CREATE DATABASE name WITH
    [LOCALE = locale]
    [LC_COLLATE = lc_collate]
    [LC_CTYPE = lc_ctype]
    [ICU_LOCALE = icu_locale]
    [ICU_RULES = icu_rules]
    [LOCALE_PROVIDER = locale_provider]
    [COLLATION_VERSION = collation_version];
```

- CREATE DATABASE WITH OWNER:

```
CREATE DATABASE name WITH OWNER = user_name;
```

不支持以下选项：

- CREATE DATABASE WITH TABLESPACE:

```
CREATE DATABASE name WITH TABLESPACE = tablespace_name;
```

- CREATE DATABASE WITH TEMPLATE:

```
CREATE DATABASE name WITH TEMPLATE = template;
```

CREATE INDEX

分片表支持 CREATE INDEX CONCURRENTLY：

```
CREATE INDEX CONCURRENTLY index_name ON table_name(column_name);
```

所有表类型都支持 CREATE UNIQUE INDEX：

```
CREATE UNIQUE INDEX index_name ON table_name(column_name);
```

不支持 CREATE UNIQUE INDEX CONCURRENTLY：

```
CREATE UNIQUE INDEX CONCURRENTLY index_name ON table_name(column_name);
```

有关更多信息，请参阅 UNIQUE。有关创建索引的一般信息，请参阅 PostgreSQL 文档中的 CREATE INDEX。

显示索引

当您使用 \d *table_name* 或类似命令时，并非所有索引在路由器上都可见。相反，可以使用 pg_catalog.pg_indexes 视图获取索引，如以下示例中所示。

```
SET rds_aurora.limitless_create_table_mode='sharded';
SET rds_aurora.limitless_create_table_shard_key='{"id"}';
CREATE TABLE items (id int PRIMARY KEY, val int);
CREATE INDEX items_my_index on items (id, val);

postgres_limitless=> SELECT * FROM pg_catalog.pg_indexes WHERE tablename='items';

 schemaname | tablename |   indexname    | tablespace |
   indexdef
------------+-----------+----------------+------------
+-------------------------------------------------------------------------
```

```
 public     | items     | items_my_index |            | CREATE INDEX items_my_index
 ON ONLY public.items USING btree (id, val)
 public     | items     | items_pkey     |            | CREATE UNIQUE INDEX
 items_pkey ON ONLY public.items USING btree (id)
(2 rows)
```

CREATE SCHEMA

不支持带有架构元素的 CREATE SCHEMA：

```
CREATE SCHEMA my_schema CREATE TABLE (column_name INT);
```

这将生成类似于以下内容的错误：

```
ERROR: CREATE SCHEMA with schema elements is not supported
```

CREATE TABLE

不支持 CREATE TABLE 语句中的关系，例如：

```
CREATE TABLE orders (orderid int, customerId int, orderDate date) WITH
 (autovacuum_enabled = false);
```

不支持 IDENTITY 列，例如：

```
CREATE TABLE orders (orderid INT GENERATED ALWAYS AS IDENTITY);
```

Aurora PostgreSQL Limitless Database 支持最多 54 个字符作为分片表名称。

CREATE TABLE AS

要使用 CREATE TABLE AS 创建表，必须使用 rds_aurora.limitless_create_table_mode 变量。对于分片表，您还必须使用 rds_aurora.limitless_create_table_shard_key 变量。有关更多信息，请参阅 使用变量创建无限表。

```
-- Set the variables.
SET rds_aurora.limitless_create_table_mode='sharded';
SET rds_aurora.limitless_create_table_shard_key='{"a"}';
```

```
CREATE TABLE ctas_table AS SELECT 1 a;

-- "source" is the source table whose columns and data types are used to create the new
 "ctas_table2" table.
CREATE TABLE ctas_table2 AS SELECT a,b FROM source;
```

您无法使用 CREATE TABLE AS 创建引用表，因为它们需要主键约束。CREATE TABLE AS 不会将主键传播到新表。

有关常规信息，请参阅 PostgreSQL 文档中的 CREATE TABLE AS。

DROP DATABASE

您可以删除已创建的数据库。

DROP DATABASE 命令在后台异步运行。在运行时，如果您尝试创建具有相同名称的新数据库，您将收到一条错误消息。

SELECT INTO

SELECT INTO 在功能上类似于 CREATE TABLE AS。您必须使用 rds_aurora.limitless_create_table_mode 变量。对于分片表，您还必须使用 rds_aurora.limitless_create_table_shard_key 变量。有关更多信息，请参阅 使用变量创建无限表。

```
-- Set the variables.
SET rds_aurora.limitless_create_table_mode='sharded';
SET rds_aurora.limitless_create_table_shard_key='{"a"}';

-- "source" is the source table whose columns and data types are used to create the new
 "destination" table.
SELECT * INTO destination FROM source;
```

当前，SELECT INTO 操作通过路由器执行，而不直接通过分片执行。因此，性能可能会有局限性。

有关常规信息，请参阅 PostgreSQL 文档中的 SELECT INTO。

约束

以下限制适用于 Aurora PostgreSQL Limitless Database 中的约束。

CHECK

支持涉及文字比较运算符的简单约束。不支持需要函数验证的更复杂表达式和约束，如以下示例所示。

```
CREATE TABLE my_table (
    id  INT CHECK (id > 0)                                          -- supported
  , val INT CHECK (val > 0 AND val < 1000)                          -- supported
  , tag TEXT CHECK (length(tag) > 0)                                -- not supported:
 throws "Expression inside CHECK constraint is not supported"
  , op_date TIMESTAMP WITH TIME ZONE CHECK (op_date <= now())  -- not supported:
 throws "Expression inside CHECK constraint is not supported"
);
```

您还可以为约束提供显式名称，如以下示例所示。

```
CREATE TABLE my_table (
    id  INT CONSTRAINT positive_id  CHECK (id > 0)
  , val INT CONSTRAINT val_in_range CHECK (val > 0 AND val < 1000)
);
```

您还可以将表级约束语法与 CHECK 约束一起使用，如以下示例所示。

```
CREATE TABLE my_table (
    id INT CONSTRAINT positive_id  CHECK (id > 0)
  , min_val INT CONSTRAINT min_val_in_range CHECK (min_val > 0 AND min_val < 1000)
  , max_val INT
  , CONSTRAINT max_val_in_range CHECK (max_val > 0 AND max_val < 1000 AND max_val >
 min_val)
);
```

EXCLUDE

Aurora PostgreSQL Limitless Database 不支持排除约束。

FOREIGN KEY

有关更多信息，请参阅 外键。

NOT NULL

支持 NOT NULL 约束，无任何限制。

PRIMARY KEY

主键意味着唯一约束，因此对唯一约束的相同限制也适用于主键。这意味着：

- 如果将表转换为分片表，则分片键必须是主键的子集。也就是说，主键包含分片键的所有列。
- 如果将表转换为引用表，则该表必须具有主键。

以下示例演示了如何使用主键。

```
-- Create a standard table.
CREATE TABLE public.my_table (
    item_id INT
  , location_code INT
  , val INT
  , comment text
);

-- Change the table to a sharded table using the 'item_id' and 'location_code'
 columns as shard keys.
CALL rds_aurora.limitless_alter_table_type_sharded('public.my_table',
 ARRAY['item_id', 'location_code']);
```

正在尝试添加不包含分片键的主键：

```
-- Add column 'item_id' as the primary key.
-- Invalid because the primary key doesnt include all columns from the shard key:
-- 'location_code' is part of the shard key but not part of the primary key
ALTER TABLE public.my_table ADD PRIMARY KEY (item_id); -- ERROR

-- add column "val" as primary key
-- Invalid because primary key does not include all columns from shard key:
--  item_id and location_code iare part of shard key but not part of the primary key
ALTER TABLE public.my_table ADD PRIMARY KEY (item_id); -- ERROR
```

正在尝试添加包含分片键的主键：

```
-- Add the 'item_id' and 'location_code' columns as the primary key.
-- Valid because the primary key contains the shard key.
ALTER TABLE public.my_table ADD PRIMARY KEY (item_id, location_code); -- OK
```

```
-- Add the 'item_id', 'location_code', and 'val' columns as the primary key.
-- Valid because the primary key contains the shard key.
ALTER TABLE public.my_table ADD PRIMARY KEY (item_id, location_code, val); -- OK
```

将标准表更改为引用表。

```
-- Create a standard table.
CREATE TABLE zipcodes (zipcode INT PRIMARY KEY, details VARCHAR);

-- Convert the table to a reference table.
CALL rds_aurora.limitless_alter_table_type_reference('public.zipcode');
```

有关创建分片表和引用表的更多信息，请参阅创建 Aurora PostgreSQL Limitless Database 表。

UNIQUE

在分片表中，唯一键必须包含分片键，也就是说，分片键必须是唯一键的子集。将表类型更改为分片表时会检查此项。在引用表中没有任何限制。

```
CREATE TABLE customer (
    customer_id INT NOT NULL
  , zipcode INT
  , email TEXT UNIQUE
);
```

支持表级 UNIQUE 约束，如以下示例所示。

```
CREATE TABLE customer (
    customer_id INT NOT NULL
  , zipcode INT
  , email TEXT
  , CONSTRAINT zipcode_and_email UNIQUE (zipcode, email)
);
```

以下示例显示了如何同时使用主键和唯一键。两个键都必须包含分片键。

```
SET rds_aurora.limitless_create_table_mode='sharded';
SET rds_aurora.limitless_create_table_shard_key='{"p_id"}';
CREATE TABLE t1 (
p_id BIGINT NOT NULL,
c_id BIGINT NOT NULL,
```

```
PRIMARY KEY (p_id),
UNIQUE (p_id, c_id)
);
```

有关更多信息，请参阅 PostgreSQL 文档中的 Constraints。

默认值

Aurora PostgreSQL Limitless Database 支持默认值中的表达式。

以下示例显示如何使用默认值。

```
CREATE TABLE t (
    a INT DEFAULT 5,
    b TEXT DEFAULT 'NAN',
    c NUMERIC
);

CALL rds_aurora.limitless_alter_table_type_sharded('t', ARRAY['a']);
INSERT INTO t DEFAULT VALUES;
SELECT * FROM t;

 a |  b  | c
---+-----+---
 5 | NAN |
(1 row)
```

支持表达式，如以下示例所示。

```
CREATE TABLE t1 (a NUMERIC DEFAULT random());
```

以下示例添加了一个新列，该列为 NOT NULL 且具有默认值。

```
ALTER TABLE t ADD COLUMN d BOOLEAN NOT NULL DEFAULT FALSE;
SELECT * FROM t;

 a |  b  | c | d
---+-----+---+---
 5 | NAN |   | f
(1 row)
```

以下示例使用默认值更改现有列。

```
ALTER TABLE t ALTER COLUMN c SET DEFAULT 0.0;
INSERT INTO t DEFAULT VALUES;
SELECT * FROM t;

 a |  b  | c   |  d
---+-----+-----+-----
 5 | NAN |     | f
 5 | NAN | 0.0 | f
(2 rows)
```

以下示例删除默认值。

```
ALTER TABLE t ALTER COLUMN a DROP DEFAULT;
INSERT INTO t DEFAULT VALUES;
SELECT * FROM t;

 a |  b  | c   |  d
---+-----+-----+-----
 5 | NAN |     | f
 5 | NAN | 0.0 | f
   | NAN | 0.0 | f
(3 rows)
```

有关更多信息，请参阅 PostgreSQL 文档中的 Default values。

扩展

Aurora PostgreSQL Limitless Database 支持以下 PostgreSQL 扩展：

- `aurora_limitless_fdw` – 此扩展已预装。您无法删除它。
- `aws_s3` – 此扩展在 Aurora PostgreSQL Limitless Database 中的工作方式与在 Aurora PostgreSQL 中的工作方式类似。

  您可以将数据从 Amazon S3 存储桶导入 Aurora PostgreSQL Limitless Database 数据库集群中，或将数据从 Aurora PostgreSQL Limitless Database 数据库集群导出至 Amazon S3 存储桶。有关更多信息，请参阅将 Amazon S3 中的数据导入到 Aurora PostgreSQL 数据库集群和将数据从 Aurora PostgreSQL 数据库集群导出到 Amazon S3。
- `btree_gin`
- `citext`
- `ip4r`
- `pg_buffercache` – 此扩展在 Aurora PostgreSQL Limitless Database 中的行为与社区 PostgreSQL 中的行为不同。有关更多信息，请参阅 Aurora PostgreSQL Limitless Database 中的 pg_buffercache 差异。
- `pg_stat_statements`
- `pg_trgm`
- `pgcrypto`
- `pgstattuple` – 此扩展在 Aurora PostgreSQL Limitless Database 中的行为与社区 PostgreSQL 中的行为不同。有关更多信息，请参阅 Aurora PostgreSQL Limitless Database 中的 pgstattuple 差异。
- `pgvector`
- `plpgsql` – 此扩展程序已预装，但您可以将其删除。
- `PostGIS` – 不支持长时间事务和表管理功能。不支持修改空间引用表。
- `unaccent`
- `uuid`

Aurora PostgreSQL Limitless Database 目前不支持大多数 PostgreSQL 扩展。但是，您仍然可以使用 shared_preload_libraries（SPL）配置设置将扩展加载到 Aurora PostgreSQL 主数据库集群中。它们还会加载到 Aurora PostgreSQL Limitless Database 中，但可能无法正常运行。

例如，您可以加载 `pg_hint_plan` 扩展，但加载它并不能保证使用查询注释中传递的提示。

> **Note**
>
> 您无法修改与 pg_stat_statements 扩展关联的对象。有关安装 pg_stat_statements 的信息，请参阅 limitless_stat_statements。

您可以使用 pg_available_extensions 和 pg_available_extension_versions 函数查找 Aurora PostgreSQL Limitless Database 中支持的扩展。

扩展支持以下 DDL：

CREATE EXTENSION

您可以像在 PostgreSQL 中那样创建扩展。

```
CREATE EXTENSION [ IF NOT EXISTS ] extension_name
    [ WITH ] [ SCHEMA schema_name ]
             [ VERSION version ]
             [ CASCADE ]
```

有关更多信息，请参阅 PostgreSQL 文档中的 CREATE EXTENSION。

ALTER EXTENSION

支持以下 DDL：

```
ALTER EXTENSION name UPDATE [ TO new_version ]

ALTER EXTENSION name SET SCHEMA new_schema
```

有关更多信息，请参阅 PostgreSQL 文档中的 ALTER EXTENSION。

DROP EXTENSION

您可以像在 PostgreSQL 中那样删除扩展。

```
DROP EXTENSION [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]
```

有关更多信息，请参阅 PostgreSQL 文档中的 DROP EXTENSION。

扩展不支持以下 DDL：

ALTER EXTENSION

您无法添加或删除扩展中的成员对象。

```
ALTER EXTENSION name ADD member_object

ALTER EXTENSION name DROP member_object
```

Aurora PostgreSQL Limitless Database 中的 pg_buffercache 差异

在 Aurora PostgreSQL Limitless Database 中，当您安装 pg_buffercache 扩展并使用 pg_buffercache 视图时，您只能从当前连接的节点接收与缓冲区相关的信息：路由器。同样，使用函数 pg_buffercache_summary 或 pg_buffercache_usage_counts 仅提供来自所连接节点的信息。

您可能有多个节点，可能需要从任意节点访问缓冲区信息才能有效诊断问题。因此，Limitless Database 提供以下函数：

- rds_aurora.limitless_pg_buffercache(*subcluster_id*)
- rds_aurora.limitless_pg_buffercache_summary(*subcluster_id*)
- rds_aurora.limitless_pg_buffercache_usage_counts(*subcluster_id*)

通过输入任何节点（无论是路由器还是分片）的子集群 ID，您可以轻松访问该节点的特定缓冲区信息。当您在无限数据库中安装 pg_buffercache 扩展时，这些函数可以直接使用。

> **Note**
>
> Aurora PostgreSQL Limitless Database 支持 pg_buffercache 扩展版本 1.4 及更高版本的函数。

limitless_pg_buffercache 视图中显示的列与 pg_buffercache 视图中显示的列略有不同：

- bufferid – 与 pg_buffercache 保持不变。
- relname – 与 pg_buffercache 中显示文件节点编号不同，limitless_pg_buffercache 如果在当前数据库或共享系统目录中可用，则显示相关的 relname，否则为 NULL。
- parent_relname – 如果 relname 列中的值代表分区表（在分片表的情况下）pg_buffercache 列中没有的这一新列会显示父 relname。否则，将显示 NULL。

- spcname – 与 pg_buffercache 中显示表空间对象标识符（OID）不同，limitless_pg_buffercache 显示表空间名称。
- datname – 与 pg_buffercache 中显示数据库 OID 不同，limitless_pg_buffercache 显示数据库名称。
- relforknumber – 与 pg_buffercache 保持不变。
- relblocknumber – 与 pg_buffercache 保持不变。
- isdirty – 与 pg_buffercache 保持不变。
- usagecount – 与 pg_buffercache 保持不变。
- pinning_backends – 与 pg_buffercache 保持不变。

limitless_pg_buffercache_summary 和 limitless_pg_buffercache_usage_counts 视图中的列分别与常规 pg_buffercache_summary 和 pg_buffercache_usage_counts 视图中的列相同。

通过使用这些功能，您可以访问 Limitless Database 环境中所有节点的详细缓冲区缓存信息，从而更有效地诊断和管理数据库系统。

Aurora PostgreSQL Limitless Database 中的 pgstattuple 差异

在 Aurora PostgreSQL 中，pgstattuple 扩展目前不支持外部表、分区表或分区索引。但是，在 Aurora PostgreSQL Limitless Database 中，用户创建的对象通常属于这些不受支持的类型。虽然有常规表和索引（例如，目录表及其索引），但大多数对象都驻留在外部节点上，这使得它们成为路由器的外部对象。

我们认识到此扩展对于获取元组级统计数据的重要性，这对于消除膨胀和收集诊断信息等任务至关重要。因此，Aurora PostgreSQL Limitless Database 为无限数据库中的 pgstattuple 扩展提供了支持。

Aurora PostgreSQL Limitless Database 在 rds_aurora 架构中包含以下函数：

元组级统计函数

rds_aurora.limitless_pgstattuple(*relation_name*)

- 用途：提取标准表及其索引的元组级统计数据
- 输入：relation_name（文本）– 关系的名称
- 输出：与 Aurora PostgreSQL 中 pgstattuple 函数返回的列一致

rds_aurora.limitless_pgstattuple(*relation_name*, *subcluster_id*)

- 用途：提取引用表、分片表、目录表及其索引的元组级统计数据
- 输入:
  - relation_name（文本）– 关系的名称
  - subcluster_id（文本）– 要提取统计数据的节点的子集群 ID
- 输出：
  - 对于引用表和目录表（包括其索引），列与 Aurora PostgreSQL 中的列一致。
  - 对于分片表，统计数据仅表示位于指定子集群上的分片表的分区。

索引统计函数

rds_aurora.limitless_pgstatindex(*relation_name*)

- 用途：提取标准表上的 B 树索引的统计数据
- 输入：relation_name（文本）– B 树索引的名称
- 输出：返回除 root_block_no 外的所有列。返回的列与 Aurora PostgreSQL 中的 pgstatindex 函数一致

rds_aurora.limitless_pgstatindex(*relation_name*, *subcluster_id*)

- 用途：提取引用表、分片表和目录表的 B 树索引的统计数据。
- 输入:
  - relation_name（文本）– B 树索引的名称
  - subcluster_id（文本）– 要提取统计数据的节点的子集群 ID
- 输出：
  - 对于引用表和目录表索引，将返回所有列（除 root_block_no 外）。返回的列与 Aurora PostgreSQL 一致
  - 对于分片表，统计数据仅表示位于指定子集群上的分片表索引的分区。tree_level 列显示请求的子集群上所有表切片的平均值。

rds_aurora.limitless_pgstatginindex(*relation_name*)

- 用途：提取标准表上通用倒排索引（GIN）的统计数据
- 输入：relation_name（文本）– GIN 的名称
- 输出：与 Aurora PostgreSQL 中 pgstatginindex 函数返回的列一致

rds_aurora.limitless_pgstatginindex(*relation_name*, *subcluster_id*)

- 用途：提取引用表、分片表和目录表的 GIN 索引的统计数据。

- 输入:
  - relation_name（文本）– 索引的名称
  - subcluster_id（文本）– 要提取统计数据的节点的子集群 ID
- 输出：
  - 对于引用表和目录表 GIN 索引，列与 Aurora PostgreSQL 中的列一致。
  - 对于分片表，统计数据仅表示位于指定子集群上的分片表索引的分区。

rds_aurora.limitless_pgstathashindex(*relation_name*)

- 用途：提取标准表上的哈希索引的统计数据
- 输入：relation_name（文本）– 哈希索引的名称
- 输出：与 Aurora PostgreSQL 中 pgstathashindex 函数返回的列一致

rds_aurora.limitless_pgstathashindex(*relation_name*, *subcluster_id*)

- 用途：提取引用表、分片表和目录表的哈希索引的统计数据。
- 输入:
  - relation_name（文本）– 索引的名称
  - subcluster_id（文本）– 要提取统计数据的节点的子集群 ID
- 输出：
  - 对于引用表和目录表哈希索引，列与 Aurora PostgreSQL 一致。
  - 对于分片表，统计数据仅表示位于指定子集群上的分片表索引的分区。

页面计数函数

rds_aurora.limitless_pg_relpages(*relation_name*)

- 用途：提取标准表及其索引的页面计数
- 输入：relation_name（文本）– 关系的名称
- 输出：指定关系的页面计数

rds_aurora.limitless_pg_relpages(*relation_name*, *subcluster_id*)

- 用途：提取引用表、分片表和目录表（包括其索引）的页面计数
- 输入:
  - relation_name（文本）– 关系的名称
  - subcluster_id（文本）– 要提取页面计数的节点的子集群 ID
- 输出：对于分片表，页面计数是指定子集群上所有表切片的页面总和。

近似元组级统计函数

`rds_aurora.limitless_pgstattuple_approx(relation_name)`

- 用途：提取标准表及其索引的近似元组级统计数据
- 输入：`relation_name`（文本）– 关系的名称
- 输出：与 Aurora PostgreSQL 中 pgstattuple_approx 函数返回的列一致

`rds_aurora.limitless_pgstattuple_approx(relation_name, subcluster_id)`

- 用途：提取引用表、分片表和目录表（包括其索引）的近似元组级统计数据
- 输入:
  - `relation_name`（文本）– 关系的名称
  - `subcluster_id`（文本）– 要提取统计数据的节点的子集群 ID
- 输出：
  - 对于引用表和目录表（包括其索引），列与 Aurora PostgreSQL 中的列一致。
  - 对于分片表，统计数据仅表示位于指定子集群上的分片表的分区。

> **Note**
>
> 目前，Aurora PostgreSQL Limitless Database 不支持实体化视图、TOAST 表或临时表上的 `pgstattuple` 扩展。
> 尽管 Aurora PostgreSQL 支持其他格式，但在 Aurora PostgreSQL Limitless Database 中，您必须以文本形式提供输入。

外键

支持外键 (FOREIGN KEY) 约束，但有一些限制：

- 带 FOREIGN KEY 的 CREATE TABLE 仅支持标准表。要使用 FOREIGN KEY 创建分片表或引用表，请先创建不带外键约束的表。然后使用以下语句对其进行修改：

```
ALTER TABLE ADD CONSTRAINT;
```

- 当表具有外键约束时，不支持将标准表转换为分片表或引用表。删除约束，在转换后再添加。
- 以下限制适用于外键约束的表类型：
  - 标准表可以对另一个标准表有外键约束。
  - 如果父表和子表并置且外键是分片键的超集，则分片表可能具有外键约束。
  - 分片表可以对引用表有外键约束。
  - 引用表可以对另一个引用表有外键约束。

主题

- 外键选项
- 示例

外键选项

Aurora PostgreSQL Limitless Database 支持某些 DDL 选项的外键。下表列出了 Aurora PostgreSQL Limitless Database 表之间支持和不支持的选项。

DDL 选项	引用到引用	分片到分片（并置）	分片到引用	标准到标准
DEFERRABLE	支持	是	是	是
INITIALLY DEFERRED	是	是	是	是
INITIALLY IMMEDIATE	是	是	是	是
MATCH FULL	是	是	是	是

DDL 选项	引用到引用	分片到分片（并置）	分片到引用	标准到标准
MATCH PARTIAL	否	否	否	否
MATCH SIMPLE	是	是	是	是
NOT DEFERRABLE	是	是	是	是
NOT VALID	是	否	否	是
ON DELETE CASCADE	是	是	是	是
ON DELETE NO ACTION	是	是	是	是
ON DELETE RESTRICT	是	是	是	是
ON DELETE SET DEFAULT	否	否	否	否
ON DELETE SET NULL	是	否	否	是
ON UPDATE CASCADE	否	否	否	是
ON UPDATE NO ACTION	是	是	是	是
ON UPDATE RESTRICT	是	是	是	是
ON UPDATE SET DEFAULT	否	否	否	否

DDL 选项	引用到引用	分片到分片（并置）	分片到引用	标准到标准
ON UPDATE SET NULL	是	否	否	是

示例

- 标准到标准：

```
set rds_aurora.limitless_create_table_mode='standard';

CREATE TABLE products(
    product_no integer PRIMARY KEY,
    name text,
    price numeric
);

CREATE TABLE orders (
    order_id integer PRIMARY KEY,
    product_no integer REFERENCES products (product_no),
    quantity integer
);

SELECT constraint_name, table_name, constraint_type
FROM information_schema.table_constraints WHERE constraint_type='FOREIGN KEY';

 constraint_name         | table_name  | constraint_type
-------------------------+-------------+-----------------
 orders_product_no_fkey  | orders      | FOREIGN KEY
(1 row)
```

- 分片到分片（并置）

```
set rds_aurora.limitless_create_table_mode='sharded';
set rds_aurora.limitless_create_table_shard_key='{"product_no"}';
CREATE TABLE products(
    product_no integer PRIMARY KEY,
    name text,
    price numeric
);
```

```
set rds_aurora.limitless_create_table_shard_key='{"order_id"}';
set rds_aurora.limitless_create_table_collocate_with='products';
CREATE TABLE orders (
    order_id integer PRIMARY KEY,
    product_no integer,
    quantity integer
);

ALTER TABLE orders ADD CONSTRAINT order_product_fk FOREIGN KEY (product_no)
 REFERENCES products (product_no);
```

- 分片到引用：

```
set rds_aurora.limitless_create_table_mode='reference';
CREATE TABLE products(
    product_no integer PRIMARY KEY,
    name text,
    price numeric
);

set rds_aurora.limitless_create_table_mode='sharded';
set rds_aurora.limitless_create_table_shard_key='{"order_id"}';
CREATE TABLE orders (
    order_id integer PRIMARY KEY,
    product_no integer,
    quantity integer
);

ALTER TABLE orders ADD CONSTRAINT order_product_fk FOREIGN KEY (product_no)
 REFERENCES products (product_no);
```

- 引用到引用：

```
set rds_aurora.limitless_create_table_mode='reference';
CREATE TABLE products(
    product_no integer PRIMARY KEY,
    name text,
    price numeric
);
CREATE TABLE orders (
    order_id integer PRIMARY KEY,
    product_no integer,
```

```
    quantity integer
);

ALTER TABLE orders ADD CONSTRAINT order_product_fk FOREIGN KEY (product_no)
 REFERENCES products (product_no);
```

函数

Aurora PostgreSQL Limitless Database 支持函数。

函数支持以下 DDL：

CREATE FUNCTION

您可以像在 Aurora PostgreSQL 中那样创建函数，但无法在替换函数时改变其波动性。

有关更多信息，请参阅 PostgreSQL 文档中的 CREATE FUNCTION。

ALTER FUNCTION

您可以像在 Aurora PostgreSQL 中那样更改函数，但无法改变其波动性。

有关更多信息，请参阅 PostgreSQL 文档中的 ALTER FUNCTION。

DROP FUNCTION

您可以像在 Aurora PostgreSQL 中那样删除函数。

```
DROP FUNCTION [ IF EXISTS ] name [ ( [ [ argmode ] [ argname ] argtype [, ...] ] ) ]
 [, ...]
    [ CASCADE | RESTRICT ]
```

有关更多信息，请参阅 PostgreSQL 文档中的 DROP FUNCTION。

主题

- 函数分布
- 函数波动性

函数分布

当一个函数的所有语句都针对单个分片时，将整个函数向下推送到目标分片是有益的。然后将结果传播回路由器，无需在路由器上展开该函数。函数和存储过程下推功能对于想要在离数据源（即分片）更近的地方运行其函数或存储过程的客户非常有用。

要分发函数，请先创建该函数，然后调用 `rds_aurora.limitless_distribute_function` 流程进行分发。该函数使用以下语法：

```
SELECT rds_aurora.limitless_distribute_function('function_prototype',
 ARRAY['shard_key'], 'collocating_table');
```

函数需要以下参数：

- *function_prototype* – 要分发的函数。仅提及输入参数，不提及任何输出参数。

  如果有任何参数被定义为 OUT 参数，则不要将其类型包含在 function_prototype 的参数中。
- ARRAY['*shard_key*'] – 标识为函数分片键的函数参数列表。
- *collocating_table* – 包含目标分片上数据范围的分片表。

为了确定在哪个分片推送该函数以运行，系统会获取 ARRAY['*shard_key*'] 参数，对其进行哈希处理，然后从 *collocating_table* 中找到托管包含该哈希值范围的分片。

限制

当您分发函数或流程时，它只处理受该分片中分片键范围限制的数据。如果函数或流程尝试访问来自其他分片的数据，则分布式函数或流程返回的结果与非分布式函数或流程返回的结果会有所不同。

例如，您创建了一个包含查询的函数，这些查询将触及多个分片，随后调用 rds_aurora.limitless_distribute_function 流程进行分发。当您通过为分片键提供参数来调用此函数时，运行该函数的结果很可能会受该分片中存在的值的限制。这些结果不同于在不分配函数的情况下生成的结果。

示例

请查看以下函数 func，其有一个带有分片键 customer_id 的分片表 customers。

```
postgres_limitless=> CREATE OR REPLACE FUNCTION func(c_id integer, sc integer)
 RETURNS int
    language SQL
    volatile
    AS $$
    UPDATE customers SET score = sc WHERE customer_id = c_id RETURNING score;
    $$;
```

现在我们分发这个函数：

```
SELECT rds_aurora.limitless_distribute_function('func(integer, integer)',
 ARRAY['c_id'], 'customers');
```

以下是查询计划示例。

```
EXPLAIN(costs false, verbose true) SELECT func(27+1,10);

                    QUERY PLAN
 ---------------------------------------------------
  Foreign Scan
    Output: (func((27 + 1), 10))
    Remote SQL:  SELECT func((27 + 1), 10) AS func
  Single Shard Optimized
 (4 rows)
```

```
EXPLAIN(costs false, verbose true)
 SELECT * FROM customers,func(customer_id, score) WHERE customer_id=10 AND score=27;

                          QUERY PLAN
 ------------------------------------------------------------------------
  Foreign Scan
    Output: customer_id, name, score, func
    Remote SQL:  SELECT customers.customer_id,
      customers.name,
      customers.score,
      func.func
     FROM public.customers,
      LATERAL func(customers.customer_id, customers.score) func(func)
    WHERE ((customers.customer_id = 10) AND (customers.score = 27))
  Single Shard Optimized
 (10 rows)
```

以下示例演示一个以 IN 和 OUT 参数作为参数的存储过程。

```
CREATE OR REPLACE FUNCTION get_data(OUT id INTEGER, IN arg_id INT)
    AS $$
    BEGIN
        SELECT customer_id,
        INTO id
        FROM customer
        WHERE customer_id = arg_id;
```

```
    END;
    $$ LANGUAGE plpgsql;
```

以下示例仅使用 IN 参数来分发该流程。

```
EXPLAIN(costs false, verbose true) SELECT * FROM get_data(1);

             QUERY PLAN
 ----------------------------------
  Foreign Scan
    Output: id
    Remote SQL:  SELECT customer_id
     FROM get_data(1) get_data(id)
  Single Shard Optimized
 (6 rows)
```

函数波动性

您可以通过检查 pg_proc 视图中的 provolatile 值来确定函数是不可变、稳定还是波动的。provolatile 值表示函数的结果是仅取决于其输入参数，还是受外部因素的影响。

此类值可以是以下任一项：

- i – 不可变函数，它们始终为相同的输入提供相同的结果
- s – 稳定函数，其结果（对于固定输入）在扫描中不会改变
- v – 波动函数，其结果可能随时发生变化。对于有副作用的函数，也要使用 v，这样对它们的调用就不会被优化掉。

以下示例显示了波动函数。

```
SELECT proname, provolatile FROM pg_proc WHERE proname='pg_sleep';

 proname  | provolatile
----------+-------------
 pg_sleep | v
(1 row)

SELECT proname, provolatile FROM pg_proc WHERE proname='uuid_generate_v4';

     proname      | provolatile
```

```
------------------+-------------
 uuid_generate_v4 | v
(1 row)

SELECT proname, provolatile FROM pg_proc WHERE proname='nextval';

 proname | provolatile
---------+-------------
 nextval | v
(1 row)
```

Aurora PostgreSQL Limitless Database 不支持更改现有函数的波动性。这适用于 ALTER FUNCTION 和 CREATE OR REPLACE FUNCTION 命令，如以下示例所示。

```
-- Create an immutable function
CREATE FUNCTION immutable_func1(name text) RETURNS text language plpgsql
AS $$
BEGIN
    RETURN name;
END;
$$IMMUTABLE;

-- Altering the volatility throws an error
ALTER FUNCTION immutable_func1 STABLE;

-- Replacing the function with altered volatility throws an error
CREATE OR REPLACE FUNCTION immutable_func1(name text) RETURNS text language plpgsql
AS $$
BEGIN
    RETURN name;
END;
$$VOLATILE;
```

我们强烈建议您为函数分配正确的波动性。例如，如果您的函数使用来自多个表或引用数据库对象的 SELECT，请不要将其设置为 IMMUTABLE。如果表内容发生变化，则不变性就会被打破。

Aurora PostgreSQL 允许在不可变函数内进行 SELECT，但结果可能不正确。Aurora PostgreSQL Limitless Database 可能同时返回错误和错误结果。有关函数波动性的更多信息，请参阅 PostgreSQL 文档中的 Function volatility categories。

序列

命名序列是按升序或降序生成唯一数字的数据库对象。CREATE SEQUENCE 创建新的序列号生成器。序列值保证具有唯一性。

当您在 Aurora PostgreSQL Limitless Database 中创建命名序列时，会创建一个分布式序列对象。然后，Aurora PostgreSQL Limitless Database 在所有分布式事务路由器（路由器）上分发不重叠的序列值块。区块在路由器上以本地序列对象的形式表示；因此，序列操作（如 nextval 和 currval）是在本地运行的。路由器独立运行，并在需要时从分布式序列中请求新的区块。

有关序列的更多信息，请参阅 PostgreSQL 文档中的 CREATE SEQUENCE。

主题

- 请求新区块
- 限制
- 不支持的选项
- 示例
- 顺序视图
- 对序列问题进行故障排除

请求新区块

您可以使用 rds_aurora.limitless_sequence_chunk_size 参数配置在路由器上分配的区块的大小。默认值为 250000。每台路由器最初拥有两个区块：活动区块和保留区块。活动区块用于配置本地序列对象（设置 minvalue 和 maxvalue），保留区块存储在内部目录表中。当活动区块达到最小值或最大值时，它会被保留区块所取代。为此，内部使用了 ALTER SEQUENCE，意味着获取了 AccessExclusiveLock。

后台工作程序每隔 10 秒在路由器节点上运行一次，以扫描序列中是否有已使用的保留区块。如果找到了使用过的区块，则工作程序会从分布式序列中请求一个新的区块。确保将区块大小设置得足够大，以便后台工作人员有足够的时间请求新的区块。远程请求永远不会发生在用户会话的上下文中，这意味着您不能直接请求新的序列。

限制

以下限制适用于 Aurora PostgreSQL Limitless Database 中的序列：

- pg_sequence 目录、pg_sequences 函数和 SELECT * FROM *sequence_name* 语句都只显示局部序列状态，而不显示分布式状态。

- 序列值保证是唯一的，并且保证在会话中是单调的。但是，如果这些会话连接到其他路由器，它们可能会与其他会话中运行的 `nextval` 语句不一致。
- 确保序列大小（可用值的数量）足够大，以便分布在所有路由器上。使用 `rds_aurora.limitless_sequence_chunk_size` 参数配置 `chunk_size`。（每台路由器都有两个区块。）
- 支持 `CACHE` 选项，但缓存必须小于 `chunk_size`。

不支持的选项

Aurora PostgreSQL Limitless Database 中的序列不支持以下选项。

顺序操作函数

不支持 `setval` 函数。有关更多信息，请参阅 PostgreSQL 文档中的 Sequence Manipulation Functions。

CREATE SEQUENCE

不支持以下选项。

```
CREATE [{ TEMPORARY | TEMP} | UNLOGGED] SEQUENCE
    [[ NO ] CYCLE]
```

有关更多信息，请参阅 PostgreSQL 文档中的 CREATE SEQUENCE。

ALTER SEQUENCE

不支持以下选项。

```
ALTER SEQUENCE
    [[ NO ] CYCLE]
```

有关更多信息，请参阅 PostgreSQL 文档中的 ALTER SEQUENCE。

ALTER TABLE

序列不支持 `ALTER TABLE` 命令。

示例

CREATE/DROP SEQUENCE

```
postgres_limitless=> CREATE SEQUENCE s;
CREATE SEQUENCE

postgres_limitless=> SELECT nextval('s');

 nextval
---------
       1
(1 row)

postgres_limitless=> SELECT * FROM pg_sequence WHERE seqrelid='s'::regclass;

 seqrelid | seqtypid | seqstart | seqincrement | seqmax | seqmin | seqcache |
 seqcycle
----------+----------+----------+--------------+--------+--------+----------
+----------
    16960 |       20 |        1 |            1 |  10000 |      1 |        1 | f
(1 row)

% connect to another router
postgres_limitless=> SELECT nextval('s');

 nextval
---------
   10001
(1 row)

postgres_limitless=> SELECT * FROM pg_sequence WHERE seqrelid='s'::regclass;

 seqrelid | seqtypid | seqstart | seqincrement | seqmax | seqmin | seqcache |
 seqcycle
----------+----------+----------+--------------+--------+--------+----------
+----------
    16959 |       20 |    10001 |            1 |  20000 |  10001 |        1 | f
(1 row)

postgres_limitless=> DROP SEQUENCE s;
DROP SEQUENCE
```

ALTER SEQUENCE

```
postgres_limitless=> CREATE SEQUENCE s;
CREATE SEQUENCE

postgres_limitless=> ALTER SEQUENCE s RESTART 500;
ALTER SEQUENCE

postgres_limitless=> SELECT nextval('s');

 nextval
---------
     500
(1 row)

postgres_limitless=> SELECT currval('s');

 currval
---------
     500
(1 row)
```

顺序操作函数

```
postgres=# CREATE TABLE t(a bigint primary key, b bigint);
CREATE TABLE

postgres=# CREATE SEQUENCE s minvalue 0 START 0;
CREATE SEQUENCE

postgres=# INSERT INTO t VALUES (nextval('s'), currval('s'));

INSERT 0 1

postgres=# INSERT INTO t VALUES (nextval('s'), currval('s'));
INSERT 0 1

postgres=# SELECT * FROM t;

 a | b
---+---
 0 | 0
 1 | 1
```

```
(2 rows)

postgres=# ALTER SEQUENCE s RESTART 10000;
ALTER SEQUENCE

postgres=# INSERT INTO t VALUES (nextval('s'), currval('s'));

INSERT 0 1

postgres=# SELECT * FROM t;

   a   |   b
-------+-------
     0 |     0
     1 |     1
 10000 | 10000
(3 rows)
```

顺序视图

Aurora PostgreSQL Limitless Database 为序列提供以下视图。

rds_aurora.limitless_distributed_sequence

此视图显示分布式序列状态和配置。minvalue、maxvalue、start、inc 和 cache 列的含义与 pg_sequences 视图中的含义相同，它们显示了创建序列时使用的选项。lastval 列显示分布式序列对象中最新的分配值或保留值。这并不意味着该值已被使用，因为路由器会在本地保留序列区块。

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_distributed_sequence WHERE
 sequence_name='test_serial_b_seq';

 schema_name |   sequence_name   | lastval | minvalue |  maxvalue  | start | inc |
 cache
-------------+-------------------+---------+----------+------------+-------+-----
+-------
 public      | test_serial_b_seq | 1250000 |        1 | 2147483647 |     1 |   1 |
   1
(1 row)
```

rds_aurora.limitless_sequence_metadata

此视图显示分布式序列元数据并聚合来自集群节点的序列元数据。它使用以下列：

- subcluster_id – 拥有区块的集群节点 ID。
- 活动区块 – 正在使用的序列区块 (active_minvalue、active_maxvalue)。
- 保留区块 – 接下来将使用的本地区块 (reserved_minvalue、reserved_maxvalue)。
- local_last_value – 本地序列中的最后观测值。
- chunk_size – 区块的大小，如创建时所配置。

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_sequence_metadata WHERE
 sequence_name='test_serial_b_seq' order by subcluster_id;

 subcluster_id |   sequence_name   | schema_name | active_minvalue | active_maxvalue
 | reserved_minvalue | reserved_maxvalue | chunk_size | chunk_state |
 local_last_value
---------------+-------------------+-------------+-----------------
+-----------------+-------------------+-------------------+------------
+-------------+------------------
```

```
 1             | test_serial_b_seq | public      |          500001 |          750000
 |           1000001 |           1250000 |     250000 |           1 |
 550010
 2             | test_serial_b_seq | public      |          250001 |          500000
 |            750001 |           1000000 |     250000 |           1 |

(2 rows)
```

对序列问题进行故障排除

序列可能会出现以下问题。

区块大小不够大

如果区块大小设置得不够大，并且事务速率很高，则在活动区块用完之前，后台工作人员可能没有足够的时间请求新的区块。这可能会导致争用和等待事件，例如 LIMITLESS:AuroraLimitlessSequenceReplace、LWLock:LockManager、Lockrelation 和 LWlock:bufferscontent。

增大 rds_aurora.limitless_sequence_chunk_size 参数的值。

序列缓存设置过高

在 PostgreSQL 中，序列缓存发生在会话级别。在一次访问序列对象期间，每个会话都会分配连续的序列值，并相应地增加序列对象的 last_value 值。然后，在该会话中下次使用 nextval 时，只需返回预先分配的值，而无需涉及序列对象。

会话结束时，任何分配但未在会话中使用的数字都将丢失，从而导致序列中出现“空洞”。这可能会快速消耗 sequence_chunk，并导致争用和等待事件，例如 LIMITLESS:AuroraLimitlessSequenceReplace、LWLock:LockManager、Lockrelation 和 LWlock:bufferscontent。

减少序列缓存设置。

下图显示了由序列问题引起的等待事件。


Database load Info
Viewed as
Absolute
Sliced by
Waits
Bar
Line
Table
Show estimated vCPU
Load is measured in average active sessions (AAS). The vCPU value is determined by the number of vCPU (virtual CPU) cores for your DB instance.
Average active sessions (AAS)
2,000
1,500
1,000
500
0
05:42:55
05:43
05:43:05
05:43:10
05:43:15
05:43:20
05:43:25
05:43:30
05:43:35
05:43:40
05:43:45
05:43:50
05:43:55
05:44
05:44:05
05:44:10
05:44:15
05:44:20
IPC:ProcArrayGroupUpdate
LIMITLESS:LimitlessConnect
LIMITLESS:LimitlessExecute
IO:XactSync
LIMITLESS:LimitlessModify
Lock:relation
LWLock:LockManager
LIMITLESS:AuroraLimitlessSequenceRe
LWLock:BufferContent
CPU
Dimensions
Load distribution
Top instances
Top waits
Top SQL
Top hosts
Top users
Top session types
Top applications
Top databases
Top SQL (25)
Find SQL statements
1 2 3
Load by waits (AAS)
SQL statements
441.03
select nextval(?) as order_id
2.50
INSERT INTO ec_sample.orderdetails (customer_id,order_id,orderline_id,product_id...
1.39
INSERT INTO ec_sample.orders (customer_id,order_id,order_date,order_status,updat...

支持和不支持的数据操作语言（DML）和查询处理 SQL 命令

下表列出了 Aurora PostgreSQL Limitless Database 支持和不支持的 DML 命令，以及限制参考或更多信息。

命令	是否支持？	限制或更多信息
ABORT	是	无
ANALYZE	是	ANALYZE
BEGIN	是	无
CALL	是	无
CHECKPOINT	是	无
CLOSE	是	无
CLUSTER	是	CLUSTER
COMMIT	是	无
COMMIT PREPARED	否	不适用
COPY	是	无
DEALLOCATE	是	无
DECLARE	是	无
DELETE	是	无
DISCARD	是	无
DO	是	无
END	是	无
EXECUTE	是	无
EXPLAIN	是	EXPLAIN

命令	是否支持？	限制或更多信息
FETCH	是	无
IMPORT FOREIGN SCHEMA	否	不适用
INSERT	是	INSERT
LISTEN	否	不适用
LOCK	是	无
MERGE	否	不适用
MOVE	是	无
NOTIFY	否	不适用
OPEN	是	无
PREPARE	是	无
PREPARE TRANSACTION	否	不适用
REFRESH MATERIALIZED VIEW	否	不适用
REINDEX	否	不适用
RELEASE SAVEPOINT	是	无
ROLLBACK	是	无
ROLLBACK PREPARED	否	不适用
ROLLBACK TO SAVEPOINT	是	无
SAVEPOINT	是	无
SELECT	是	无
SELECT INTO	是	无

命令	是否支持？	限制或更多信息
SHOW	是	无
START TRANSACTION	是	无
UNLISTEN	否	无
UPDATE	是	UPDATE
UPDATE … WHERE CURRENT OF	否	不适用
VACUUM	是	VACUUM
VALUES	是	无

Aurora PostgreSQL Limitless Database 的 DML 限制和其他信息

以下主题描述了 Aurora PostgreSQL Limitless Database 中的 DML 和查询处理 SQL 命令的限制或提供了更多信息。

主题

- ANALYZE
- CLUSTER
- EXPLAIN
- INSERT
- UPDATE
- VACUUM

ANALYZE

ANALYZE 命令收集有关数据库中表内容的统计数据。随后，查询计划器使用这些统计数据来帮助确定最有效的查询执行计划。有关更多信息，请参阅 PostgreSQL 文档中的 ANALYZE。

在 Aurora PostgreSQL Limitless Database 中，ANALYZE 命令在运行时收集所有路由器和分片的表统计数据。

为避免在 ANALYZE 运行期间在每台路由器上计算统计数据，表统计数据在其中一台路由器上计算，然后再将其复制到对等路由器。

CLUSTER

CLUSTER 命令根据索引对表进行物理重新排序。索引必须已经在表上定义。在 Aurora PostgreSQL Limitless Database 中，集群是每个分片上存在的索引部分的本地集群。

有关更多信息，请参阅 PostgreSQL 文档中的 CLUSTER。

EXPLAIN

您可以使用以下参数来配置 EXPLAIN 命令的输出：

- rds_aurora.limitless_explain_options – EXPLAIN 输出中要包含的内容。默认值为 single_shard_optimization：显示计划是否经过单分片优化，但不包括分片计划。

在此示例中，EXPLAIN 输出未显示来自分片的计划。

```
postgres_limitless=> EXPLAIN SELECT * FROM employees where id =25;

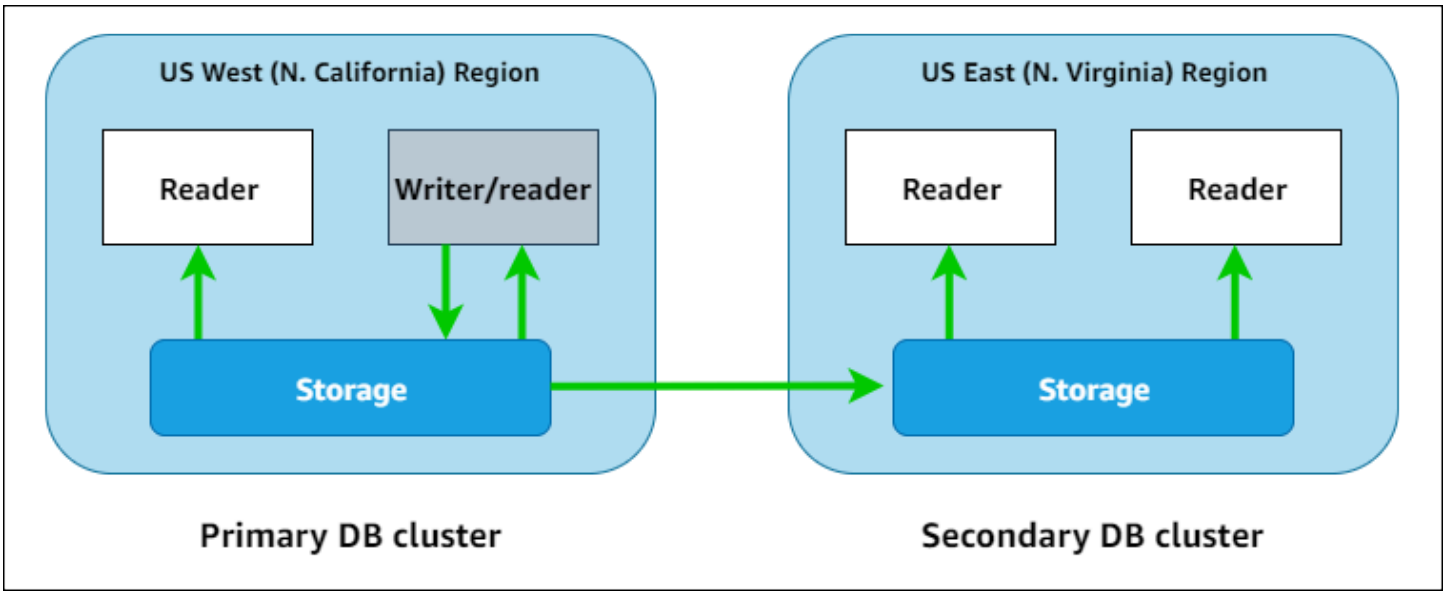
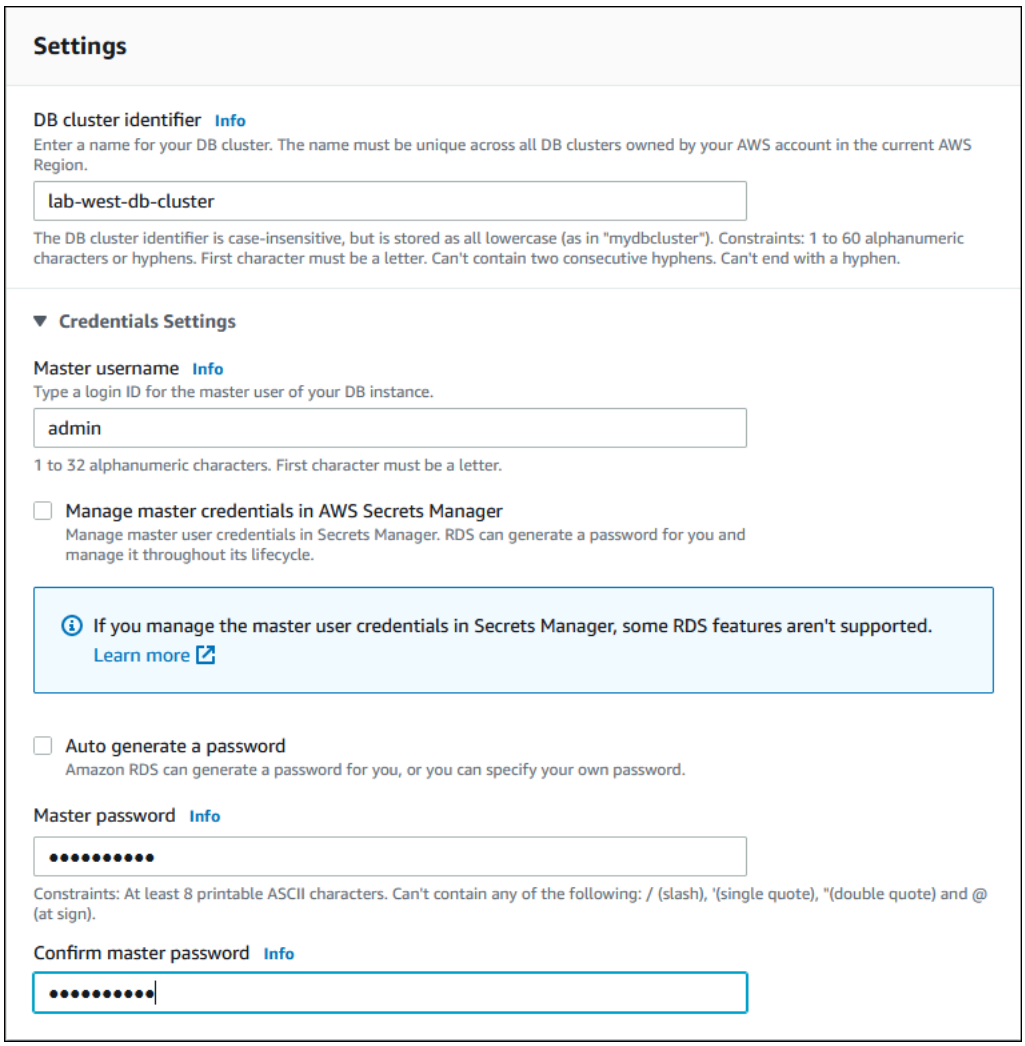
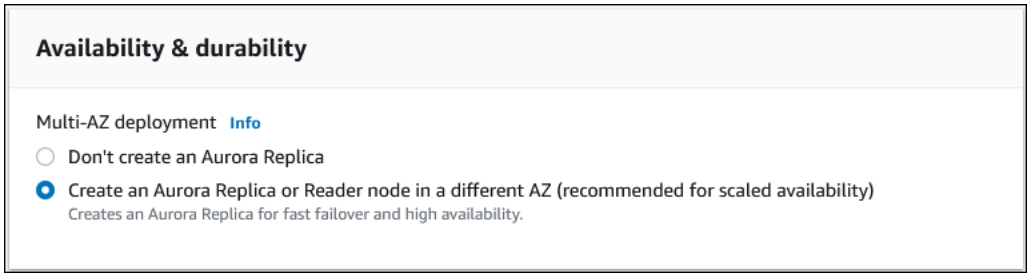
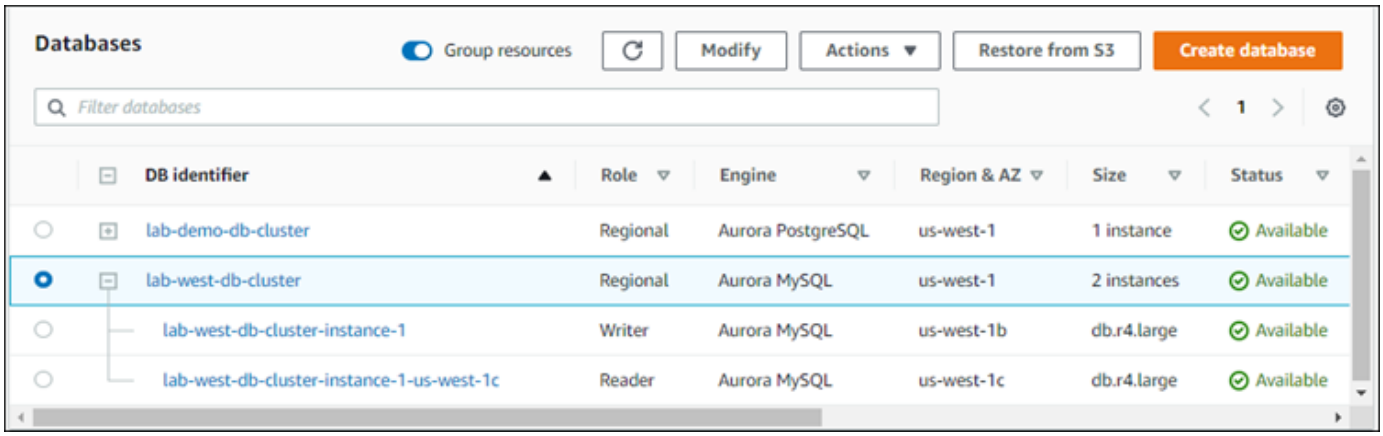
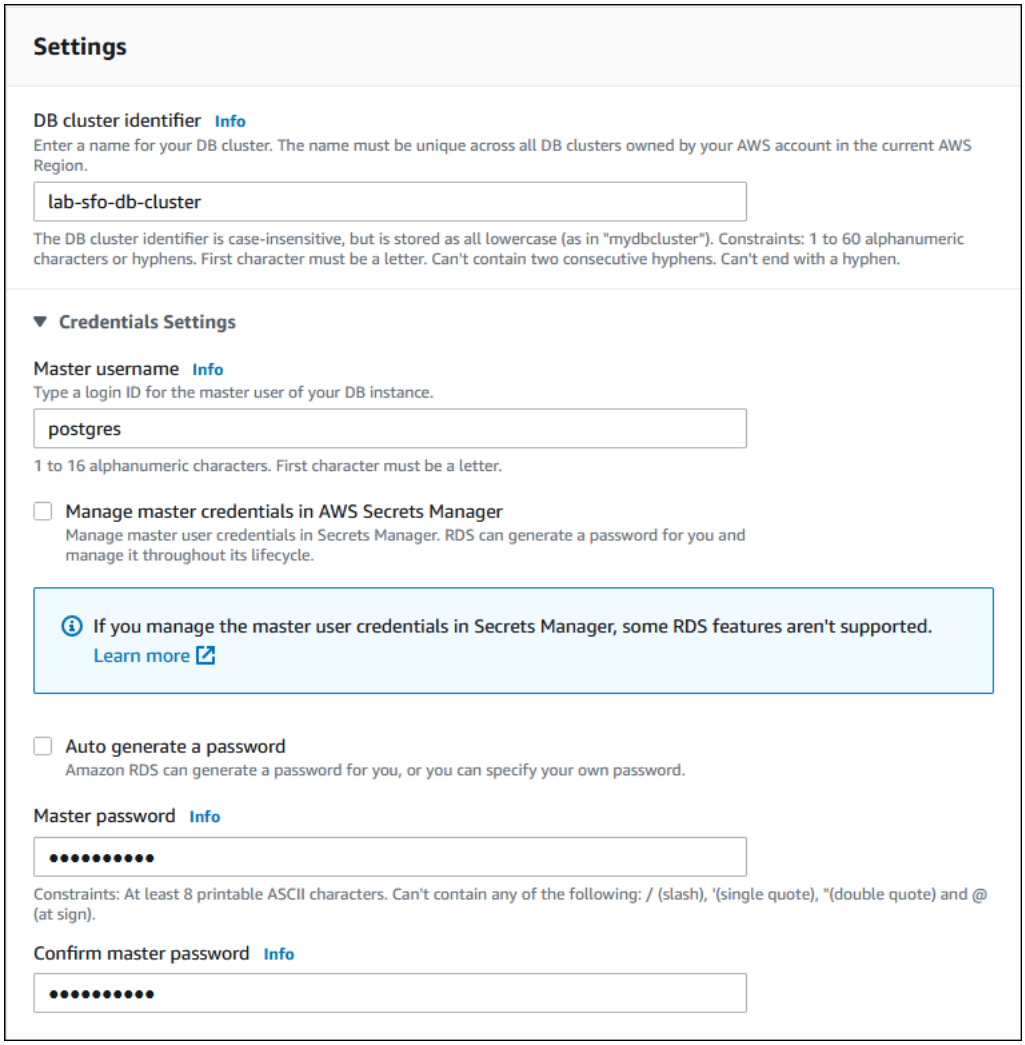
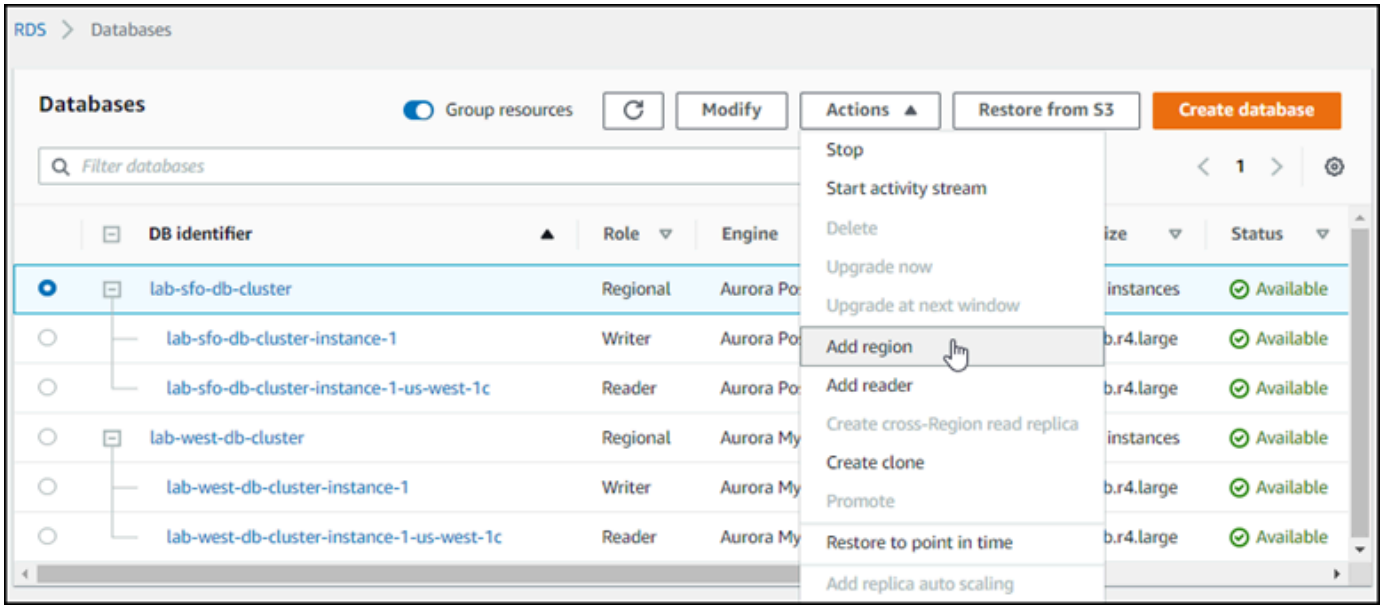
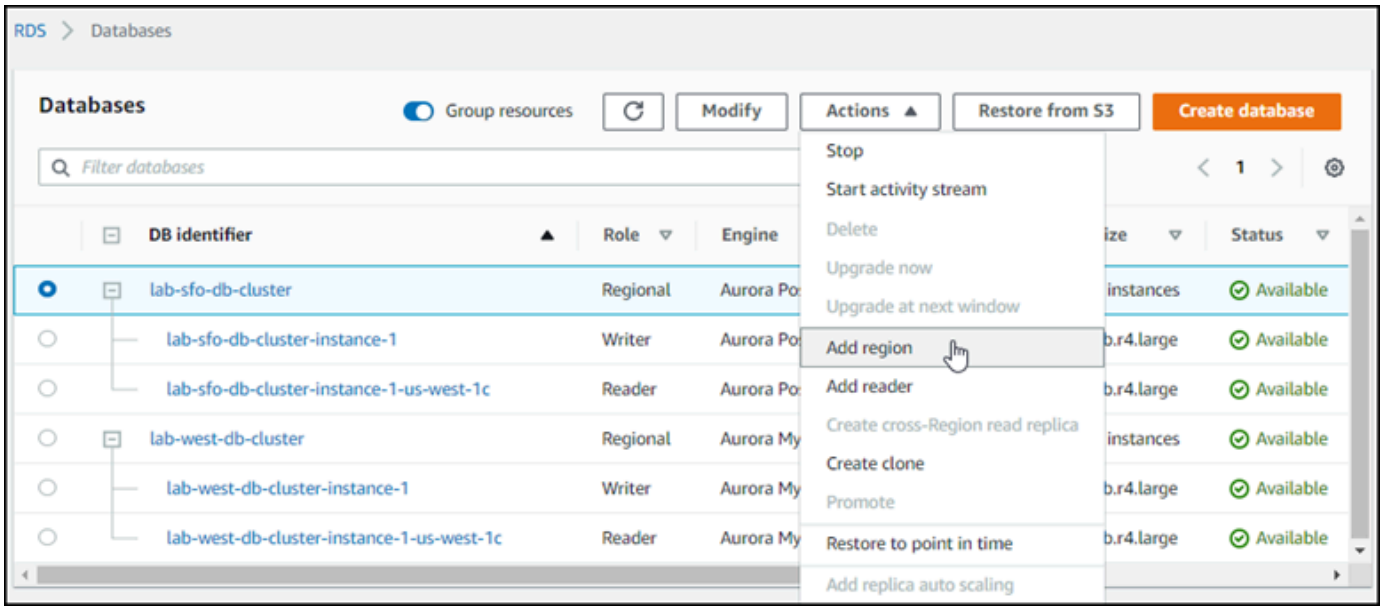
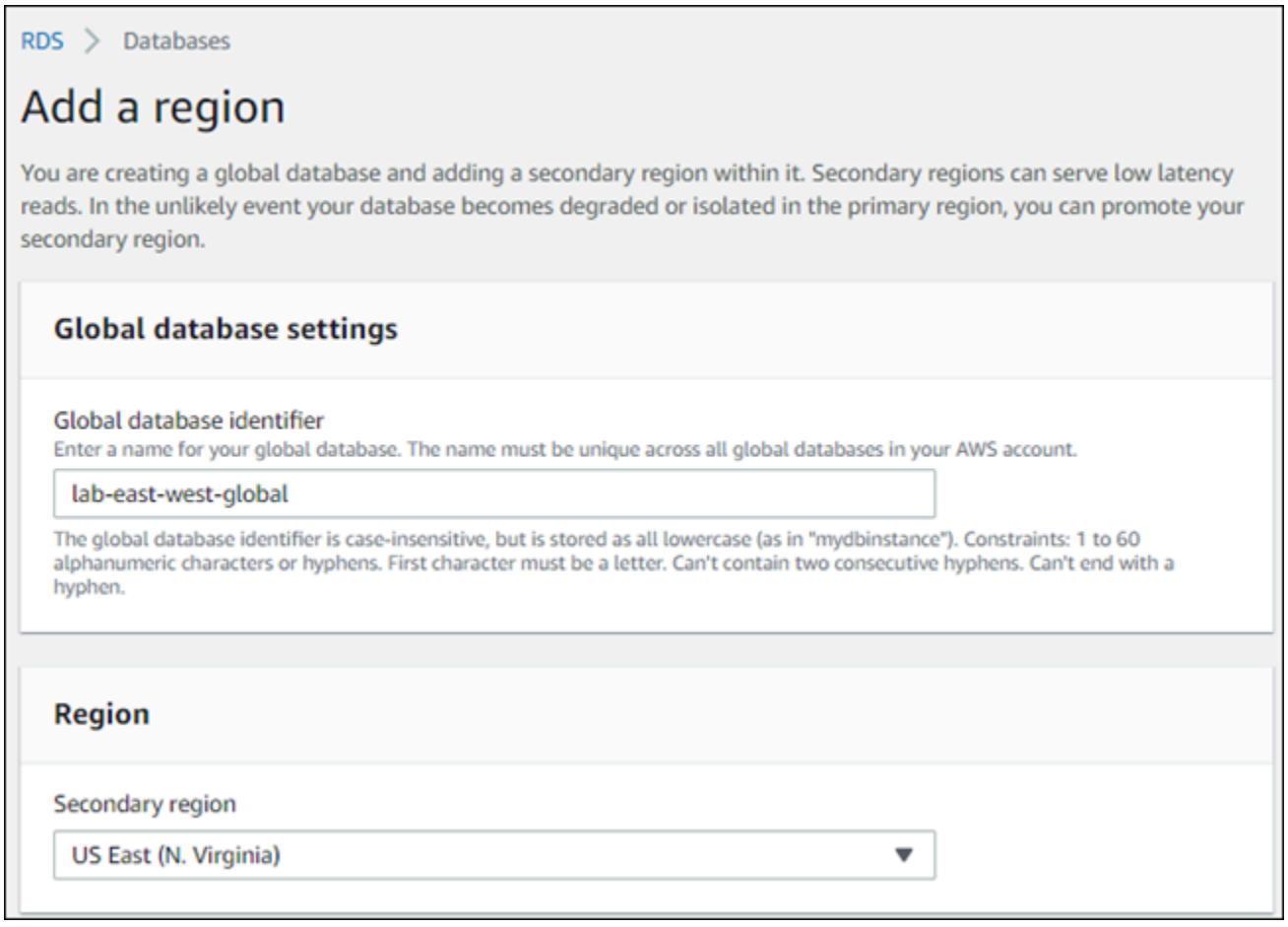
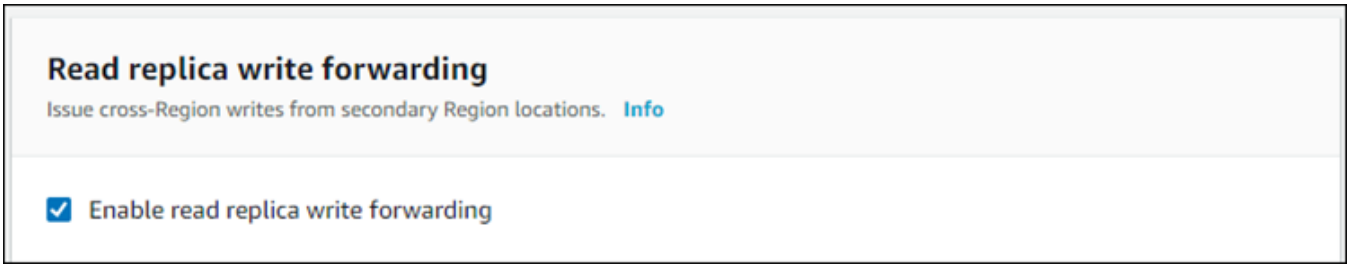
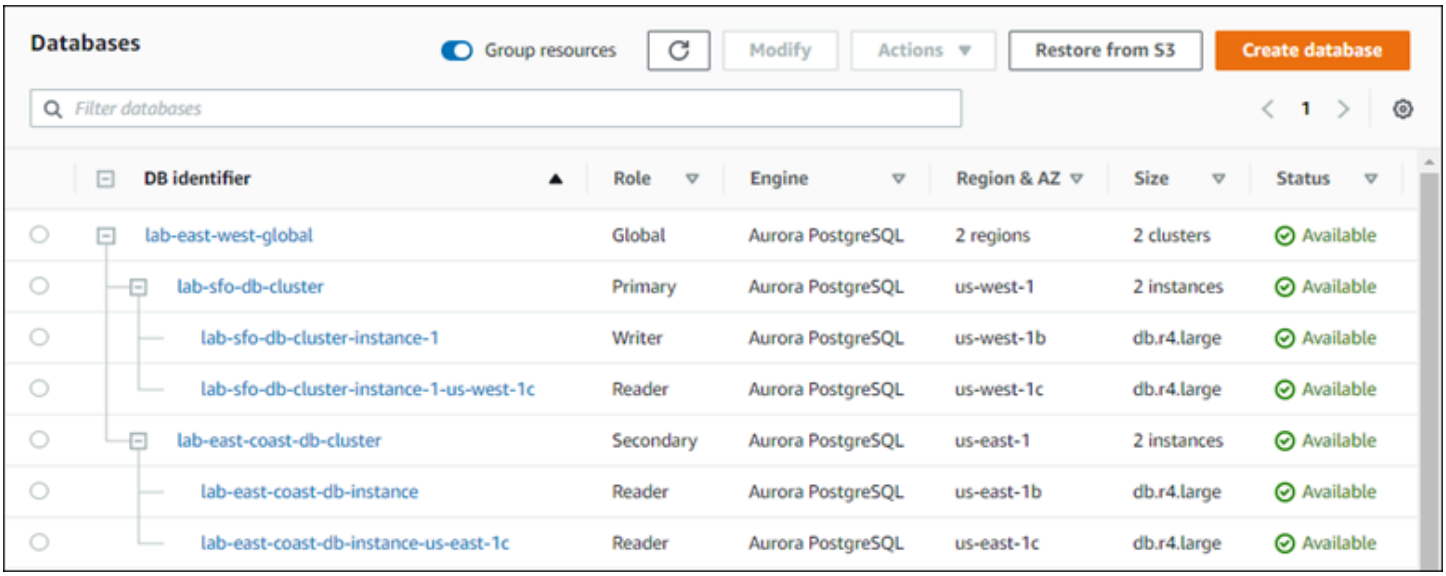
                       QUERY PLAN
-------------------------------------------------------
 Foreign Scan  (cost=100.00..101.00 rows=100 width=0)
 Single Shard Optimized
(2 rows)
```

现在，我们将 rds_aurora.limitless_explain_options 设置为包含 shard_plans 和 single_shard_optimization。我们可以在路由器和分片上查看语句的执行计划。此外，我们可以禁用 enable_seqscan 参数以强制在分片层上使用索引扫描。

```
postgres_limitless=> SET rds_aurora.limitless_explain_options = shard_plans,
 single_shard_optimization;
SET

postgres_limitless=> SET enable_seqscan = OFF;
SET

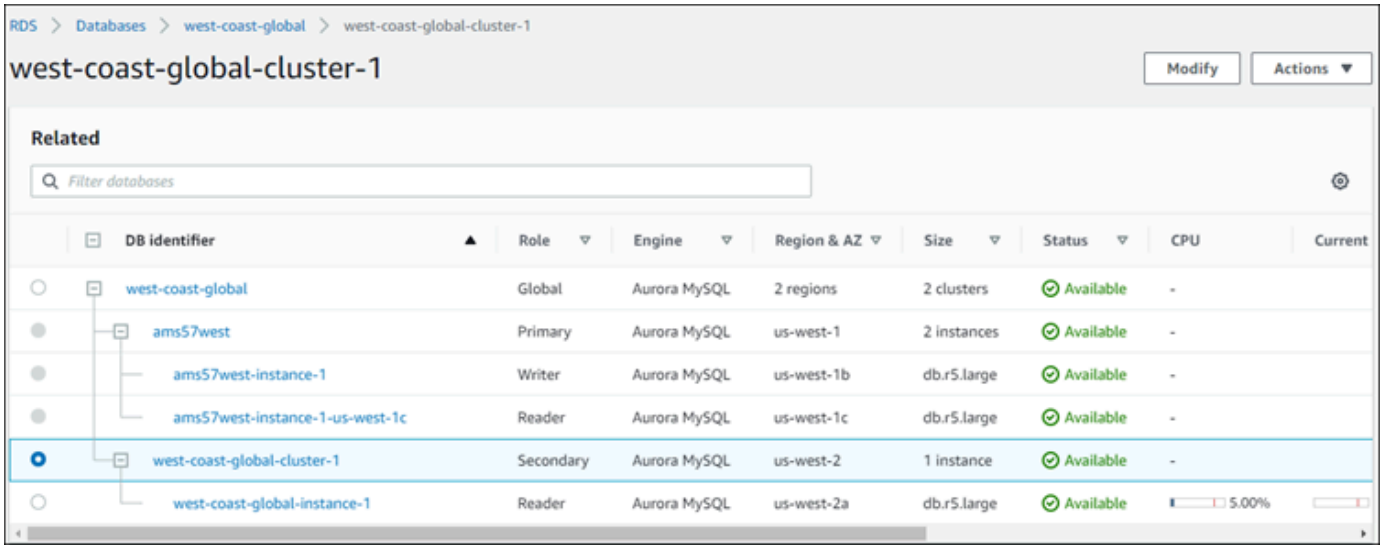
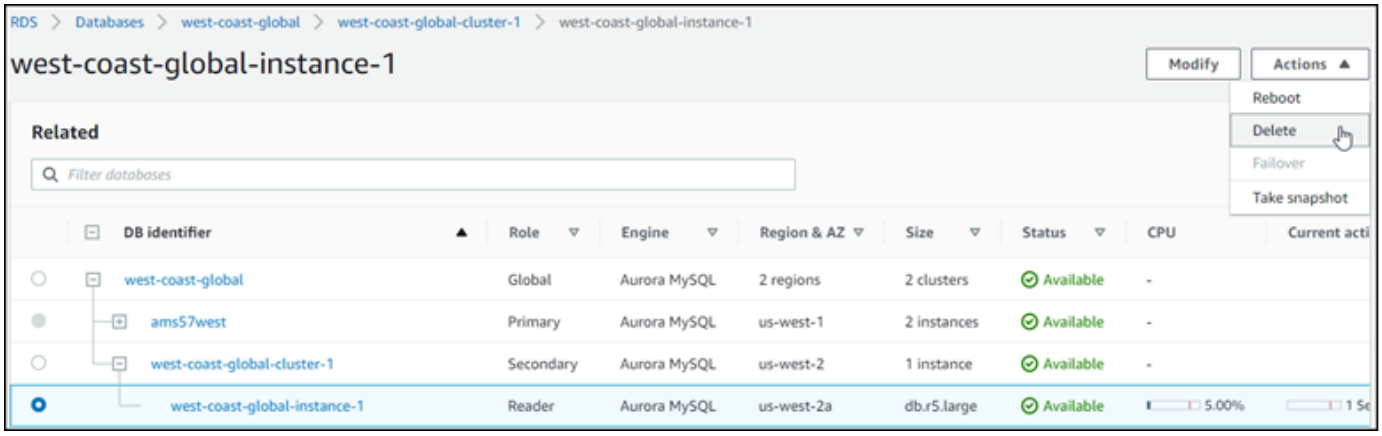
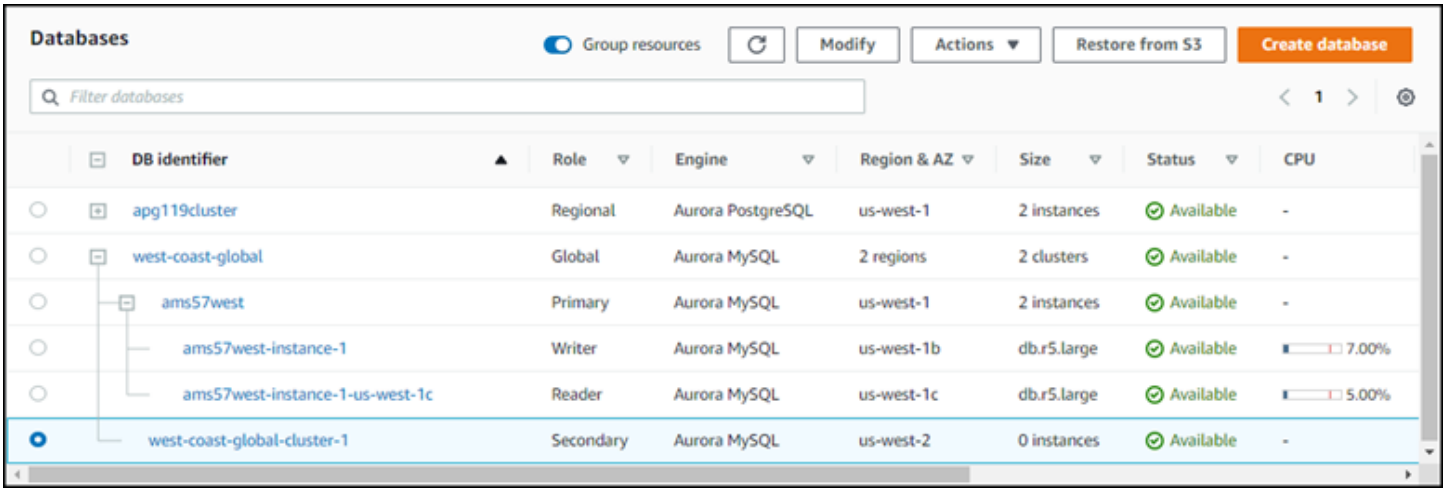
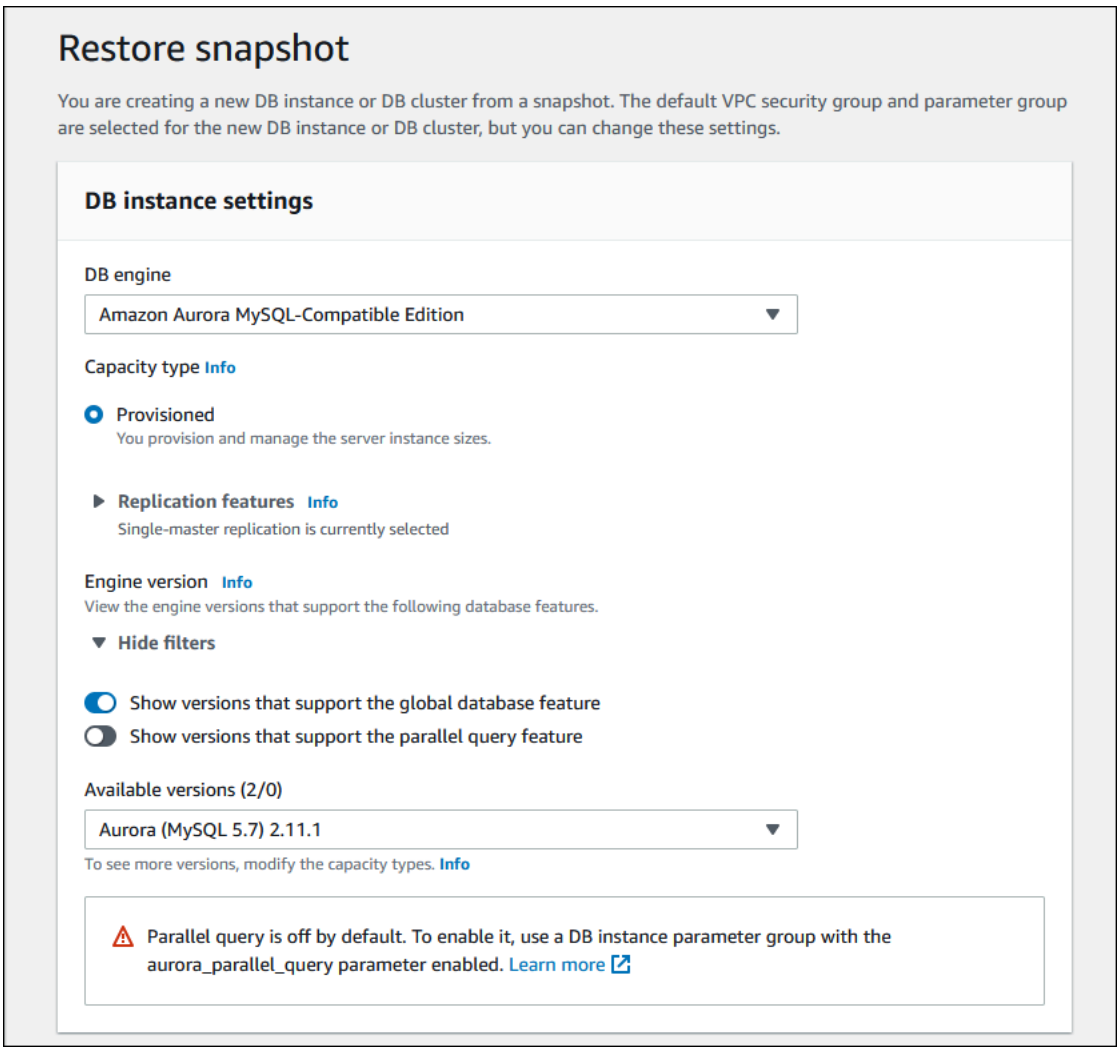
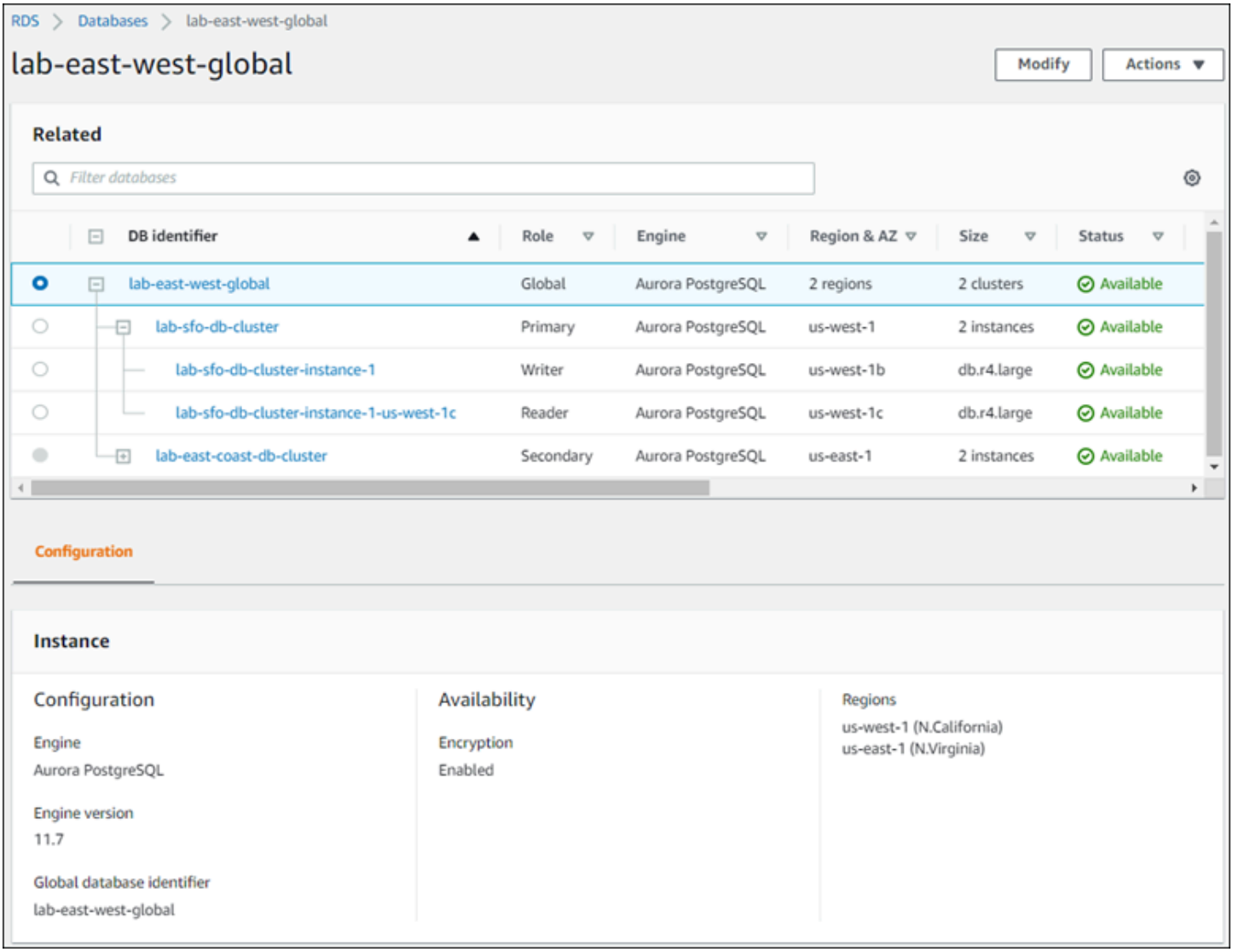
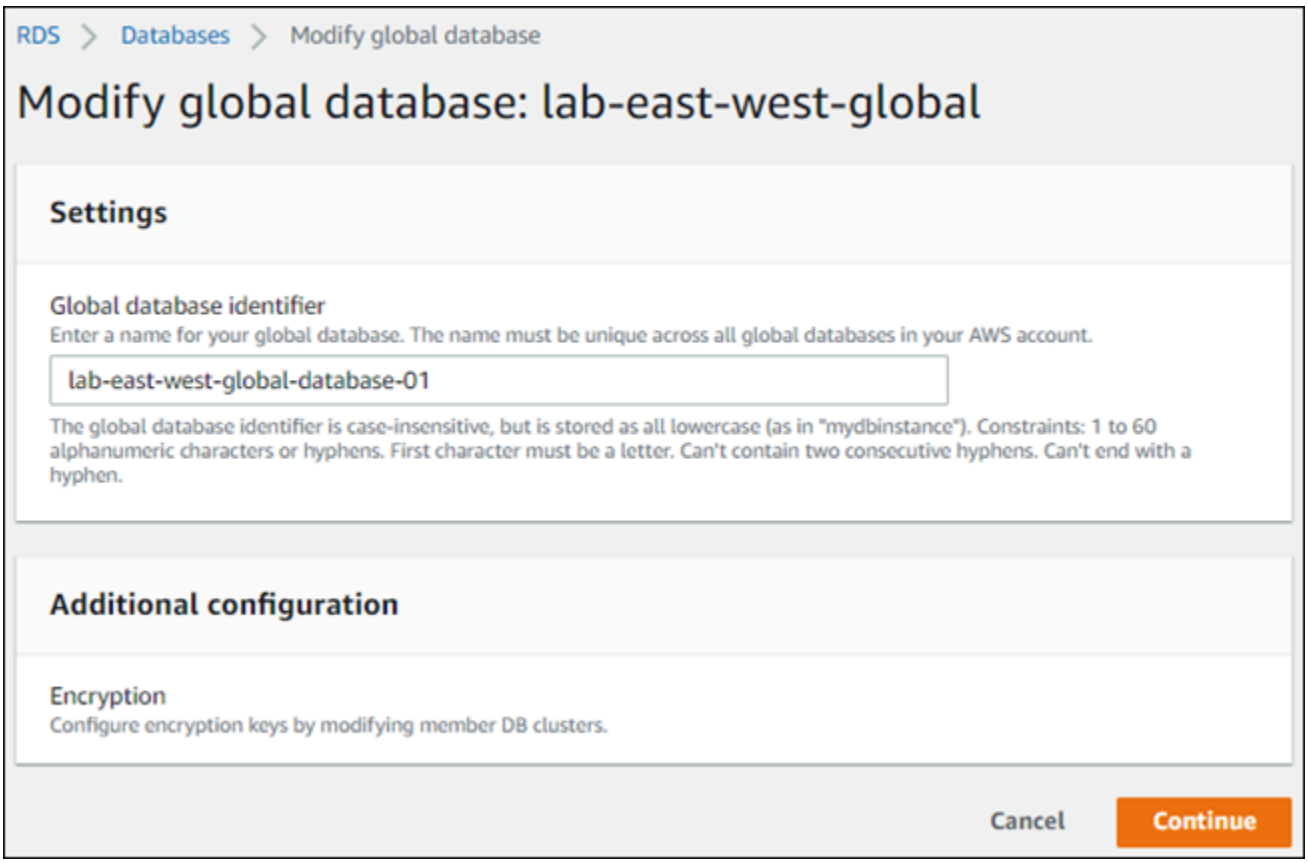
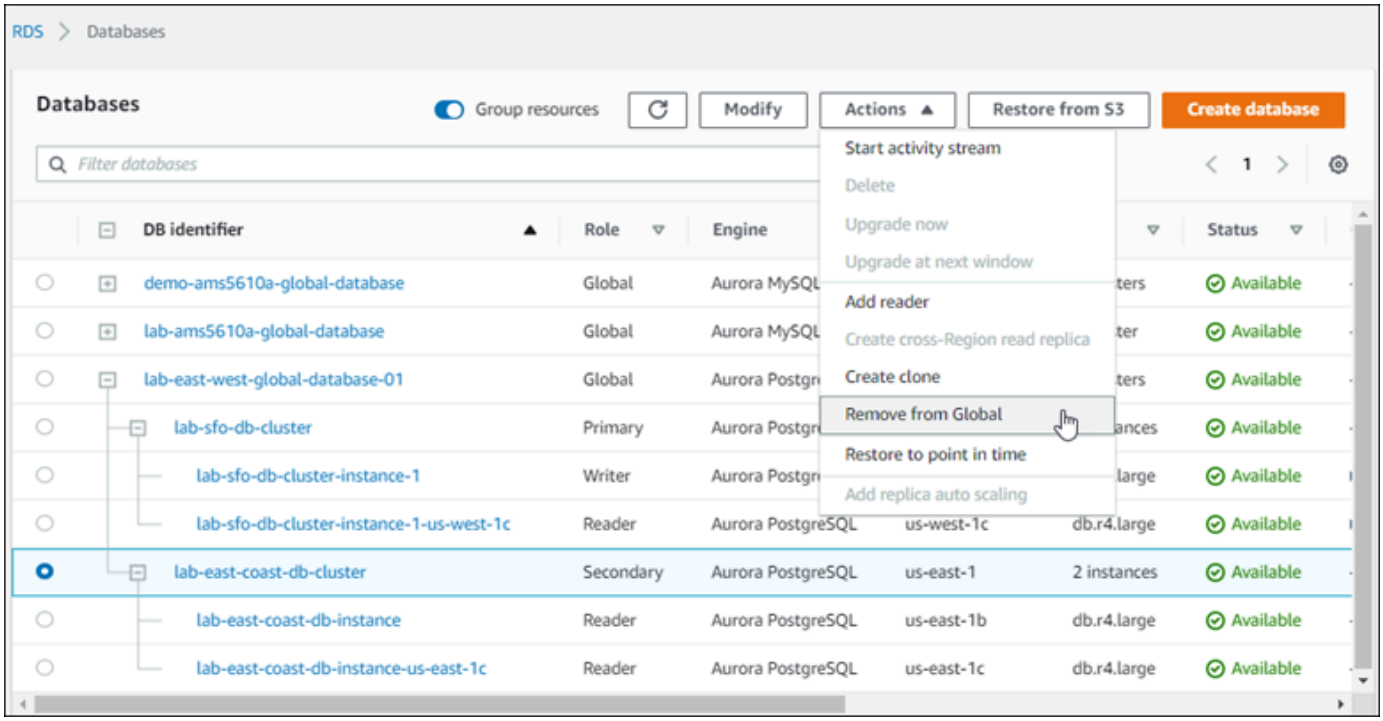
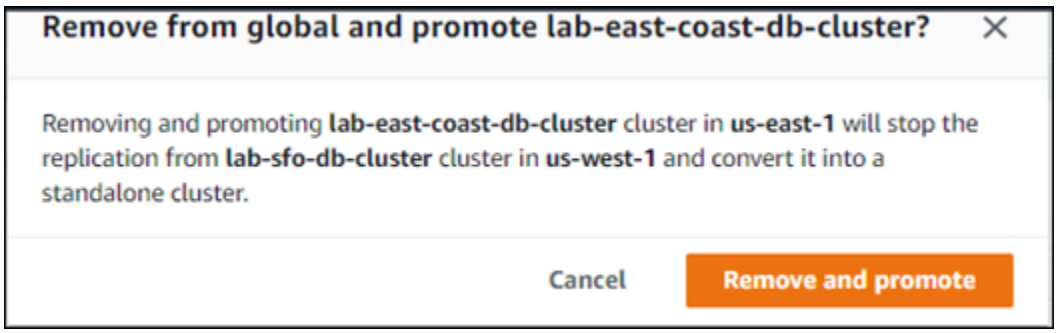
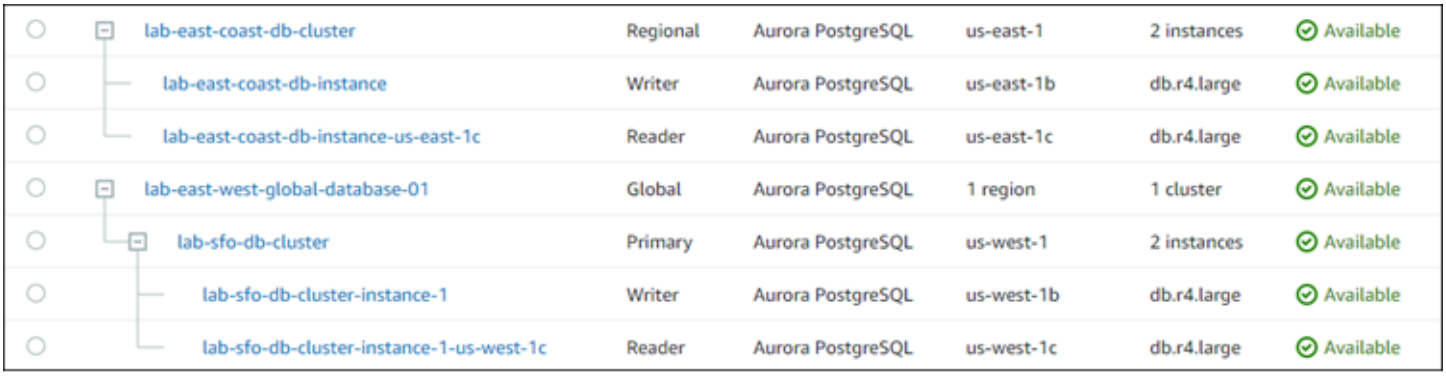
postgres_limitless=> EXPLAIN SELECT * FROM employees WHERE id = 25;

                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------
 Foreign Scan  (cost=100.00..101.00 rows=100 width=0)
   Remote Plans from Shard postgres_s4:
         Index Scan using employees_ts00287_id_idx on employees_ts00287
 employees_fs00003  (cost=0.14..8.16 rows=1 width=15)
           Index Cond: (id = 25)
 Single Shard Optimized
(5 rows)
```

有关 EXPLAIN 命令的更多信息，请参阅 PostgreSQL 文档中的 EXPLAIN。

INSERT

Aurora PostgreSQL Limitless Database 支持大多数 INSERT 命令。

PostgreSQL 没有明确的 UPSERT 命令，但它确实支持 INSERT ... ON CONFLICT 语句。

如果冲突操作有子查询或可变函数，则不支持 INSERT ... ON CONFLICT：

```
-- RANDOM is a mutable function.
INSERT INTO sharded_table VALUES (1, 100) ON CONFLICT (id) DO UPDATE SET other_id =
 RANDOM();

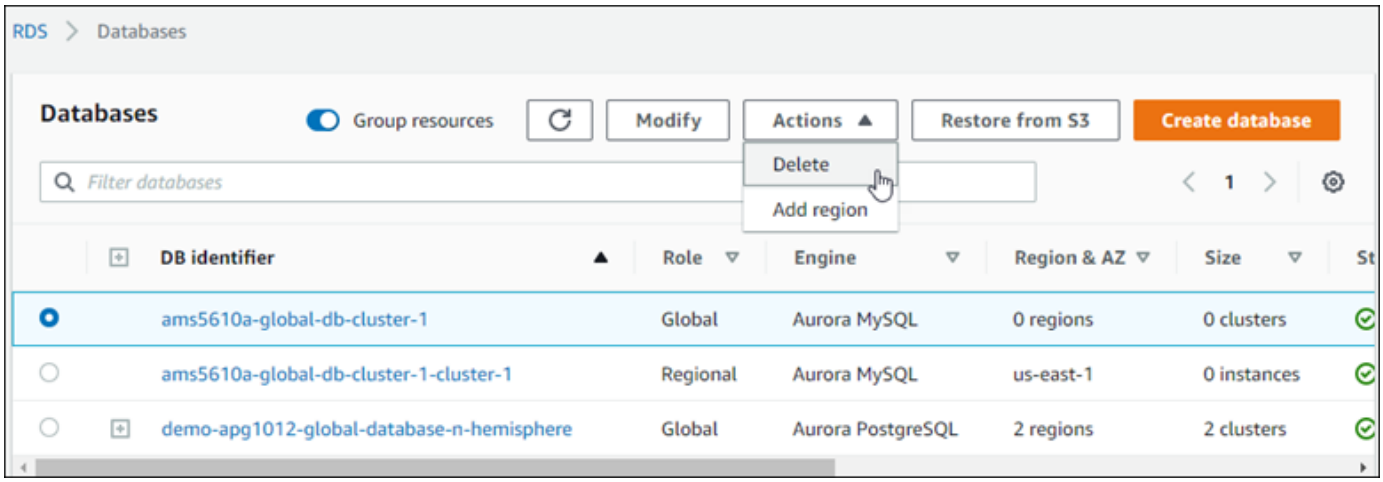
ERROR: Aurora Limitless Tables doesn't support pushdown-unsafe functions with DO UPDATE
 clauses.
```

有关 INSERT 命令的更多信息，请参阅 PostgreSQL 文档中的 INSERT。

UPDATE

不支持更新分片密钥。例如，您有一个名为 customers 的分片表，其分片键为 customer_id。以下 DML 语句会导致错误：

```
postgres_limitless=> UPDATE customers SET customer_id = 11 WHERE customer_id =1;
ERROR:  Shard key column update is not supported

postgres_limitless=> UPDATE customers SET customer_id = 11 WHERE customer_name='abc';
ERROR:  Shard key column update is not supported
```

要更新分片键，必须首先 DELETE 带有分片键的行，然后 INSERT 带有更新后分片键值的新行。

有关 UPDATE 命令的更多信息，请参阅 PostgreSQL 文档中的 Updating data。

VACUUM

您可以对分片表和引用表执行清理。Aurora PostgreSQL Limitless Database 完全支持以下 VACUUM 函数：

- VACUUM
- ANALYZE
- DISABLE_PAGE_SKIPPING

- FREEZE
- FULL
- INDEX_CLEANUP
- PARALLEL
- PROCESS_TOAST
- TRUNCATE
- VERBOSE

Aurora PostgreSQL Limitless Database 上的 VACUUM 具有以下限制：

- 不支持 pg_visibility_map 扩展。
- 不支持使用 pg_stat_all_indexes 视图检查未使用的索引。
- 未实施 pg_stat_user_indexes、pg_class 和 pg_stats 的合并视图。

有关 VACUUM 命令的更多信息，请参阅 PostgreSQL 文档中的 VACUUM。有关 Aurora PostgreSQL Limitless Database 中清理工作原理的更多信息，请参阅通过清理回收存储空间。

Aurora PostgreSQL Limitless Database 中的变量

您可以使用以下变量配置 Aurora PostgreSQL Limitless Database。

rds_aurora.limitless_active_shard_key

在查询数据库时设置单个分片键，可使所有 SELECT 和 DML 查询都将分片键作为常量谓词附加到数据库中。有关更多信息，请参阅 设置活动分片密钥。

rds_aurora.limitless_create_table_collocate_with

将此变量设置为特定的表名，以将新创建的表与该表并置。有关更多信息，请参阅 使用变量创建无限表。

rds_aurora.limitless_create_table_mode

设置表创建模式。有关更多信息，请参阅 使用变量创建无限表。

rds_aurora.limitless_create_table_shard_key

将此变量设置为列名数组以用作分片键。有关更多信息，请参阅 使用变量创建无限表。

rds_aurora.limitless_explain_options

EXPLAIN 输出中要包含的内容。有关更多信息，请参阅 EXPLAIN。

Aurora PostgreSQL Limitless Database 中的数据库集群参数

您可以使用以下数据库集群参数配置 Aurora PostgreSQL Limitless Database。

rds_aurora.limitless_adaptive_fetch_size

增强了批量预取。设置为 `true` 时，此参数允许为预取自动调整（自适应）提取大小。设置为 `false` 时，提取大小恒定不变。

rds_aurora.limitless_auto_scale_options

设置可用于在数据库分片组中添加路由器或拆分分片的选项。值可以是 `add_router`、`split_shard` 或两者兼有。

有关更多信息，请参阅向数据库分片组中添加路由器和在数据库分片组中拆分一个分片。

rds_aurora.limitless_distributed_deadlock_timeout

在检查是否存在分布式死锁之前，锁定所需的等待时间，以毫秒为单位。默认值为 `1000`（1 秒）。

有关更多信息，请参阅 Aurora PostgreSQL Limitless Database 中的分布式死锁。

rds_aurora.limitless_enable_auto_scale

允许在数据库分片组中添加路由器和拆分分片。

有关更多信息，请参阅向数据库分片组中添加路由器 和在数据库分片组中拆分一个分片。

rds_aurora.limitless_finalize_split_shard_mode

确定系统启动的分片拆分如何完成。有关更多信息，请参阅 在数据库分片组中拆分一个分片。

rds_aurora.limitless_maximum_adaptive_fetch_size

设置自适应提取大小的上限。范围为 `1–INT_MAX`。默认为 `1000`。

使用 Amazon Aurora Global Database

使用 Amazon Aurora Global Database 特征，您可以设置跨多个 AWS 区域的多个 Aurora 数据库集群。Aurora 会自动将主数据库集群中进行的所有更改同步到一个或多个辅助集群。1 个 Aurora Global Database 在 1 个区域中有 1 个主数据库集群，在不同区域中最多有 10 个辅助数据库集群。这种多区域配置可从可能影响整个 AWS 区域的罕见停机事件中快速恢复。如果多个地理位置中的所有数据有一个完整副本，那么对于从世界各地相距甚远的位置连接的应用程序，还可以实现低延迟读取操作。

主题

- Amazon Aurora Global Database 概览
- Amazon Aurora Global Database 的优势
- 区域和版本可用性
- Amazon Aurora Global Database 的限制
- 开始使用 Amazon Aurora Global Database
- 管理 Amazon Aurora Global Database
- 连接到 Amazon Aurora Global Database
- 在 Amazon Aurora Global Database 中使用写入转发
- 在 Amazon Aurora Global Database 中使用切换或失效转移
- 监控 Amazon Aurora Global Database
- 将 Amazon Aurora Global Database 与其他AWS服务结合使用
- 升级 Amazon Aurora Global Database

Amazon Aurora Global Database 概览

使用 Amazon Aurora Global Database 特征，您可以通过跨多个 AWS 区域的单个 Aurora 数据库来运行您的全局分布式应用程序。

Aurora Global Database 由 1 个写入数据的主 AWS 区域和最多 10 个只读辅助 AWS 区域组成。您可将写入操作发送到主 AWS 区域中的主数据库集群。一种极为便捷的方法是连接到 Aurora Global Database 写入器端点，该端点始终指向主数据库集群，即使在切换或失效转移到其他 AWS 区域之后也是如此。执行任何写入操作之后，Aurora 会将数据复制到使用专用基础设施的辅助 AWS 区域，延迟通常不到一秒。

下图显示了跨越两个 AWS 区域 的 Aurora Global Database 示例。



您可以通过添加一个或多个 Aurora 读取器实例以承担只读工作负载，从而独立地纵向扩展每个辅助集群。您可以对读取器实例使用 Aurora Serverless v2，以实现更精细和更灵活的扩展。

只有主集群才能执行写入操作。执行写入操作的客户端连接到 Aurora Global Database 写入器端点，该端点始终指向主集群的写入器数据库实例。如图所示，Aurora 使用集群存储卷（而不是数据库引擎）进行快速、低开销复制。要了解更多信息，请参阅Amazon Aurora 存储概述。

Aurora Global Database 专为遍布全球的应用程序而设计。多个 AWS 区域中的只读辅助数据库集群有助于优化更靠近应用程序用户的读取操作。借助写入转发特征，您还可以配置全局数据库，以使辅助集群将写入请求发送到主集群。有关更多信息，请参阅 在 Amazon Aurora Global Database 中使用写入转发。

Aurora Global Database 支持两种不同的操作来更改主数据库集群的区域，具体视情况而定：Aurora Global Database 切换和 Aurora Global Database 失效转移。

- 对于计划的操作过程（如区域轮换），请使用切换机制（以前称为“托管式计划失效转移”）。通过此特征，您可以将运行正常的 Aurora Global Database 的主集群重新放置到其辅助区域之一，而不会造成数据丢失。要了解更多信息，请参阅对 Amazon Aurora Global Database 执行切换。
- 要在主区域中发生停机后恢复 Aurora Global Database，请使用失效转移机制。通过此特征，您可以执行从主数据库集群到另一个区域的失效转移（跨区域失效转移）。要了解更多信息，请参阅执行 Aurora 全球数据库的托管式失效转移。

Amazon Aurora Global Database 的优势

使用 Aurora Global Database，您可以获得以下优势：

- 全局读取本地延迟 - 如果您在世界各地设有办事处，则可以使用 Aurora Global Database 在主 AWS 区域保持主要信息来源为最新状态。您其他区域的办事处可以访问各自区域中的信息，存在本地延迟。
- 可扩展辅助 Aurora 数据库集群 - 您可以通过向辅助 AWS 区域添加更多只读实例来扩展辅助集群。辅助集群为只读模式，因此它最多可以支持 16 个只读数据库实例，而不符合单个 Aurora 集群通常 15 个此类实例的限制。
- 从主到辅助 Aurora 数据库集群快速复制 - Aurora Global Database 执行的复制几乎不会影响主数据库集群的性能。数据库实例的资源完全专用于承担应用程序读取和写入工作负载。
- 从区域范围内的中断中恢复 - 辅助集群使您能够比传统复制解决方案更快地（RTO 更低）在新的主 AWS 区域中使用 Aurora Global Database，并且数据损失更少（RPO 更低）。

区域和版本可用性

特征可用性和支持因每个 Aurora 数据库引擎的特定版本以及 AWS 区域而异。有关 Aurora Global Database 的版本和区域可用性的更多信息，请参阅支持 Aurora 全球数据库的区域和数据库引擎。

Amazon Aurora Global Database 的限制

以下限制目前适用于 Aurora Global Database：

- Aurora Global Database 在某些 AWS 区域可用，且适用于特定 Aurora MySQL 和 Aurora PostgreSQL 版本。有关更多信息，请参阅 支持 Aurora 全球数据库的区域和数据库引擎。
- Aurora Global Database 对于支持的 Aurora 数据库实例类、AWS 区域的最大数量等有特定的配置要求。有关更多信息，请参阅 Amazon Aurora Global Database 的配置要求。
- 为了实现 Aurora MySQL 与 MySQL 5.7 兼容，Aurora Global Database 切换需要版本 2.09.1 或更高的次要版本。
- 仅当主数据库集群和辅助数据库集群具有相同的主要和次要引擎版本时，您才能对 Aurora Global Database 执行托管式跨区域切换或失效转移。根据引擎和引擎版本，补丁级别可能需要相同，也可以不同。有关允许在具有不同补丁级别的主集群和辅助集群之间进行这些操作的引擎和引擎版本列表，请参阅托管式跨区域切换和失效转移的补丁级别兼容性。如果您的引擎版本需要相同的补丁级别，您可以按照执行 Aurora 全球数据库的手动失效转移中的步骤手动执行失效转移。

- Aurora Global Database 目前不支持以下 Aurora 特征：
  - Aurora Serverless v1
  - 正在 Aurora 中回溯
- 有关在 Aurora Global Database 中使用 RDS 代理特征的限制，请参阅对全局数据库使用 RDS 代理的限制。
- 自动次要版本升级不适用于作为全局数据库一部分的 Aurora MySQL 和 Aurora PostgreSQL 集群。请注意，您可以为属于全局数据库集群的数据库实例指定该设置，但该设置无效。
- Aurora Global Database 目前不支持辅助数据库集群的 Aurora Auto Scaling。
- 要在运行 Aurora MySQL 5.7 的 Aurora Global Database 上使用数据库活动流（DAS），引擎版本必须为 2.08 或更高版本。有关 DAS 的信息，请参阅使用数据库活动流监控 Amazon Aurora。
- 以下限制目前适用于升级 Aurora Global Database：
  - 在对该 Aurora 全局数据库执行主要版本升级时，无法将自定义参数组应用于全局数据库集群。您可以在全局集群的每个区域中创建自定义参数组，然后在升级后手动将它们应用于区域集群。
  - 使用基于 Aurora MySQL 的 Aurora 全局数据库时，如果开启了 `lower_case_table_names` 参数，则无法执行从 Aurora MySQL 版本 2 到版本 3 的就地升级。有关可以使用的方法的更多信息，请参阅主要版本升级。
  - 使用 Aurora Global Database 时，如果启用了恢复点目标（RPO）特征，则无法对 Aurora PostgreSQL 数据库引擎执行主要版本升级。有关 RPO 功能的信息，请参阅 管理基于 Aurora PostgreSQL 的全局数据库的 RPO。
  - 使用 Aurora Global Database 时，您无法使用标准流程执行从 Aurora MySQL 版本 3.01 或 3.02 到 3.03 或更高版本的次要版本升级。有关要使用的过程的详细信息，请参阅通过修改引擎版本升级 Aurora MySQL。

  有关升级 Aurora Global Database 的信息，请参阅 升级 Amazon Aurora Global Database。
- 您无法单独地停止或启动全局数据库中的 Aurora 数据库集群。要了解更多信息，请参阅停止和启动 Amazon Aurora 数据库集群。
- 在某些情况下，附加到辅助 Aurora 数据库集群的 Aurora 读取器数据库实例可能会重新启动。如果主 AWS 区域的写入器数据库实例重新启动或发生失效转移，则辅助区域中的读取器数据库实例也会重新启动。随后辅助集群将不可用，直到所有读取器数据库实例与主数据库集群的写入器实例恢复同步。重启或失效转移时主集群的行为与单个非全局数据库集群的行为相同。有关更多信息，请参阅使用 Amazon Aurora 进行复制。

  在更改主数据库集群之前，请务必了解对 全局数据库的影响。要了解更多信息，请参阅从计划外停机中恢复 Amazon Aurora Global Database。

- 当 Amazon Aurora 无法访问数据库集群的 AWS KMS 密钥时，Aurora Global Database 目前不支持 `inaccessible-encryption-credentials-recoverable` 状态。在这些情况下，加密的数据库集群会直接进入最终 `inaccessible-encryption-credentials` 状态。有关这些状态的更多信息，请参阅 查看数据库集群状态。
- Secrets Manager 不支持 Aurora Global Database。向全球数据库添加区域时，您必须先关闭数据库实例的 Secrets Manager 集成。
- 使用 Aurora Global Database 的基于 Aurora PostgreSQL 的数据库集群存在以下限制：
  - 作为 Aurora Global Database 一部分的 Aurora PostgreSQL 辅助数据库集群不支持集群缓存管理。
  - 如果全局数据库的主数据库集群基于 Amazon RDS PostgreSQL 实例的副本，则无法创建辅助集群。切勿尝试使用 AWS Management Console、AWS CLI 或 `CreateDBCluster` API 操作从该集群创建辅助。尝试这样做会超时，也不会创建辅助集群。

我们建议您使用与主数据库相同版本的 Aurora 数据库引擎为全局数据库创建辅助数据库集群。有关更多信息，请参阅 创建 Amazon Aurora Global Database。

开始使用 Amazon Aurora Global Database

要开始使用 Aurora Global Database，首先需要决定您要使用哪个 Aurora 数据库引擎以及在哪些 AWS 区域中使用。只有某些 AWS 区域中的 Aurora MySQL 和 Aurora PostgreSQL 数据库引擎的特定版本才支持 Aurora Global Database。有关完整列表，请参阅 支持 Aurora 全球数据库的区域和数据库引擎。

您可以通过以下方式之一创建 Aurora Global Database：

- 使用新 Aurora 数据库集群和 Aurora 数据库实例创建新 Aurora Global Database – 您可以按照 创建 Amazon Aurora Global Database 中的步骤执行此操作。创建主 Aurora 数据库集群后，按照 将 AWS 区域 添加到 Amazon Aurora Global Database 中的步骤添加辅助 AWS 区域。
- 使用支持 Aurora Global Database 特征的现有 Aurora 数据库集群并向其添加 AWS 区域 - 只有当现有 Aurora 数据库集群使用全局兼容的数据库引擎版本时，才能执行此操作。

  检查您是否可以在选择 Aurora 数据库集群时为AWS Management Console的 Action (操作) 选择 Add region (添加区域)。如果可以，您可以将该 Aurora 数据库集群用于 Aurora 全局集群。有关更多信息，请参阅“将 AWS 区域 添加到 Amazon Aurora Global Database”。

在创建 Aurora Global Database 之前，建议您先了解所有配置要求。

主题

- Amazon Aurora Global Database 的配置要求
- 创建 Amazon Aurora Global Database
- 将 AWS 区域 添加到 Amazon Aurora Global Database
- 在辅助区域中创建无管控 Aurora 数据库集群
- 从 Aurora 或 Amazon RDS 快照创建 Amazon Aurora 全球数据库

Amazon Aurora Global Database 的配置要求

在开始使用全球数据库之前，请规划要使用的集群和实例名称、AWS 区域、实例数量和实例类。确保您的选择符合以下配置要求。

每个 Aurora Global Database 跨越至少两个 AWS 区域。主 AWS 区域 支持具有一个写入器 Aurora 数据库实例的 Aurora 数据库集群。辅助 AWS 区域 运行完全由 Aurora 副本组成的只读 Aurora 数据库集群。需要至少 1 个辅助 AWS 区域，而 Aurora Global Database 最多可拥有 10 个辅助 AWS 区域。该表列出了 Aurora 全局数据库中允许的 Aurora 数据库集群、Aurora 数据库实例和 Aurora 副本的最大数量。

描述	主 AWS 区域	辅助 AWS 区域
Aurora 数据库集群	1	10（最大值）
写入器实例	1	0
每个 Aurora 数据库集群的只读实例（Aurora 副本）	15（最大值）	16（合计）
只读实例（允许的最大值，辅助区域的指定实际数量）	15 - s	s = 辅助 AWS 区域 的总数

组成 Aurora 全局数据库的 Aurora 数据库集群具有以下特定要求：

- 数据库实例类要求 – Aurora 全局数据库需要针对内存密集型应用程序进行了优化的数据库实例类。有关内存优化的数据库实例类的信息，请参阅数据库实例类类型。建议使用 db.r5 或更高的实例类。
- AWS 区域 要求 - Aurora Global Database 需要位于一个 AWS 区域 中的主 Aurora 数据库集群，以及位于另一个区域中的至少一个辅助 Aurora 数据库集群。您最多可以创建 10 个辅助（只

读）Aurora 数据库集群，这些集群必须分别位于不同的区域中。换句话说，Aurora Global Database 中没有两个 Aurora 数据库集群可位于同一个 AWS 区域 中。

有关 Aurora 数据库引擎、引擎版本和 AWS 区域的哪些组合可以与 Aurora Global Database 结合使用的信息，请参阅支持 Aurora 全球数据库的区域和数据库引擎。

- 命名要求 - 为每个 Aurora 数据库集群选择的名称在所有 AWS 区域 中必须唯一。即使不同的 Aurora 数据库集群位于不同的区域中，也不能对它们使用相同的名称。
- Aurora Serverless v2 的容量要求：对于使用 Aurora Serverless v2 的全球数据库，主 AWS 区域中数据库集群的最小建议容量为 8 个 ACU。

在按照本节中的程序操作之前，您需要一个 AWS 账户。完成使用 Amazon Aurora 所需的设置任务。有关更多信息，请参阅“为 Amazon Aurora 设置环境”。您还需要完成创建任何 Aurora 数据库集群的其他准备步骤。要了解更多信息，请参阅创建 Amazon Aurora 数据库集群。

准备好设置全球数据库时，请参阅创建 Amazon Aurora Global Database来了解创建所有必要资源的过程。也可以按照将 AWS 区域 添加到 Amazon Aurora Global Database中的过程，使用现有 Aurora 集群作为主集群来创建全球数据库。

创建 Amazon Aurora Global Database

要使用 AWS Management Console、AWS CLI 或 RDS API 来创建 Aurora 全球数据库及其关联资源，请使用以下步骤。

> **Note**
>
> 如果您有一个在全局兼容型 Aurora 数据库引擎上运行的现有 Aurora 数据库集群，则可以使用此过程的简短形式。在这种情况下，可以向现有数据库集群添加另一个 AWS 区域，来创建 Aurora 全球数据库。为此，请参阅 将 AWS 区域 添加到 Amazon Aurora Global Database。

控制台

创建 Aurora Global Database 的步骤首先是登录支持 Aurora Global Database 特征的 AWS 区域。有关完整列表，请参阅支持 Aurora 全球数据库的区域和数据库引擎。

下列步骤之一用于根据 Amazon VPC 为 Aurora 数据库集群选择 Virtual Private Cloud (VPC)。要使用您自己的 VPC，我们建议您提前创建它，使其可供您选择。同时，创建任意相关子网，并根据需要创建子网组和安全组。要了解如何操作，请参阅教程：创建 VPC 以用于数据库集群（仅限 IPv4）。

有关创建 Aurora 数据库集群的一般信息，请参阅 创建 Amazon Aurora 数据库集群。

创建 Aurora 全局数据库

1. 登录AWS Management Console并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 选择创建数据库。在 Create database (创建数据库) 页面上，执行以下操作：
   - 为数据库创建方法选择 Standard create (标准创建)。（请勿选择 Easy create (轻松创建)。）
   - 对于引擎选项部分中的 `Engine type`，选择适用的引擎类型，即 Aurora（MySQL 兼容）或 Aurora（PostgreSQL 兼容）。
3. 使用以下过程中的步骤继续创建 Aurora 全局数据库。

使用 Aurora MySQL 创建全局数据库

以下步骤适用于 Aurora MySQL 的所有版本。

使用 Aurora MySQL 创建 Aurora 全局数据库

填充 Create database (创建数据库) 页面。

1. 对于 Engine options（引擎选项），请选择以下设置：
   - 对于 Engine version（引擎版本），选择您要用于 Aurora 全局数据库的 Aurora MySQL 版本。
2. 对于 Templates (模板)，选择 Production (生产)。或者您可以根据自己的使用案例选择开发/测试。请勿在生产环境做使用开发/测试。
3. 对于 Settings (设置)，请执行以下操作：
   a. 为数据库集群标识符输入有意义的名称。完成 Aurora 全局数据库的创建后，此名称将标识主数据库集群。
   b. 为数据库实例的 `admin` 用户账户输入您自己的密码，或者让 Aurora 为您生成密码。如果选择自动生成密码，则可以选择复制该密码。



4. 对于 DB instance class (数据库实例类)，选择 `db.r5.large` 或其他经内存优化的数据库实例类。建议使用 db.r5 或更高的实例类。

5. 为了确保可用性和持久性，我们建议您选择让 Aurora 在您的其他可用区 (AZ) 中创建 Aurora 副本。如果您现在不创建 Aurora 副本，则将需要稍后创建。



6. 对于连接，选择基于 Amazon VPC 的 Virtual Private Cloud (VPC)（定义了此数据库实例的虚拟网络环境）。您可以选择默认值以简化此任务。

7. 完成 Database authentication（数据库身份验证）设置。为了简化此流程，您可以现在选择 Password authentication（密码身份验证），随后再设置 AWS Identity and Access Management (IAM)。
8. 对于 Additional configuration (其他配置)，执行以下操作：

   a. 为 Initial database name (初始数据库名称) 输入相应的名称，以便为此集群创建主 Aurora 数据库实例。这是 Aurora 主数据库集群的写入器节点。

      将数据库集群参数组和数据库参数组保留为默认选中状态，除非您有自己想要使用的自定义参数组。

   b. 取消选中 Enable backtrack (启用回溯) 复选框（如果已选中）。Aurora Global Database 不支持回溯。否则，您需要接受 Additional configuration（其他配置）的其他默认设置。
9. 选择创建数据库。

   Aurora 可能需要几分钟才能完成 Aurora 数据库实例、其 Aurora 副本和 Aurora 数据库集群的创建流程。您可以通过 Aurora 数据库集群状态来判断其何时可以用作 Aurora 全局数据库中的主数据库集群。当使用准备就绪时，数据库集群及其写入器和副本节点都显示为 Available（可用）状态，如下所示。



当主数据库集群可用时，请通过向其添加辅助集群来创建 Aurora 全局数据库。为此，请按照 将 AWS 区域 添加到 Amazon Aurora Global Database 中的步骤操作。

使用 Aurora PostgreSQL 创建全局数据库

使用 Aurora PostgreSQL 创建 Aurora 全局数据库

填充 Create database (创建数据库) 页面。

1. 对于 Engine options（引擎选项），请选择以下设置：

- 对于 Engine version（引擎版本），选择您要用于 Aurora 全局数据库的 Aurora PostgreSQL 版本。

2. 对于 Templates (模板)，选择 Production (生产)。或者您可以选择开发/测试（如果适用）。请勿在生产环境做使用开发/测试。
3. 对于 Settings (设置)，请执行以下操作：

   a. 为数据库集群标识符输入有意义的名称。完成 Aurora 全局数据库的创建后，此名称将标识主数据库集群。

   b. 为数据库集群的默认管理员账户输入您自己的密码，或者让 Aurora 为您生成密码。如果选择自动生成密码，则可以选择复制该密码。



4. 对于 DB instance class (数据库实例类)，选择 `db.r5.large` 或其他经内存优化的数据库实例类。建议使用 db.r5 或更高的实例类。

5. 为了确保可用性和持久性，我们建议您选择让 Aurora 在您的其他可用区中创建 Aurora 副本。如果您现在不创建 Aurora 副本，则将需要稍后创建。
6. 对于连接，选择基于 Amazon VPC 的 Virtual Private Cloud (VPC)（定义了此数据库实例的虚拟网络环境）。您可以选择默认值以简化此任务。
7. （可选）完成 Database authentication（数据库身份验证）设置。密码身份验证始终处于启用状态。为了简化此过程，您可以跳过此部分，随后再设置 IAM 或密码和 Kerberos 身份验证。
8. 对于 Additional configuration (其他配置)，执行以下操作：

   a. 为 Initial database name (初始数据库名称) 输入相应的名称，以便为此集群创建主 Aurora 数据库实例。这是 Aurora 主数据库集群的写入器节点。

      将数据库集群参数组和数据库参数组保留为默认选中状态，除非您有自己想要使用的自定义参数组。

   b. 接受 Additional configuration（其他配置）的所有其他默认，例如加密、日志导出等。
9. 选择创建数据库。

   Aurora 可能需要几分钟才能完成 Aurora 数据库实例、其 Aurora 副本和 Aurora 数据库集群的创建流程。当集群做好使用准备时，Aurora 数据库集群及其写入器和副本节点都显示 Available (可用) 状态。添加辅助数据库集群后，这将成为 Aurora 全局数据库的主数据库集群。



当主数据库集群可用时，按照 将 AWS 区域 添加到 Amazon Aurora Global Database 中的步骤创建一个或多个辅助集群。

AWS CLI

以下程序中的 AWS CLI 命令可完成以下任务：

1. 创建一个 Aurora 全局数据库，为其命名并指定计划使用的 Aurora 数据库引擎类型。
2. 为 Aurora 全局数据库创建 Aurora 数据库集群。
3. 为集群创建 Aurora 数据库实例。这是全局数据库的主要 Aurora 数据库集群。
4. 为 Aurora 数据库集群创建第二个数据库实例。这是用于完成 Aurora 数据库集群的读取器。
5. 按照 将 AWS 区域 添加到 Amazon Aurora Global Database 中的步骤，在另一个区域中创建第二个 Aurora 数据库集群，然后将其添加到您的 Aurora 全局数据库。

按照 Aurora 数据库引擎的程序进行操作。

使用 Aurora MySQL 创建全局数据库

使用 Aurora MySQL 创建 Aurora 全局数据库

1. 使用 `create-global-cluster` CLI 命令，传递 AWS 区域 的名称、Aurora 数据库引擎和版本。

   对于 Linux、macOS 或 Unix：

   ```
   aws rds create-global-cluster --region primary_region \
       --global-cluster-identifier global_database_id \
       --engine aurora-mysql \
       --engine-version version # optional
   ```

   对于 Windows：

   ```
   aws rds create-global-cluster ^
       --global-cluster-identifier global_database_id ^
       --engine aurora-mysql ^
       --engine-version version # optional
   ```

   这将创建一个“空” Aurora 全局数据库，只有名称（标识符）和 Aurora 数据库引擎。Aurora 全局数据库可能需要过几分钟才能使用。在进入下一步之前，请使用 `describe-global-clusters` CLI 命令查看它是否可用。

```
aws rds describe-global-clusters --region primary_region --global-cluster-
identifier global_database_id
```

当 Aurora 全局数据库可用时，您可以创建其主 Aurora 数据库集群。

2. 要创建主 Aurora 数据库集群，请使用 create-db-cluster CLI 命令。使用 --global-cluster-identifier 参数以包含 Aurora 全球数据库的名称。

   对于 Linux、macOS 或 Unix：

```
aws rds create-db-cluster \
  --region primary_region \
  --db-cluster-identifier primary_db_cluster_id \
  --master-username userid \
  --master-user-password password \
  --engine aurora-mysql \
  --engine-version version \
  --global-cluster-identifier global_database_id
```

   对于 Windows：

```
aws rds create-db-cluster ^
  --region primary_region ^
  --db-cluster-identifier primary_db_cluster_id ^
  --master-username userid ^
  --master-user-password password ^
  --engine aurora-mysql ^
  --engine-version version ^
  --global-cluster-identifier global_database_id
```

   使用 describe-db-clusters AWS CLI 命令确认 Aurora 数据库集群已准备就绪。要选出特定 Aurora 数据库集群，请使用 --db-cluster-identifier 参数。或者，您可以在命令中保留 Aurora 数据库集群名称，以获取有关指定区域中所有 Aurora 数据库集群的详细信息。

```
aws rds describe-db-clusters --region primary_region --db-cluster-
identifier primary_db_cluster_id
```

   当集群的响应显示 "Status": "available" 时，它就可随时使用了。

3. 为主 Aurora 数据库集群创建数据库实例。为此，使用 create-db-instance CLI 命令。为命令提供 Aurora 数据库集群的名称，然后指定实例的配置详细信息。您不需要在命令中传递 --master-username 和 --master-user-password 参数，因为它会从 Aurora 数据库集群获取这些参数。

   对于 --db-instance-class，您只能使用来自经内存优化的类的内容，例如 db.r5.large。建议使用 db.r5 或更高的实例类。有关这些类的信息，请参阅数据库实例类类型。

   对于 Linux、macOS 或 Unix：

```
aws rds create-db-instance \
  --db-cluster-identifier primary_db_cluster_id \
  --db-instance-class instance_class \
  --db-instance-identifier db_instance_id \
  --engine aurora-mysql \
  --engine-version version \
  --region primary_region
```

   对于 Windows：

```
aws rds create-db-instance ^
  --db-cluster-identifier primary_db_cluster_id ^
  --db-instance-class instance_class ^
  --db-instance-identifier db_instance_id ^
  --engine aurora-mysql ^
  --engine-version version ^
  --region primary_region
```

   create-db-instance 操作可能需要一些时间才能完成。继续操作之前，请检查状态以查看 Aurora 数据库实例是否可用。

```
aws rds describe-db-clusters --db-cluster-identifier primary_db_cluster_id
```

   当命令返回 available 状态时，您可以为主数据库集群创建另一个 Aurora 数据库实例。这是 Aurora 数据库集群的读取器实例（Aurora 副本）。

4. 要为集群创建另一个 Aurora 数据库实例，请使用 create-db-instance CLI 命令。

   对于 Linux、macOS 或 Unix：

```
aws rds create-db-instance \
  --db-cluster-identifier primary_db_cluster_id \
  --db-instance-class instance_class \
  --db-instance-identifier replica_db_instance_id \
  --engine aurora-mysql
```

对于 Windows：

```
aws rds create-db-instance ^
  --db-cluster-identifier primary_db_cluster_id ^
  --db-instance-class instance_class ^
  --db-instance-identifier replica_db_instance_id ^
  --engine aurora-mysql
```

当数据库实例可用时，从写入器节点到副本的复制开始。继续操作之前，请使用 describe-db-instances CLI 命令检查数据库实例是否可用。

此时，您有一个 Aurora 全局数据库，其主 Aurora 数据库集群包含写入器数据库实例和 Aurora 副本。现在，您可以在不同区域中添加只读 Aurora 数据库集群来完成 Aurora 全局数据库。为此，请按照将 AWS 区域 添加到 Amazon Aurora Global Database中的步骤进行操作。

使用 Aurora PostgreSQL 创建全局数据库

使用以下命令为 Aurora 全局数据库创建 Aurora 对象时，每个对象可能需要过几分钟后才能变为可用状态。我们建议在完成任何指定命令后，检查特定 Aurora 对象的状态以确保其状态为“可用”。

为此，使用 describe-global-clusters CLI 命令。

```
aws rds describe-global-clusters --region primary_region
    --global-cluster-identifier global_database_id
```

使用 Aurora PostgreSQL 创建 Aurora 全局数据库

1. 使用 create-global-cluster CLI 命令。

   对于 Linux、macOS 或 Unix：

   ```
   aws rds create-global-cluster --region primary_region \
       --global-cluster-identifier global_database_id \
   ```

```
    --engine aurora-postgresql \
    --engine-version version # optional
```

对于 Windows：

```
aws rds create-global-cluster ^
    --global-cluster-identifier global_database_id ^
    --engine aurora-postgresql ^
    --engine-version version # optional
```

当 Aurora 全局数据库可用时，您可以创建其主 Aurora 数据库集群。

2. 要创建主 Aurora 数据库集群，请使用 create-db-cluster CLI 命令。使用 `--global-cluster-identifier` 参数以包含 Aurora 全球数据库的名称。

   对于 Linux、macOS 或 Unix：

   ```
   aws rds create-db-cluster \
     --region primary_region \
     --db-cluster-identifier primary_db_cluster_id \
     --master-username userid \
     --master-user-password password \
     --engine aurora-postgresql \
     --engine-version version \
     --global-cluster-identifier global_database_id
   ```

   对于 Windows：

   ```
   aws rds create-db-cluster ^
     --region primary_region ^
     --db-cluster-identifier primary_db_cluster_id ^
     --master-username userid ^
     --master-user-password password ^
     --engine aurora-postgresql ^
     --engine-version version ^
     --global-cluster-identifier global_database_id
   ```

   检查 Aurora 数据库集群是否准备就绪。如果对 Aurora 数据库集群运行以下命令后的响应显示 `"Status": "available"`，您可以继续。

```
aws rds describe-db-clusters --region primary_region --db-cluster-
identifier primary_db_cluster_id
```

3. 为主 Aurora 数据库集群创建数据库实例。为此，使用 create-db-instance CLI 命令。

   使用 --db-cluster-identifier 参数传递 Aurora 数据库集群的名称。

   您不需要在命令中传递 --master-username 和 --master-user-password 参数，因为它会从 Aurora 数据库集群获取这些参数。

   对于 --db-instance-class，您只能使用来自经内存优化的类的内容，例如 db.r5.large。建议使用 db.r5 或更高的实例类。有关这些类的信息，请参阅数据库实例类类型。

   对于 Linux、macOS 或 Unix：

```
aws rds create-db-instance \
  --db-cluster-identifier primary_db_cluster_id \
  --db-instance-class instance_class \
  --db-instance-identifier db_instance_id \
  --engine aurora-postgresql \
  --engine-version version \
  --region primary_region
```

   对于 Windows：

```
aws rds create-db-instance ^
  --db-cluster-identifier primary_db_cluster_id ^
  --db-instance-class instance_class ^
  --db-instance-identifier db_instance_id ^
  --engine aurora-postgresql ^
  --engine-version version ^
  --region primary_region
```

4. 继续操作之前，请检查 Aurora 数据库实例的状态。

```
aws rds describe-db-clusters --db-cluster-identifier primary_db_cluster_id
```

   如果响应显示 Aurora 数据库实例状态为 available，则可以为主数据库集群创建另一个 Aurora 数据库实例。

5. 要为 Aurora 数据库集群创建 Aurora 副本，请使用 create-db-instance CLI 命令。

对于 Linux、macOS 或 Unix：

```
aws rds create-db-instance \
  --db-cluster-identifier primary_db_cluster_id \
  --db-instance-class instance_class \
  --db-instance-identifier replica_db_instance_id \
  --engine aurora-postgresql
```

对于 Windows：

```
aws rds create-db-instance ^
  --db-cluster-identifier primary_db_cluster_id ^
  --db-instance-class instance_class ^
  --db-instance-identifier replica_db_instance_id ^
  --engine aurora-postgresql
```

当数据库实例可用时，从写入器节点到副本的复制开始。继续操作之前，请使用 describe-db-instances CLI 命令检查数据库实例是否可用。

您的 Aurora 全局数据库存在，但它只有主区域，其中包含由写入器数据库实例和 Aurora 副本组成的 Aurora 数据库集群。现在，您可以在不同区域中添加只读 Aurora 数据库集群来完成 Aurora 全局数据库。为此，请按照将 AWS 区域 添加到 Amazon Aurora Global Database中的步骤进行操作。

RDS API

要使用 RDS API 创建 Aurora 全局数据库，请运行 CreateGlobalCluster 操作。

将 AWS 区域 添加到 Amazon Aurora Global Database

可以使用以下过程向现有的全球数据库添加其他辅助集群。也可以通过使用此过程添加第一个辅助 AWS 区域，从独立的 Aurora 数据库集群创建全球数据库。

Aurora Global Database 至少需要一个与主 Aurora 数据库集群在不同 AWS 区域 中的辅助 Aurora 数据库集群。您最多可以将 10 个辅助数据库集群挂载到 Aurora Global Database。对每个新的辅助数据库集群重复以下过程。对于添加到 Aurora 全局数据库的每个辅助数据库集群，将主数据库集群的允许 Aurora 副本数量减少一个。

例如，如果您的 Aurora Global Database 有 10 个辅助区域，则主数据库集群只能有 5 个（而不是 15 个）Aurora 副本。有关更多信息，请参阅 Amazon Aurora Global Database 的配置要求。

主数据库集群中的 Aurora 副本（读取器实例）的数量决定了您可以添加的备用数据库集群的数量。主数据库集群中的读取器实例数加上辅助集群数不可超过 15 个。例如，如果有 1 个备用集群且主数据库集群中有 14 个读取器实例，则无法向全局数据库添加辅助集群。

> **Note**
>
> 对于 Aurora MySQL 版本 3，当您创建辅助集群时，确保 `lower_case_table_names` 的值匹配主集群中的值。此设置是影响服务器处理标识符区分大小写方式的数据库参数。有关数据库参数的更多信息，请参阅 Amazon Aurora 的参数组。
>
> 我们建议您在创建辅助集群时，对主集群和辅助集群使用相同的数据库引擎版本。如有必要，请将主集群升级为与辅助集群相同的版本。有关更多信息，请参阅 托管式跨区域切换和失效转移的补丁级别兼容性。

控制台

将 AWS 区域 添加到 Aurora Global Database

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在 AWS Management Console 的导航窗格中，选择 Databases (数据库)。
3. 选择需要辅助 Aurora 数据库集群的 Aurora 全局数据库。确保主 Aurora 数据库集群为 `Available`。
4. 对于操作，选择添加 AWS 区域。



5. 在 Add a region（添加区域）页面上，选择辅助 AWS 区域。

   您不能为同一个 Aurora Global Database 选择已有辅助 Aurora 数据库集群的 AWS 区域。此外，该区域也不能是主 Aurora 数据库集群所在的同一个区域。

   > **Note**
   >
   > 只有在辅助区域中开启控制 Babelfish 首选项的参数后，适用于 Aurora PostgreSQL 的 Babelfish 全球数据库才能在这些区域中运行。有关更多信息，请参阅 Babelfish 的数据库集群参数组设置。



6. 在新AWS区域中填写辅助 Aurora 集群的其余字段。这些配置选项与任何 Aurora 数据库集群实例的配置选项相同，但以下选项仅适用基于 Aurora MySQL 的 Aurora 全局数据库：

   - 启用只读副本写入转发 – 此可选设置可以让 Aurora 全局数据库的辅助数据库集群将写入操作转发到主群集。有关更多信息，请参阅“在 Amazon Aurora Global Database 中使用写入转发”。



7. 选择添加 AWS 区域。

完成将区域添加到 Aurora Global Database 的操作后，它会出现在屏幕截图中所示的AWS Management Console的 Databases (数据库) 列表中。



AWS CLI

将辅助 AWS 区域 添加到 Aurora Global Database

要使用 CLI 向全球数据库添加辅助集群，您必须已经具有全局集群容器对象。如果您尚未运行 `create-global-cluster` 命令，请参阅创建 Amazon Aurora Global Database中的 CLI 过程。

1. 使用带有 Aurora 全局数据库名称 (`create-db-cluster`) 的 `--global-cluster-identifier` CLI 命令。对于其他参数，请执行以下操作：
2. 对于 `--region`，请选择与 Aurora 主区域不同的 AWS 区域。
3. 为 `--engine` 和 `--engine-version` 参数选择特定的值。这些值与 Aurora 全局数据库中的主 Aurora 数据库集群的值相同。
4. 对于加密集群，请将主 AWS 区域 指定为 `--source-region` 以进行加密。

以下示例创建了一个新 Aurora 数据库集群，并将其作为只读辅助 Aurora 数据库集群挂载到 Aurora 全局数据库。在最后一步中，Aurora 数据库实例将添加到新 Aurora 数据库集群。

对于 Linux、macOS 或 Unix：

```
aws rds --region secondary_region \
  create-db-cluster \
    --db-cluster-identifier secondary_cluster_id \
    --global-cluster-identifier global_database_id \
    --engine aurora-mysql | aurora-postgresql \
    --engine-version version

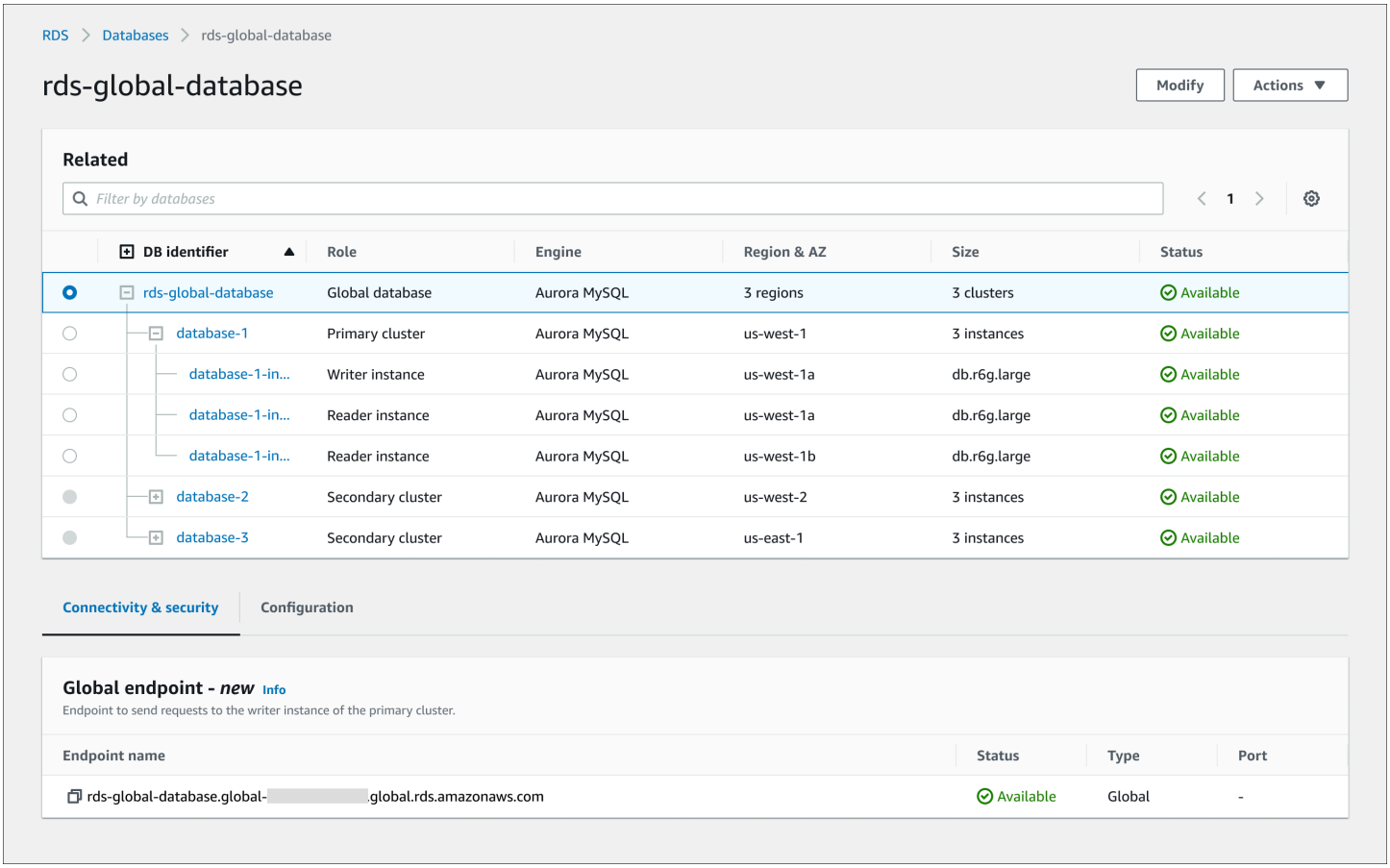
aws rds --region secondary_region \
  create-db-instance \
    --db-instance-class instance_class \
    --db-cluster-identifier secondary_cluster_id \
    --db-instance-identifier db_instance_id \
    --engine aurora-mysql | aurora-postgresql
```

对于 Windows：

```
aws rds --region secondary_region ^
  create-db-cluster ^
    --db-cluster-identifier secondary_cluster_id ^
    --global-cluster-identifier global_database_id_id ^
    --engine aurora-mysql | aurora-postgresql ^
    --engine-version version

aws rds --region secondary_region ^
  create-db-instance ^
    --db-instance-class instance_class ^
    --db-cluster-identifier secondary_cluster_id ^
    --db-instance-identifier db_instance_id ^
    --engine aurora-mysql | aurora-postgresql
```

RDS API

要使用 RDS API 将新 AWS 区域 添加到 Aurora Global Database，请运行 CreateDBCluster 操作。使用 `GlobalClusterIdentifier` 参数指定现有全局数据库的标识符。

在辅助区域中创建无管控 Aurora 数据库集群

尽管 Aurora Global Database 要求在与主区域之外的不同 AWS 区域 中至少有一个辅助 Aurora 数据库集群，但您可以对辅助集群使用无管控配置。无管控辅助 Aurora 数据库集群是没有数据库实例的集群。此类型的配置可以降低 Aurora 全局数据库的开支。在 Aurora 数据库集群中，计算和存储是分离的。如果没有数据库实例，您就无需为计算付费，而只需为存储付费。如果设置正确，无管控辅助存储卷将与主 Aurora 数据库集群保持同步。

您可以像平常创建 Aurora 全局数据库一样添加辅助集群。如果要在全球数据库中创建所有集群，请按照创建 Amazon Aurora Global Database中的过程操作。如果您已经有一个数据库集群可用作主集群，请按照将 AWS 区域 添加到 Amazon Aurora Global Database中的过程操作。

在主 Aurora 数据库集群开始复制到辅助数据库集群之后，您可以从辅助 Aurora 数据库集群中删除该 Aurora 只读数据库实例。此辅助集群现在被视为“无管控”集群，因为其不再有数据库实例。即使辅助集群中没有任何数据库实例，Aurora 也会将存储卷与主 Aurora 数据库集群保持同步。

> **Warning**
>
> 使用 Aurora PostgreSQL 时，要在辅助 AWS 区域 中创建无管控集群，请使用 AWS CLI 或 RDS API 添加辅助 AWS 区域。跳过为辅助集群创建读取器数据库实例的步骤。目前，RDS 控制台不支持创建无管控集群。有关要使用的 CLI 和 API 过程，请参阅 将 AWS 区域 添加到 Amazon Aurora Global Database。
>
> 如果全球数据库使用的 Aurora PostgreSQL 引擎版本低于 13.4、12.8 或 11.13，则在辅助区域中创建读取器数据库实例并随后将其删除，可能会导致主区域的写入器数据库实例上出现 Aurora PostgreSQL vacuum 问题。如果遇到此问题，请在删除辅助区域的读取器数据库实例后重启主区域的写入器数据库实例。

将无管控辅助 Aurora 数据库集群添加到您的 Aurora 全局数据库

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在 AWS Management Console 的导航窗格中，选择 Databases (数据库)。
3. 选择需要辅助 Aurora 数据库集群的 Aurora 全局数据库。确保主 Aurora 数据库集群为 `Available`。
4. 对于操作，选择添加 AWS 区域。
5. 在 Add a region（添加区域）页面上，选择辅助 AWS 区域。

您不能为同一个 Aurora Global Database 选择已有辅助 Aurora 数据库集群的 AWS 区域。此外，该区域也不能是主 Aurora 数据库集群所在的同一个区域。

6. 填写新的 AWS 区域中辅助 Aurora 集群的其余字段。这些配置选项与任何 Aurora 数据库集群实例的配置选项相同。

   对于基于 Aurora MySQL 的 Aurora 全局数据库，请忽略 Enable read replica write forwarding（启用只读副本写入转发）选项。删除读取器实例后，此选项不再起作用。

7. 选择添加 AWS 区域。完成将区域添加到 Aurora Global Database 的操作后，它会出现在屏幕截图中所示的AWS Management Console的 Databases (数据库) 列表中。



8. 继续操作之前，请先使用AWS Management Console或 AWS CLI 检查辅助 Aurora 数据库集群及其读取器实例的状态。例如：

```
$ aws rds describe-db-clusters --db-cluster-identifier secondary-cluster-id --query
 '*[].[Status]' --output text
```

   新添加的辅助 Aurora 数据库集群的状态可能需要几分钟的时间才能从 `creating` 更改为 `available`。当 Aurora 数据库集群处于可用状态时，您可以删除读取器实例。

9. 在辅助 Aurora 数据库集群中选择读取器实例，然后选择 Delete (删除)。



删除读取器实例后，辅助集群仍然是 Aurora 全局数据库的组成部分。该集群没有与其关联的实例，如下所示。



如果主 AWS 区域 发生计划外停机事件，您可以使用此无管控辅助 Aurora 数据库集群手动恢复 Amazon Aurora Global Database。

从 Aurora 或 Amazon RDS 快照创建 Amazon Aurora 全球数据库

您可以还原 Aurora 数据库集群的快照或从 Amazon RDS 数据库实例还原，以作为 Aurora 全局数据库的起点。您可以同时还原快照并创建新的 Aurora 预置数据库集群。然后，您可以将另一个 AWS 区域添加到还原的数据库集群，从而将其变成 Aurora Global Database。用这种方式使用快照创建的任何 Aurora 数据库集群都将成为 Aurora 全局数据库的主集群。

您使用的快照可以来自 `provisioned`，也可以来自 `serverless` Aurora 数据库集群。

在还原过程中，选择与快照相同的数据库引擎类型。例如，假设您要还原从正在运行 Aurora PostgreSQL 的 Aurora Serverless 数据库集群创建的快照。在这种情况下，您要使用相同的 Aurora 数据库引擎和版本创建 Aurora PostgreSQL 数据库集群。

在向 Aurora Global Database 添加 AWS 区域时，还原的数据库集群将代入该全局数据库的主集群角色。此主集群中包含的所有数据都将复制到您添加到 Aurora 全局数据库的任何辅助集群中。



管理 Amazon Aurora Global Database

您可对构成 Aurora 全球数据库的各个集群执行大多数的管理操作。当您在控制台中 Databases（数据库）页面上选择 Group related resources（对相关资源分组）时，您可以看到主集群和辅助集群分组到关联的全局数据库之下。要查找全局数据库集群正在运行的 AWS 区域、Aurora 数据库引擎和版本以及标识符，请使用 Configuration（配置）选项卡。

跨区域数据库失效转移过程仅适用于 Aurora 全球数据库，而不适用于单个 Aurora 数据库集群。要了解更多信息，请参阅“在 Amazon Aurora Global Database 中使用切换或失效转移”。

要从主区域的计划外停机中恢复 Aurora 全局数据库，请参阅 从计划外停机中恢复 Amazon Aurora Global Database。

主题

- 修改 Amazon Aurora Global Database
- 修改 Aurora 全局数据库的参数
- 从 Amazon Aurora Global Database 删除集群
- 删除 Amazon Aurora Global Database
- 标记 Amazon Aurora Global Database 资源

修改 Amazon Aurora Global Database

AWS Management Console中的 Databases (数据库) 页面列出您所有的 Aurora Global Database，同时显示每个全局数据库的主集群和辅助集群。Aurora 全局数据库有自己的配置设置。具体来说，它具有与其主集群和辅助集群关联的 AWS 区域，如以下屏幕截图所示。



当您对 Aurora 全局数据库进行更改时，您有机会取消更改，如以下屏幕截图中所示。



选择 Continue (继续) 时，即表示您确认更改。

修改 Aurora 全局数据库的参数

您可以为 Aurora 全局数据库中的每个 Aurora 集群独立配置 Aurora 数据库集群参数组。大多数参数的工作方式与其他类型的 Aurora 集群相同。我们建议您在全局数据库中使所有集群之间的设置保持一致。在将辅助集群提升为主集群时，此操作有助于避免意外的行为变化。

例如，对于时区和字符集使用相同设置，可避免在不同集群作为主集群时出现不一致的行为。

`aurora_enable_repl_bin_log_filtering` 和 `aurora_enable_replica_log_compression` 配置设置没有效果。

从 Amazon Aurora Global Database 删除集群

出于多种不同原因，您可以从 Aurora 全局数据库中删除 Aurora 数据库集群。例如，如果主集群被降级或隔离，您可能希望从 Aurora 全局数据库中删除 Aurora 数据库集群。然后，它将成为独立的预置 Aurora 数据库集群，可用于创建新的 Aurora 全局数据库。要了解更多信息，请参阅“从计划外停机中恢复 Amazon Aurora Global Database”。

您也可能会想要删除 Aurora 数据库集群，因为您想要删除不再需要的 Aurora 全局数据库。在删除（分离）所有关联的 Aurora 数据库集群以后，您才能删除 Aurora 全局数据库，仅保留主数据库。有关更多信息，请参阅“删除 Amazon Aurora Global Database”。

当 Aurora 数据库集群从 Aurora 全局数据库中分离时，它将不再与主数据库同步。它将成为一个拥有完全读/写功能的独立预置 Aurora 数据库集群。

您可以使用AWS Management Console、AWS CLI 或 RDS API 从 Aurora Global Database 中删除 Aurora 数据库集群。

控制台

从 Aurora 全局数据库删除 Aurora 集群

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在 Databases (数据库) 页面上选择集群。
3. 对于 Actions (操作)，选择 Remove from Global (从全局数据库移除)。



将会出现一个提示，要求确认您要将辅助集群从 Aurora 全局数据库中分离。



4. 选择 Remove and promote (删除并提升) 以从全局数据库中删除集群。

Aurora 数据库集群不再作为 Aurora 全局数据库中的辅助集群，也不再与主数据库集群同步。它是一个拥有完全读/写功能的独立 Aurora 数据库集群。



在移除或删除所有辅助集群后，您可以按同样方式移除主集群。在删除全部辅助集群之前，您无法将主 Aurora 数据库集群从 Aurora 全局数据库中分离（删除）。

Aurora 全局数据库可能保留在数据库列表中，其中有零个区域和可用区。如果不想再使用此 Aurora 全局数据库，则可以删除。有关更多信息，请参阅“删除 Amazon Aurora Global Database”。

AWS CLI

要从 Aurora 全局数据库删除 Aurora 集群，请使用以下参数运行 remove-from-global-cluster CLI 命令：

- `--global-cluster-identifier` – Aurora 全局数据库的名称（标识符）。
- `--db-cluster-identifier` – 要从 Aurora 全局数据库删除的每个 Aurora 数据库集群的名称。删除主数据库集群之前，先删除所有 Aurora 辅助数据库集群。

以下示例先从 Aurora 全局数据库中移除辅助集群，然后移除主集群。

对于 Linux、macOS 或 Unix：

```
aws rds --region secondary_region \
  remove-from-global-cluster \
    --db-cluster-identifier secondary_cluster_ARN \
    --global-cluster-identifier global_database_id

aws rds --region primary_region \
  remove-from-global-cluster \
    --db-cluster-identifier primary_cluster_ARN \
    --global-cluster-identifier global_database_id
```

对 Aurora Global Database 中的每个辅助 AWS 区域 重复 `remove-from-global-cluster --db-cluster-identifier secondary_cluster_ARN` 命令。

对于 Windows：

```
aws rds --region secondary_region ^
  remove-from-global-cluster ^
    --db-cluster-identifier secondary_cluster_ARN ^
    --global-cluster-identifier global_database_id

aws rds --region primary_region ^
  remove-from-global-cluster ^
    --db-cluster-identifier primary_cluster_ARN ^
    --global-cluster-identifier global_database_id
```

对 Aurora Global Database 中的每个辅助 `remove-from-global-cluster --db-cluster-identifier secondary_cluster_ARN` 重复 AWS 区域 命令。

RDS API

要使用 RDS API 从 Aurora 全局数据库移除 Aurora 集群，请运行 RemoveFromGlobalCluster 操作。

删除 Amazon Aurora Global Database

由于 Aurora 全局数据库通常容纳业务关键型数据，因此您不能一步删除全局数据库及其关联集群。要删除 Aurora 全局数据库，请执行以下操作：

- 从 Aurora 全局数据库删除所有辅助数据库集群。每个集群都会变成独立的 Aurora 数据库集群。要了解如何操作，请参阅从 Amazon Aurora Global Database 删除集群。
- 从每个独立 Aurora 数据库集群中删除所有 Aurora 副本。
- 从 Aurora 全局数据库删除主数据库集群。这将成为独立的 Aurora 数据库集群。

- 从 Aurora 主数据库集群中，首先删除所有 Aurora 副本，然后删除写入器数据库实例。

从新近独立的 Aurora 数据库集群中删除写入器实例通常还会删除 Aurora 数据库集群和 Aurora 全局数据库。

有关更多一般信息，请参阅 从 Aurora 数据库集群中删除数据库实例。

要删除 Aurora Global Database，您可以使用AWS Management Console、AWS CLI 或 RDS API。

控制台

删除 Aurora 全局数据库

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 选择 Databases (数据库)，然后在列表中找到要删除的 Aurora 全局数据库。
3. 确认所有集群从 Aurora 全局数据库中移除。Aurora 全局数据库应显示 0 个区域和可用区且集群的大小为 0。

   如果 Aurora 全局数据库包含任何 Aurora 数据库集群，则无法将其删除。如有必要，请从 Aurora 全局数据库分离主和辅助 Aurora 数据库集群。有关更多信息，请参阅“从 Amazon Aurora Global Database 删除集群”。
4. 在列表中选择 Aurora 全球数据库，然后从操作菜单中选择删除。



AWS CLI

要删除 Aurora Global Database，请使用 AWS 区域 的名称和 Aurora Global Database 标识符运行 delete-global-cluster CLI 命令，如下例中所示。

对于 Linux、macOS 或 Unix：

```
aws rds --region primary_region delete-global-cluster \
   --global-cluster-identifier global_database_id
```

对于 Windows：

```
aws rds --region primary_region delete-global-cluster ^
   --global-cluster-identifier global_database_id
```

RDS API

要删除作为 Aurora 全局数据库一部分的集群，请运行 DeleteGlobalCluster API 操作。

标记 Amazon Aurora Global Database 资源

借助 Aurora Global Database 特征，您可以将 RDS 标签应用于全局数据库中不同级别的资源。如果您不熟悉如何对 AWS 或 Aurora 资源使用标签，请在全局数据库中应用标签之前先参阅为 Amazon Aurora 和 Amazon RDS 资源添加标签。

> **Note**
>
> 由于 AWS 处理标签数据是其成本报告机制的一部分，因此不要在标签名称或值中包含任何敏感数据或个人身份信息（PII）。

您可以将标签应用于全局数据库中的以下类型的资源：

- 整个全局数据库的容器对象。此资源称为全局集群。

  在控制台中执行添加 AWS 区域操作创建全局集群后，您可以使用该全局集群的详细信息页面添加标签。在全局集群详细信息页面的标签选项卡上，您可以通过选择管理标签来添加、删除或修改标签及其关联值。

  使用 AWS CLI 和 RDS API，您可以在创建全局集群的同时向其添加标签。您还可以添加、删除或修改现有全局集群的标签。
- 主集群。此处使用与独立 Aurora 集群相同的步骤来处理标签。您可以在将原始 Aurora 集群转换为全局数据库之前设置标签。您也可以通过在数据库集群详细信息页面的标签选项卡上，选择管理标签来添加、删除或修改标签及其关联值。

- 任何辅助集群。此处使用与独立 Aurora 集群相同的步骤来处理标签。您可以通过使用控制台中的添加 AWS 区域操作，在创建辅助 Aurora 集群的同时设置标签。您也可以通过在数据库集群详细信息页面的标签选项卡上，选择管理标签来添加、删除或修改标签及其关联值。
- 主集群或辅助集群中的各个数据库实例。此处使用与 Aurora or RDS 数据库实例相同的步骤来处理标签。您可以通过使用控制台中的添加读取器操作，在向 Aurora 集群添加新数据库实例的同时设置标签。您也可以通过在数据库实例详细信息页面的标签选项卡上，选择管理标签来添加、删除或修改标签及其关联值。

以下是您可能在全局数据库中分配的标签类型的一些示例：

- 您可能会向全局集群添加标签以记录有关应用程序的总体信息，例如代表组织内所有者和联系人的匿名标识符。您可能会使用标签来表示使用全局数据库的应用程序的属性。
- 您可能会向主集群和辅助集群添加标签，以便在 AWS 区域级别跟踪应用程序的成本。有关该过程的详细信息，请参阅如何将 AWS 账单与 Amazon RDS 中的标签结合使用。
- 您可能会向具有 Aurora 集群的特定数据库实例添加标签，以指明其特殊用途。例如，在主集群中，您可能有一个失效转移优先级较低的读取器实例，该实例专门用于报告生成。标签可以将此特殊用途的数据库实例与主集群中专用于实现高可用性的其他实例区分开来。
- 您可以在全局数据库资源的各个级别上使用标签，通过 IAM 策略来控制访问权限。有关更多信息，请参阅《AWS Identity and Access Management 用户指南》中的控制对 AWS 资源的访问。

  > **Tip**
  >
  > 在 AWS Management Console中，创建全局集群容器后，您可以在单独的步骤中向该容器添加标签。如果希望在没有访问控制标签的全局集群存在时避免任何时间间隔，则可以在 `CreateGlobalCluster` 操作期间通过 AWS CLI、RDS API 或 AWS CloudFormation 模板创建该资源来应用这些标签。

- 您可能会在集群级别或对全局集群使用标签，以记录有关应用程序质量保证和测试的信息。例如，您可能会在数据库集群上指定一个标签，以记录您上次切换到该集群的时间。您可能会在全局集群上指定一个标签，以记录整个应用程序上次灾难恢复演练的时间。

全局数据库标记的 AWS CLI 示例

以下 AWS CLI 示例说明如何为全局数据库中所有类型的 Aurora 资源指定和检查标签。

您可以使用 `create-global-cluster` 命令为全局集群容器指定标签。以下示例创建一个全局集群并为其分配两个标签。这些标签带有键 `tag1` 和 `tag2`。

```
$  aws rds create-global-cluster --global-cluster-identifier my_global_cluster_id \
  --engine aurora-mysql --tags Key=tag1,Value=val1 Key=tag2,Value=val2
```

您可以使用 describe-global-clusters 命令在全局集群容器上列出标签。使用标签时，您通常会先运行该命令，以检索全局集群的 Amazon 资源名称（ARN）。在后续命令中，您将使用 ARN 作为参数来处理标签。以下命令显示 TagList 属性中的标签信息。它还显示 ARN，该 ARN 在后面的示例中将用作参数。

```
$  aws rds describe-global-clusters --global-cluster-identifier my_global_cluster_id
{
    "GlobalClusters": [
        {
            "Status": "available",
            "Engine": "aurora-mysql",
            "GlobalClusterArn": "my_global_cluster_arn",
            ...
            "TagList": [
                {
                    "Value": "val1",
                    "Key": "tag1"
                },
                {
                    "Value": "val2",
                    "Key": "tag2"
                }
            ]
        }
    ]
}
```

您可以使用 add-tags-to-resource 命令添加新标签。使用该命令，您可以指定全局集群的 Amazon 资源名称（ARN），而不是集群标识符。添加与现有标签具有相同名称的标签会覆盖标签值。如果您在标签值中包含空格或特殊字符，请根据操作系统或命令 shell 引用相应的值。以下示例修改上一个示例中的全局集群的标签。最初，集群的标签带有键 tag1 和 tag2。命令完成后，全局集群会有一个带有键 tag3 的新标签，而带有键 tag1 的标签具有不同的值。

```
$  aws rds add-tags-to-resource --resource-name my_global_cluster_arn \
  --tags Key=tag1,Value="new value for tag1" Key=tag3,Value="entirely new tag"

$  aws rds describe-global-clusters --global-cluster-identifier my_global_cluster_id
{
```

```
    "GlobalClusters": [
        {
            "Status": "available",
            "Engine": "aurora-mysql",
            ...
            "TagList": [
                {
                    "Value": "new value for tag1",
                    "Key": "tag1"
                },
                {
                    "Value": "val2",
                    "Key": "tag2"
                },
                {
                    "Value": "entirely new tag",
                    "Key": "tag3"
                }
            ]
        }
    ]
}
```

您可以使用 `remove-tags-from-resource` 命令从全局集群中删除标签。使用该命令，您只需指定一组标签键，而无需指定任何标签值。以下示例修改上一个示例中的全局集群的标签。最初，集群的标签带有键 `tag1`、`tag2` 和 `tag3`。命令完成后，只剩下带有键 `tag1` 的标签。

```
$  aws rds remove-tags-from-resource --resource-name my_global_cluster_arn --tag-keys
 tag2 tag3

$  aws rds describe-global-clusters --global-cluster-identifier my_global_cluster_id
{
    "GlobalClusters": [
        {
            "Status": "available",
            "Engine": "aurora-mysql",
            ...
            "TagList": [
                {
                    "Value": "new value for tag1",
                    "Key": "tag1"
                }
            ]
```

```
        }
    ]
}
```

连接到 Amazon Aurora Global Database

每个 Aurora Global Database 都有一个写入器端点，Aurora 会自动更新该端点，以将请求路由到主数据库集群的当前写入器实例。有了写入器端点，您在使用托管式 Aurora Global Database 切换和失效转移功能更改主区域的位置后，便无需修改连接字符串。要了解有关使用写入器端点以及 Aurora Global Database 切换和失效转移的更多信息，请参阅在 Amazon Aurora Global Database 中使用切换或失效转移。有关使用 RDS 代理连接到 Aurora Global Database 的信息，请参阅在 Aurora Global Database 中使用 RDS 代理。

主题

- 选择满足您的应用程序需求的端点
- 查看 Amazon Aurora Global Database 的端点
- 使用全局写入器端点时的注意事项

选择满足您的应用程序需求的端点

是否连接到 Aurora Global Database 取决于您对数据库的读取或写入需求，以及您要将请求路由到的 AWS 区域。以下是一些典型的使用案例：

- 将请求路由到写入器实例：如果您需要运行数据操作语言（DML）和数据定义语言（DDL）语句，或者如果您需要在读取和写入之间保持强一致性，请连接到 Aurora Global Database 写入器端点。该端点将请求路由到您的全局数据库主集群中的写入器实例。该端点会自动更新以将请求路由到写入器实例，从而无需在每次更改全局集群中的写入器位置时更新应用程序。您还可以使用全局端点向写入器发送跨区域读/写请求。

> **Note**
>
> 如果您在 Aurora Global Database 写入器端点可用之前设置全局数据库，则您的应用程序可能会连接到主集群的集群端点。在这种情况下，建议将连接设置切换为使用全局写入器端点。这样一来，就无需在每次 Aurora Global Database 切换或失效转移后更改连接设置。

写入器端点名称的第一部分是您的 Aurora Global Database 的名称。因此，如果重命名 Aurora Global Database，则写入器端点名称会更改，使用该名称的任何代码都必须更新为新名称。

- 将读取扩展至更接近应用程序的区域：要在您的应用程序所在或附近的 AWS 区域中扩展只读请求，请连接到主或辅助 Aurora 集群的读取器端点。
- 通过偶尔跨区域写入来扩展读取：对于偶尔的 DML 语句（例如用于维护和数据清理），请连接到启用了写入转发的辅助集群的读取器端点。通过写入转发，Aurora 会自动将写入语句转发给 Aurora Global Database 主区域中的写入器。写入转发提供以下优势：
  - 您无需执行繁重的工作即可在辅助集群和主集群之间建立连接以发送跨区域写入。
  - 您无需在应用程序中拆分读取和写入请求。
  - 您无需开发复杂的逻辑来管理先写后读请求的一致性。

  但是，使用写入转发时，在执行跨区域失效转移或切换后，您确实需要更新应用程序代码或配置，才能连接到新提升的主区域的读取器端点。建议您监控通过写入转发完成的操作的延迟，以检查处理写入请求的开销。最后，写入转发不支持某些 MySQL 或 PostgreSQL 操作，例如更改数据定义语言（DDL）或执行 `SELECT FOR UPDATE` 语句。

  要了解有关跨 AWS 区域使用写入转发的更多信息，请参阅 在 Amazon Aurora Global Database 中使用写入转发。

有关不同类型的 Aurora 端点的详细信息，请参阅 连接到 Amazon Aurora 数据库集群。

查看 Amazon Aurora Global Database 的端点

当您在控制台中查看 Aurora Global Database 时，可以看到与其所有集群关联的所有端点。下图显示了您在查看主数据库集群详细信息时看到的端点类型的示例：

- 全局写入器 – 始终指向全局数据库集群的当前写入器数据库实例的单个读/写端点。
- 写入器 – 发送至全局数据库集群中主数据库集群的读/写请求的连接端点。
- 读取器 – 发送至全局数据库集群中主数据库集群或辅助数据库集群的只读请求的连接端点。为了最大限度地减少延迟，请选择您所在 AWS 区域或离您最近的 AWS 区域的任何读取器端点。



控制台

查看全局数据库的端点

1. 登录AWS Management Console并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Databases (数据库)。
3. 在列表中，选择要查看其端点的全局数据库，或者主数据库集群或辅助数据库集群。
4. 选择连接和安全性选项卡以查看端点详细信息。显示哪些端点取决于您选择的集群类型，如下所示：

   - 全局数据库 – 全局写入器端点。
   - 主数据库集群 – 全局写入器端点，以及主集群的集群端点和读取器端点。
   - 辅助数据库集群 – 辅助集群的集群端点和读取器端点。在辅助集群上，集群端点显示为非活动状态，因为它不处理写入请求。您仍可连接到集群端点，但仅限于读取查询。

AWS CLI

要查看全局集群的写入器端点，请使用 AWS CLI describe-global-clusters 命令，如以下示例所示。

```
aws rds describe-global-clusters --region aws_region
{
    "GlobalClusters": [
        {
            "GlobalClusterIdentifier": "global_cluster_id",
            "GlobalClusterResourceId": "cluster-unique_string",
            "GlobalClusterArn": "arn:aws:rds::123456789012:global-
cluster:global_cluster_id",
            "Status": "available",
            "Engine": "aurora-mysql",
            "EngineVersion": "5.7.mysql_aurora.2.11.2",
            "GlobalClusterMembers": [

              ...
            ],
            "Endpoint":
 "global_cluster_id.global-unique_string.global.rds.amazonaws.com"
        }
    ]
}
```

要查看全局集群的成员数据库集群的集群端点和读取器端点，请使用 AWS CLI describe-db-clusters 命令，如以下示例所示。Endpoint 和 ReaderEndpoint 的返回值分别是集群端点和读取器端点。

```
aws rds describe-db-clusters --region primary_region --db-cluster-
identifier db_cluster_id
{
    "DBClusters": [
        {
            "AllocatedStorage": 1,
            "AvailabilityZones": [
                "az_1",
                "az_2",
                "az_3"
            ],
            "BackupRetentionPeriod": 1,
            "DBClusterIdentifier": "db_cluster_id",
            "DBClusterParameterGroup": "default.aurora-mysql5.7",
            "DBSubnetGroup": "default",
```

```
            "Status": "available",
            "EarliestRestorableTime": "2023-08-01T18:21:11.301Z",
            "Endpoint":
 "db_cluster_id.cluster-unique_string.primary_region.rds.amazonaws.com",
            "ReaderEndpoint": "db_cluster_id.cluster-
ro-unique_string.primary_region.rds.amazonaws.com",
            "MultiAZ": false,
            "Engine": "aurora-mysql",
            "EngineVersion": "5.7.mysql_aurora.2.11.2",

            "ReadReplicaIdentifiers": [
                "arn:aws:rds:secondary_region:123456789012:cluster:db_cluster_id"
            ],
            "DBClusterMembers": [
                {
                    "DBInstanceIdentifier": "db_instance_id",
                    "IsClusterWriter": true,
                    "DBClusterParameterGroupStatus": "in-sync",
                    "PromotionTier": 1
                }
            ],

            ...
            "TagList": [],
            "GlobalWriteForwardingRequested": false
        }
    ]
}
```

RDS API

要查看全局集群的写入器端点，请使用 RDS API DescribeGlobalClusters 操作。要查看全局集群的成员数据库集群的集群端点和读取器端点，请使用 RDS API DescribeDBClusters 操作。

使用全局写入器端点时的注意事项

您可以按照以下指南和最佳实践来有效利用 Aurora Global Database 写入器端点：

- 为了更大限度地减少跨区域失效转移或切换后的中断，您可以在应用程序计算与主区域和辅助 AWS 区域之间设置 VPC 连接。例如，假设您的应用程序或客户端系统与主集群在同一 VPC 中运行。如果辅助集群已提升，则全局写入器端点会自动更改为指向该集群。尽管全局写入器端点使您能够避免更改应用程序的连接设置，但在两个 VPC 之间设置网络之前，您的应用程序无法访问新提升的

主 AWS 区域 VPC 中的 IP 地址。要评估设置此连接的不同选项，请参阅 Amazon VPC 到 Amazon VPC 的连接选项。

- 全球数据库失效转移或切换后的全局写入器端点更新可能需要很长时间，具体取决于您的域名服务（DNS）缓存持续时间。要了解更多信息，请参阅 Amazon Aurora MySQL Database Administrator's Handbook。Aurora Global Database 在发现全局写入器端点上的 DNS 更改时会触发 RDS 事件。您可以使用该事件来设计策略，以确保 DNS 缓存不会超过事件生成后的时间。有关更多信息，请参阅数据库集群事件。
- Aurora Global Database 以异步方式复制数据。跨区域失效转移方法可能导致一些写入事务数据在启动失效转移之前没有复制到所选的辅助实例。尽管 Aurora 会尽量尝试阻止原始主 AWS 区域中的写入，但失效转移可能会受到脑裂问题的影响。尽量减少数据丢失和脑裂风险的注意事项也适用于 Aurora Global Database 写入器端点。有关更多信息，请参阅执行 Aurora 全球数据库的托管式失效转移。

在 Amazon Aurora Global Database 中使用写入转发

使用写入转发功能，可以减少管理在 Aurora 全局数据库上运行的应用程序所需的终端节点数量。启用写入转发功能后，Aurora 全局数据库中的辅助集群会将执行写入操作的 SQL 语句转发到主集群。主集群会更新源，然后将产生的更改传播回所有辅助 AWS 区域。

写入转发配置可让您不必实施自己的机制以将写入操作从辅助AWS区域发送到主要区域。Aurora 处理跨区域联网设置。Aurora 还会为每个语句传输所有必要的会话和事务上下文。始终首先在主集群上更改数据，然后复制到 Aurora 全局数据库中的辅助集群。这样，主集群成为所有数据的真实来源并始终拥有所有数据的最新副本。

主题

- 在 Aurora MySQL 全局数据库中使用写入转发
- 在 Aurora PostgreSQL 全局数据库中使用写入转发

在 Aurora MySQL 全局数据库中使用写入转发

主题

- Aurora MySQL 中写入转发的区域和版本可用性
- 在 Aurora MySQL 中启用写入转发
- 检查辅助集群是否在 Aurora MySQL 中启用了写入转发
- 应用程序和 SQL 与 Aurora MySQL 中写入转发的兼容性

- Aurora MySQL 中写入转发的隔离和一致性
- 使用 Aurora MySQL 中的写入转发运行多部分语句
- 使用 Aurora MySQL 中写入转发执行的事务
- Aurora MySQL 中写入转发的配置参数
- Aurora MySQL 中写入转发的 Amazon CloudWatch 指标
- 用于写入转发的 Aurora MySQL 状态变量

Aurora MySQL 中写入转发的区域和版本可用性

在每个提供基于 Aurora MySQL 的全局数据库的区域，Aurora MySQL 2.08.1 及更高版本都支持写入转发。

有关 Aurora MySQL 全局数据库的版本和区域可用性的更多信息，请参阅使用 Aurora MySQL 的 Aurora 全局数据库。

在 Aurora MySQL 中启用写入转发

默认情况下，在将辅助集群添加到 Aurora Global Database 时不启用写入转发。

要使用 AWS Management Console启用写入转发，请在为全局数据库添加区域时，选中只读副本写入转发下的开启全局写入转发复选框。对于现有的辅助集群，将集群修改为开启全局写入转发。要关闭写入转发，请在添加区域或修改辅助集群时，清除开启全局写入转发复选框。

要使用 AWS CLI 启用写入转发，请使用 `--enable-global-write-forwarding` 选项。当您使用 `create-db-cluster` 命令创建新的辅助集群时，此选项将起作用。当您使用 `modify-db-cluster` 命令修改现有辅助集群时，它也可以起作用。它要求全局数据库使用支持写入转发的 Aurora 版本。您可以通过对这些相同的 CLI 命令使用 `--no-enable-global-write-forwarding` 选项来关闭写入转发。

要使用 Amazon RDS API 启用写入转发，请将 `EnableGlobalWriteForwarding` 参数设置为 `true`。当您使用 `CreateDBCluster` 操作创建新的辅助集群时，此参数起作用。当您使用 `ModifyDBCluster` 操作修改现有辅助集群时，它也起作用。它要求全局数据库使用支持写入转发的 Aurora 版本。您可以通过将 `EnableGlobalWriteForwarding` 参数设置为 `false` 来关闭写入转发。

> **Note**
>
> 要使数据库会话使用写入转发，请为 aurora_replica_read_consistency 配置参数指定设置。在使用写入转发功能的每个会话中执行此操作。有关此参数的信息，请参阅Aurora MySQL 中写入转发的隔离和一致性。
> RDS 代理功能对于 aurora_replica_read_consistency 变量不支持 SESSION 值。设置此值可能导致意外行为。

以下 CLI 示例说明如何可以在启用或禁用写入转发的情况下设置 Aurora Global Database。突出显示的项表示在为 Aurora Global Database 设置基础设施时务必指定且保持一致的命令和选项。

以下示例创建启用写入转发的 Aurora Global Database、主集群和辅助集群。将用户名、密码以及主要和辅助 AWS 区域替换为您自己选择的内容。

```
# Create overall global database.
aws rds create-global-cluster --global-cluster-identifier write-forwarding-test \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-east-1

# Create primary cluster, in the same AWS Region as the global database.
aws rds create-db-cluster --global-cluster-identifier write-forwarding-test \
  --db-cluster-identifier write-forwarding-test-cluster-1 \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --master-username user_name --master-user-password password \
  --region us-east-1

aws rds create-db-instance --db-cluster-identifier write-forwarding-test-cluster-1 \
  --db-instance-identifier write-forwarding-test-cluster-1-instance-1 \
  --db-instance-class db.r5.large \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-east-1

aws rds create-db-instance --db-cluster-identifier write-forwarding-test-cluster-1 \
  --db-instance-identifier write-forwarding-test-cluster-1-instance-2 \
  --db-instance-class db.r5.large \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-east-1

# Create secondary cluster, in a different AWS Region than the global database,
# with write forwarding enabled.
aws rds create-db-cluster --global-cluster-identifier write-forwarding-test \
```

```
  --db-cluster-identifier write-forwarding-test-cluster-2 \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-east-2 \
  --enable-global-write-forwarding

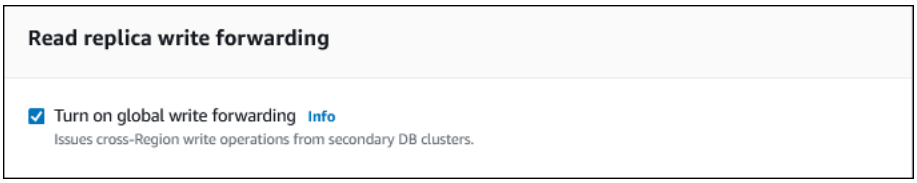
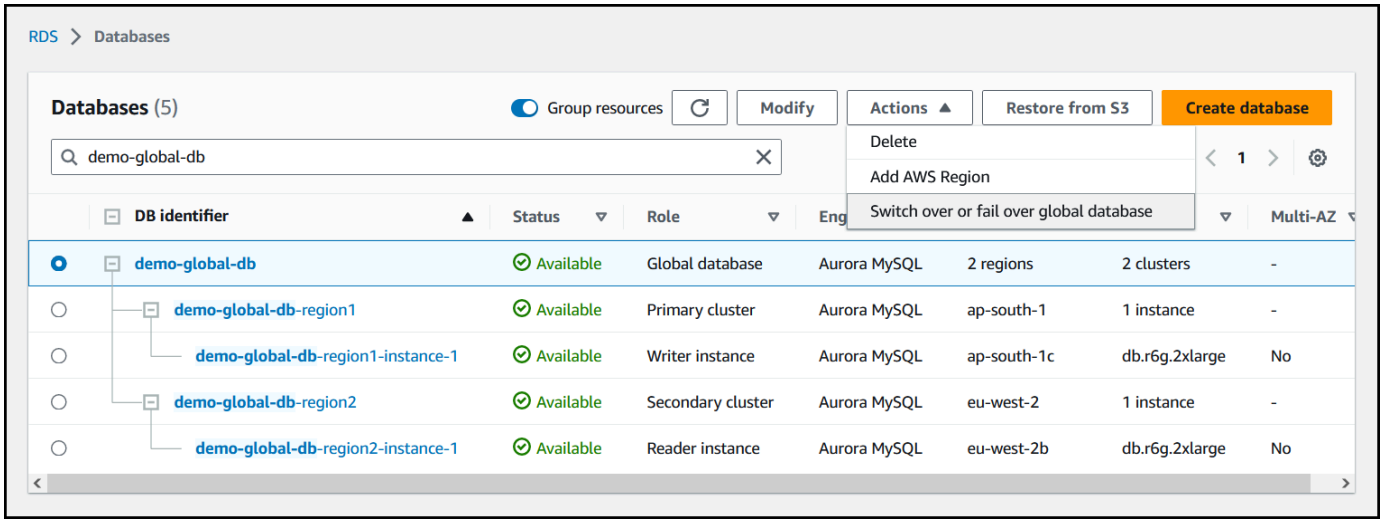
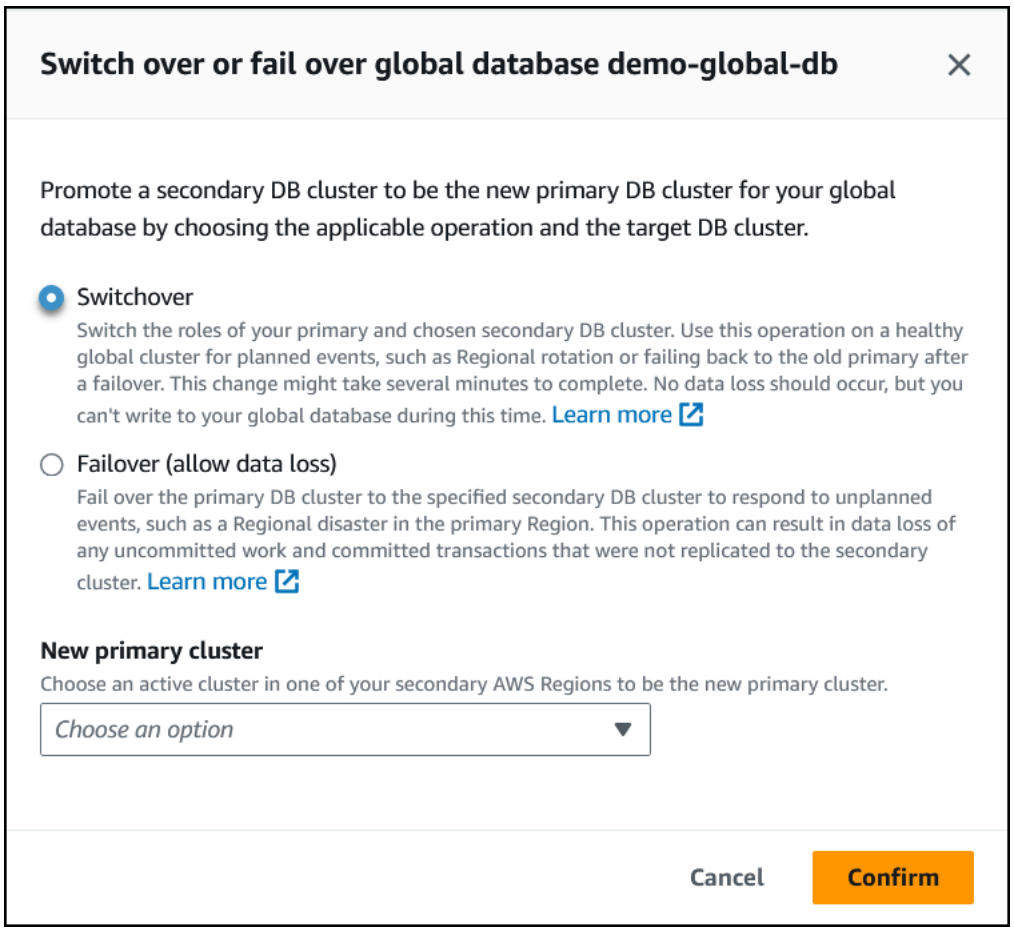
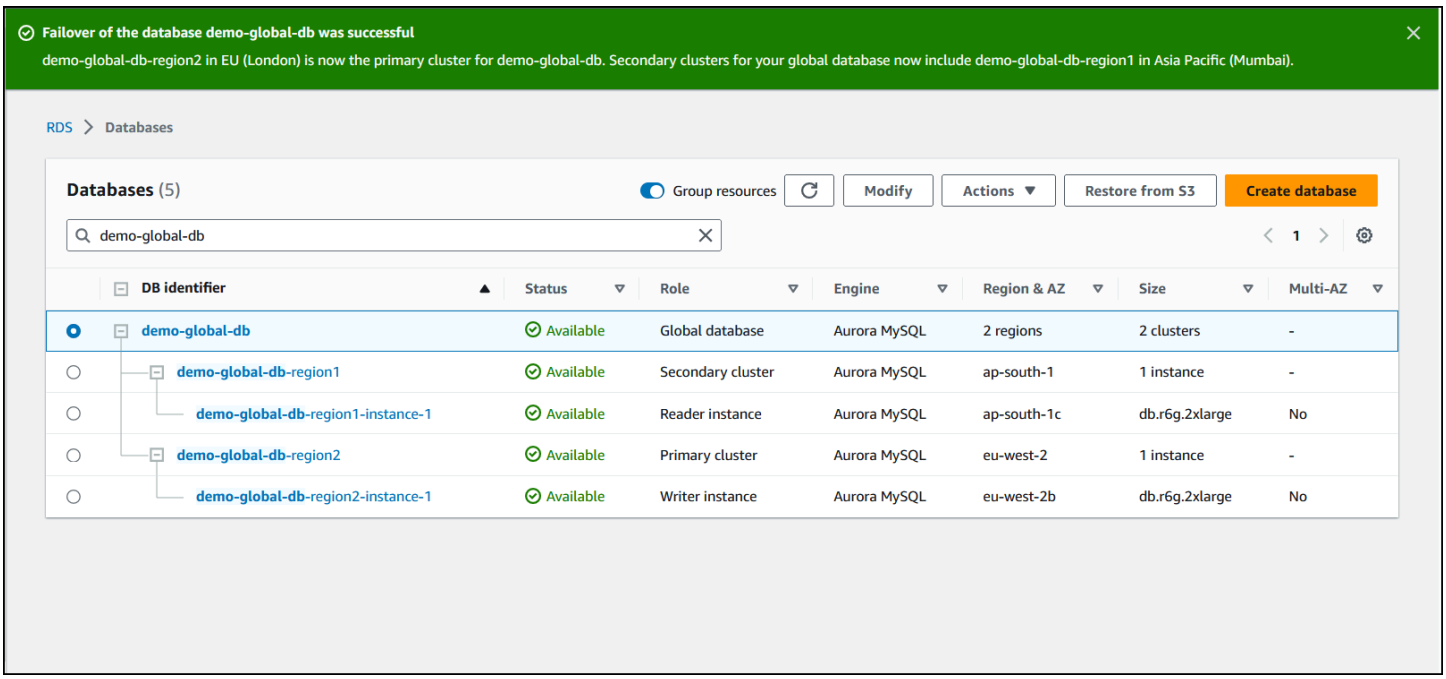
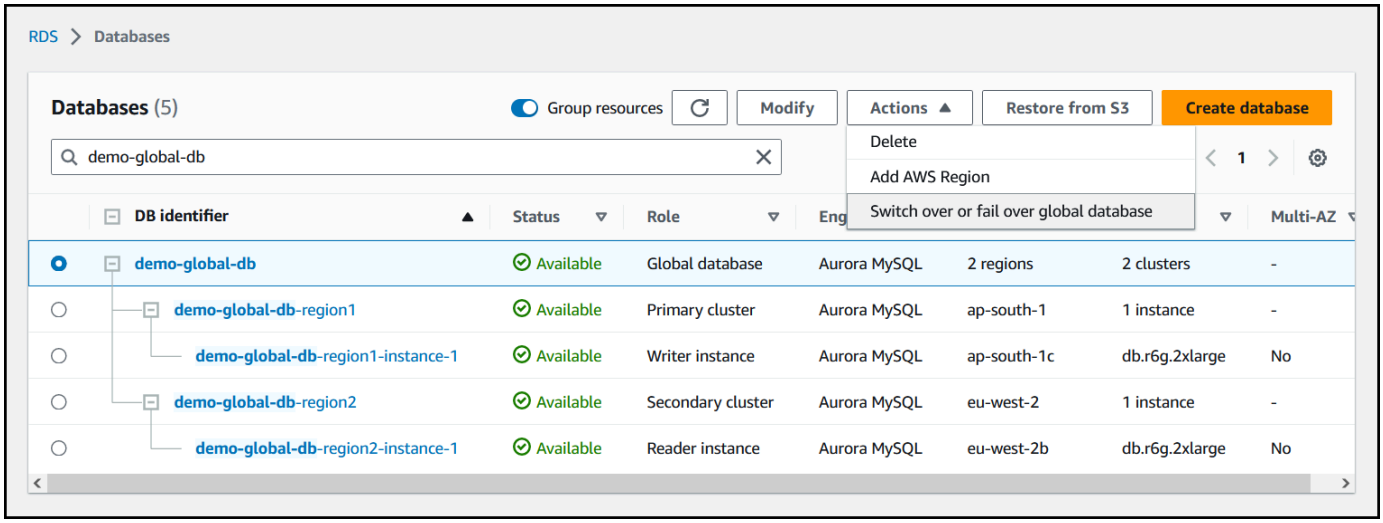
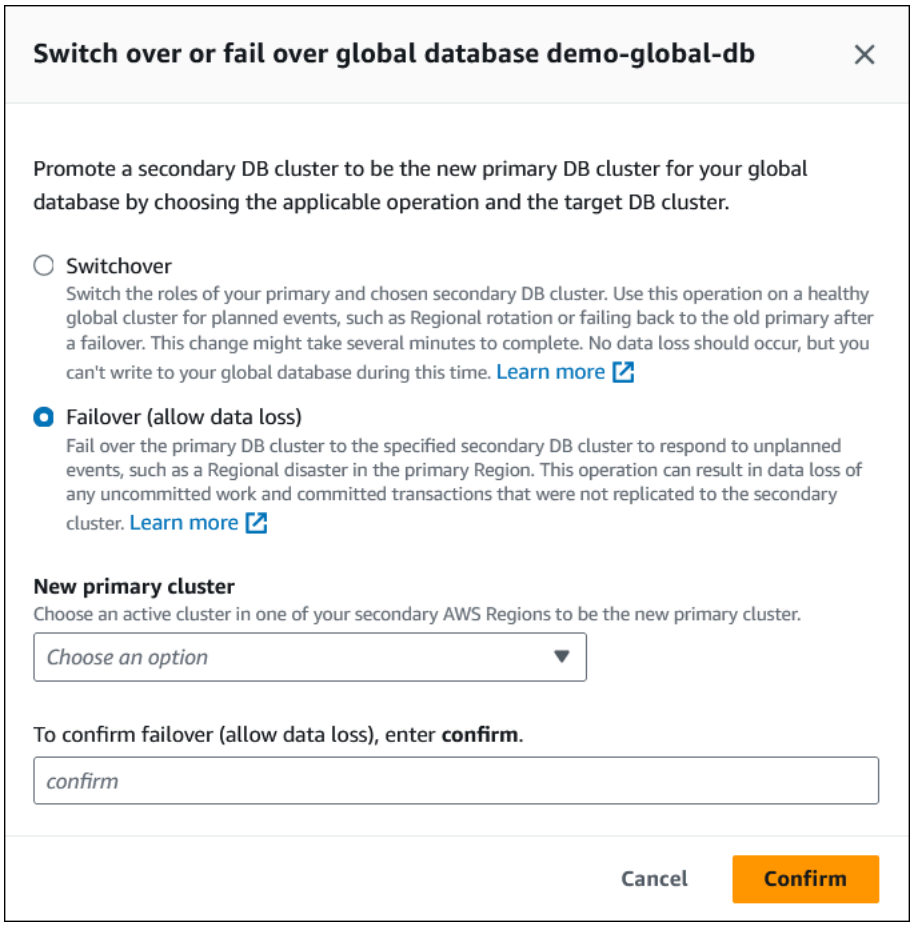
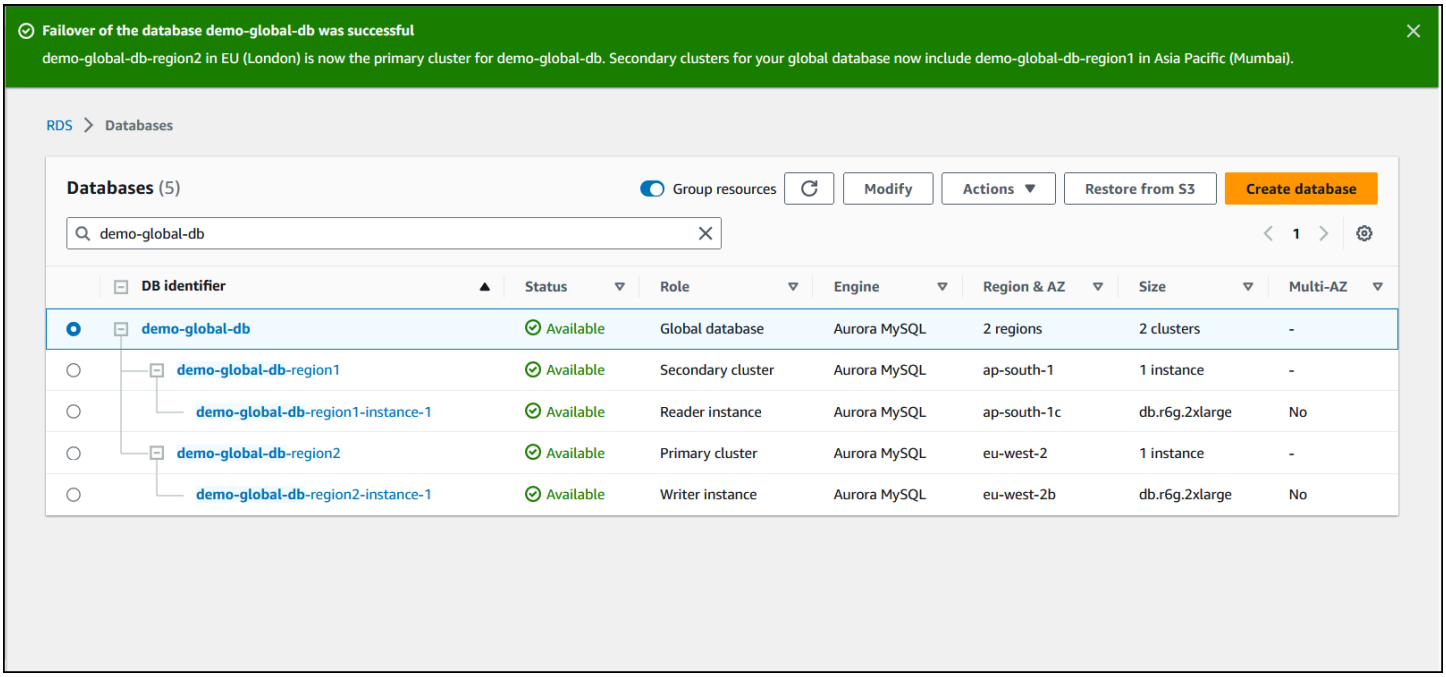
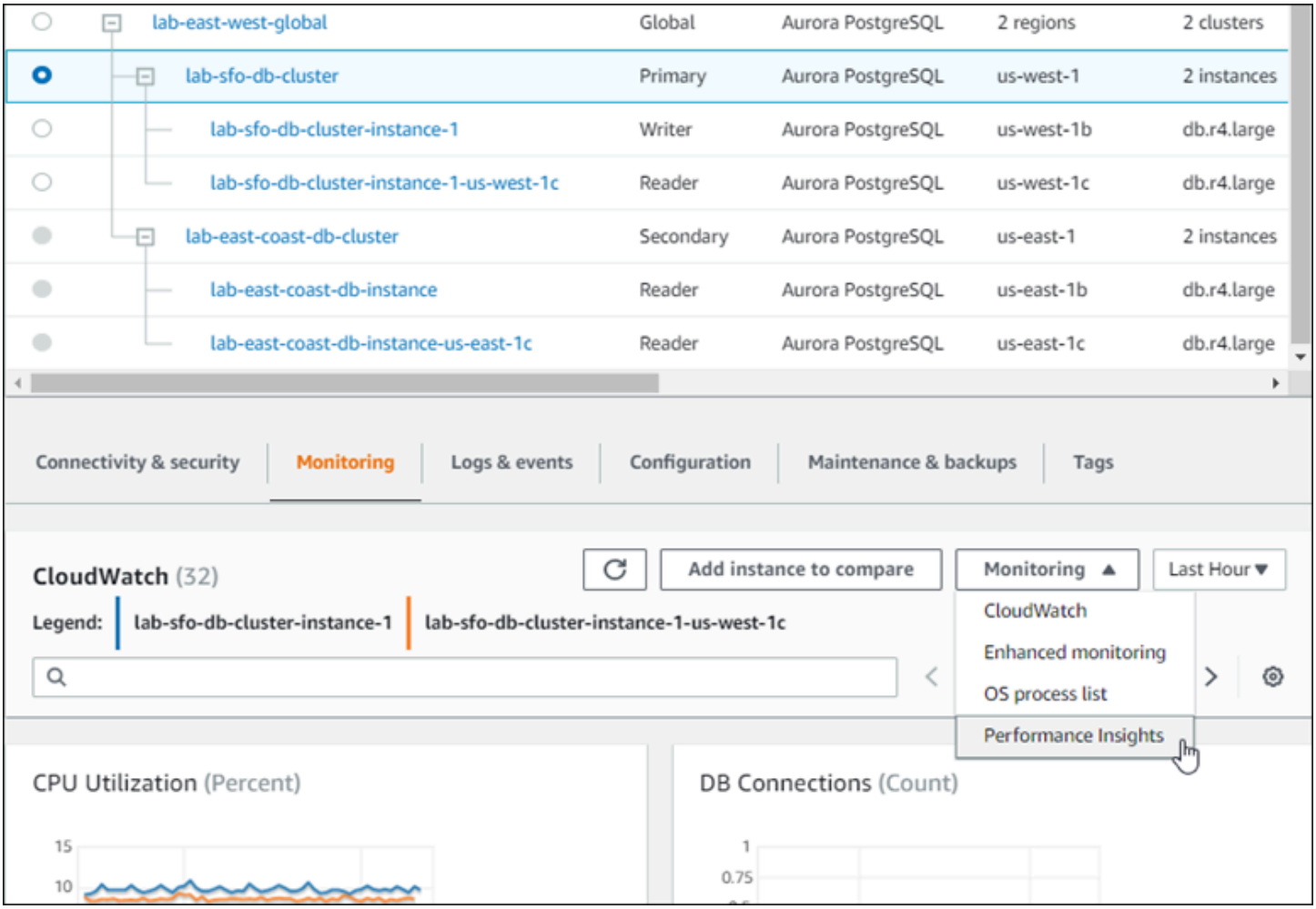
aws rds create-db-instance --db-cluster-identifier write-forwarding-test-cluster-2 \
  --db-instance-identifier write-forwarding-test-cluster-2-instance-1 \
  --db-instance-class db.r5.large \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-east-2

aws rds create-db-instance --db-cluster-identifier write-forwarding-test-cluster-2 \
  --db-instance-identifier write-forwarding-test-cluster-2-instance-2 \
  --db-instance-class db.r5.large \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-east-2
```

以下示例从上一个示例继续。它创建一个未启用写入转发的辅助集群，然后启用写入转发。完成此示例后，全局数据库中的所有辅助集群都启用了写入转发功能。

```
# Create secondary cluster, in a different AWS Region than the global database,
# without write forwarding enabled.
aws rds create-db-cluster --global-cluster-identifier write-forwarding-test \
  --db-cluster-identifier write-forwarding-test-cluster-2 \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-west-1

aws rds create-db-instance --db-cluster-identifier write-forwarding-test-cluster-2 \
  --db-instance-identifier write-forwarding-test-cluster-2-instance-1 \
  --db-instance-class db.r5.large \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-west-1

aws rds create-db-instance --db-cluster-identifier write-forwarding-test-cluster-2 \
  --db-instance-identifier write-forwarding-test-cluster-2-instance-2 \
  --db-instance-class db.r5.large \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-west-1

aws rds modify-db-cluster --db-cluster-identifier write-forwarding-test-cluster-2 \
  --region us-east-2 \
  --enable-global-write-forwarding
```

检查辅助集群是否在 Aurora MySQL 中启用了写入转发

要确定是否可以使用辅助集群中的写入转发，可以检查集群是否具有属性 `"GlobalWriteForwardingStatus": "enabled"`。

在 AWS Management Console中，在集群详细信息页面的配置选项卡上，您可以看到全局只读副本写入转发的状态为已启用。

要查看所有集群的全局写入转发设置的状态，请运行以下 AWS CLI 命令。

辅助集群显示值 `"enabled"` 或 `"disabled"`，以指示写入转发是打开还是关闭。`null` 的值表示写入转发对该集群不可用。集群不是全局数据库的一部分，或者是主集群而不是辅助集群。如果写入转发处于打开或关闭的过程中，则该值也可能是 `"enabling"` 或 `"disabling"`。

Example

```
aws rds describe-db-clusters \
--query '*[].
{DBClusterIdentifier:DBClusterIdentifier,GlobalWriteForwardingStatus:GlobalWriteForwardingStatu

[
    {
        "GlobalWriteForwardingStatus": "enabled",
        "DBClusterIdentifier": "aurora-write-forwarding-test-replica-1"
    },
    {
        "GlobalWriteForwardingStatus": "disabled",
        "DBClusterIdentifier": "aurora-write-forwarding-test-replica-2"
    },
    {
        "GlobalWriteForwardingStatus": null,
        "DBClusterIdentifier": "non-global-cluster"
    }
]
```

要查找启用了全局写入转发功能的所有辅助集群，请运行以下命令。此命令还会返回集群的读取器终端节点。在 Aurora Global Database 中使用从辅助集群到主集群的写入转发时，可以使用辅助集群的读取器端点。

Example

```
aws rds describe-db-clusters --query 'DBClusters[].
{DBClusterIdentifier:DBClusterIdentifier,GlobalWriteForwardingStatus:GlobalWriteForwardingStatu
 | [?GlobalWriteForwardingStatus == `enabled`]'
[
    {
        "GlobalWriteForwardingStatus": "enabled",
        "ReaderEndpoint": "aurora-write-forwarding-test-replica-1.cluster-ro-
cnpexample.us-west-2.rds.amazonaws.com",
        "DBClusterIdentifier": "aurora-write-forwarding-test-replica-1"
    }
]
```

应用程序和 SQL 与 Aurora MySQL 中写入转发的兼容性

您可以将以下类型的 SQL 语句与写入转发一起使用：

- 数据操作语言 (DML) 语句，如 INSERT、DELETE 和 UPDATE。对于您可以与写入转发一起使用的这些语句的属性存在一些限制，如下所述。
- SELECT ... LOCK IN SHARE MODE 和 SELECT FOR UPDATE 语句。
- PREPARE 和 EXECUTE 语句。

在具有写入转发的全局数据库中使用某些语句时，不允许这些语句或它们可能产生过时的结果。因此，默认情况下，辅助集群的 EnableGlobalWriteForwarding 设置处于关闭状态。在启用它之前，请检查以确保您的应用程序代码不受任何这些限制的影响。

以下限制适用于与写入转发一起使用的 SQL 语句。在某些情况下，您可以对在集群级别启用写入转发的辅助集群使用语句。如果 aurora_replica_read_consistency 配置参数未在会话中打开写入转发，则此方法可用。当由于写入转发而不允许使用语句时，尝试使用语句会导致以下格式的错误消息。

```
ERROR 1235 (42000): This version of MySQL doesn't yet support 'operation with write
 forwarding'.
```

数据定义语言 (DDL)

连接到主集群以运行 DDL 语句。您不能从读取器数据库实例运行它们。

使用临时表中的数据更新永久表

您可以在启用写入转发的辅助集群上使用临时表。但是，如果语句引用临时表，则不能使用 DML 语句来修改永久表。例如，您不能使用从临时表中获取数据的 INSERT ... SELECT 语句。临时表存在于辅助集群上，当该语句在主集群上运行时不可用。

XA 事务

在会话内打开写入转发时，不能在辅助集群上使用以下语句。您可以在未启用写入转发的辅助集群上使用这些语句，或者在 aurora_replica_read_consistency 设置为空的会话中使用这些语句。在会话内启用写入转发之前，请检查您的代码是否使用这些语句。

```
XA {START|BEGIN} xid [JOIN|RESUME]
XA END xid [SUSPEND [FOR MIGRATE]]
XA PREPARE xid
XA COMMIT xid [ONE PHASE]
XA ROLLBACK xid
XA RECOVER [CONVERT XID]
```

永久表的 LOAD 语句

您不能在启用写入转发的辅助集群上使用以下语句。

```
LOAD DATA INFILE 'data.txt' INTO TABLE t1;
        LOAD XML LOCAL INFILE 'test.xml' INTO TABLE t1;
```

您可以将数据加载到辅助集群上的临时表中。但是，请确保您仅在主集群上运行任何引用永久表的 LOAD 语句。

插件语句

您不能在启用写入转发的辅助集群上使用以下语句。

```
INSTALL PLUGIN example SONAME 'ha_example.so';
UNINSTALL PLUGIN example;
```

SAVEPOINT 语句

在会话内打开写入转发时，不能在辅助集群上使用以下语句。您可以在未启用写入转发的辅助集群上使用这些语句，或者在 aurora_replica_read_consistency 设置为空白的会话中使用这些语句。在会话内启用写入转发之前，请检查您的代码是否使用这些语句。

```
SAVEPOINT t1_save;
ROLLBACK TO SAVEPOINT t1_save;
RELEASE SAVEPOINT t1_save;
```

Aurora MySQL 中写入转发的隔离和一致性

在使用写入转发的会话中，您只能使用 REPEATABLE READ 隔离级别。尽管您也可以将 READ COMMITTED 隔离级别用于辅助 AWS 区域中的只读集群，但该隔离级别不适用于写入转发。有关 REPEATABLE READ 和 READ COMMITTED 隔离级别的信息，请参阅 Aurora MySQL 隔离级别。

您可以控制辅助集群上的读取一致性程度。读取一致性级别确定辅助集群在每次读取操作之前要等待多少时间，以确保从主集群复制某些或所有更改。您可以调整读取一致性级别，以确保会话中的所有转发的写入操作在辅助集群中可见，然后再进行任何后续查询。您还可以使用此设置来确保辅助集群上的查询始终看到主集群中的最新更新，即使是由其他会话或其他集群提交的更新。要为应用程序指定此类行为，请为会话级别参数 aurora_replica_read_consistency 选择一个值。

> **Important**
>
> 始终为要转发写入的任何会话设置 aurora_replica_read_consistency 参数。若不设置，Aurora 不会为该会话启用写入转发。默认情况下，此参数的值为空，因此在使用此参数时请选择一个特定值。aurora_replica_read_consistency 参数仅对启用写入转发的辅助集群产生影响。
>
> 对于低于 3.04 的 Aurora MySQL 版本 2 和版本 3，使用 aurora_replica_read_consistency 作为会话变量。对于 Aurora MySQL 版本 3.04 及更高版本，可以使用 aurora_replica_read_consistency 作为会话变量或作为数据库集群参数。

对于 aurora_replica_read_consistency 参数，您可以指定值 EVENTUAL、SESSION 和 GLOBAL。

随着您提高一致性级别，您的应用程序会花更多时间等待在 AWS 区域之间传播更改。您可以在快速响应时间与确保在运行查询之前在其他位置进行的更改完全可用之间选择平衡。

将读取一致性设置为 EVENTUAL 后，使用写入转发的辅助 AWS 区域中的查询可能会看到由于复制滞后而稍微过时的数据。在对主区域执行写入操作并将其复制到当前区域之前，看不到同一会话中写入操作的结果。查询不会等待更新的结果可用。因此，它可能会检索较旧的数据或更新的数据，具体取决于语句的时间和复制滞后量。

将读取一致性设置为 SESSION 后，辅助 AWS 区域中使用写入转发的所有查询都会看到在该会话中所做的所有更改的结果。无论事务是否已提交，这些更改都是可见的。如有必要，查询将等待转发的写入操作的结果复制到当前区域。它不会等待在其他区域或当前区域内的其他会话中执行的写入操作的更新结果。

将读取一致性设置为 GLOBAL 后，辅助 AWS 区域中的会话会看到该会话所做的更改。它还可以查看来自主 AWS 区域和其他辅助 AWS 区域的所有已提交的更改。每个查询可能会等待一段时间，该时间取决于会话滞后量。从查询开始时，辅助集群与来自主集群的所有已提交数据处于最新状态时，查询将继续进行。

有关写入转发涉及的所有参数的更多信息，请参阅 Aurora MySQL 中写入转发的配置参数。

使用写入转发的示例

这些示例使用 aurora_replica_read_consistency 作为会话变量。对于 Aurora MySQL 版本 3.04 及更高版本，可以使用 aurora_replica_read_consistency 作为会话变量或作为数据库集群参数。

在以下示例中，主集群位于 US East (N. Virginia) 区域中。辅助集群位于 美国东部（俄亥俄州）区域中。该示例显示运行 INSERT 语句后跟 SELECT 语句的效果。根据 aurora_replica_read_consistency 设置的值，结果可能会因语句的时间而异。为了实现更高的一致性，您可以在发出 SELECT 语句之前稍等一会。或者，Aurora 可以自动等到结果复制完成后再继续进行 SELECT。

在此示例中，读取一致性设置为 eventual。在 INSERT 语句之后立即运行 SELECT 语句仍将返回 COUNT(*) 的值。此值反映插入新行之前的行数。稍后再次运行 SELECT 将返回更新的行计数。这些 SELECT 语句不会等待。

```
mysql> set aurora_replica_read_consistency = 'eventual';
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        5 |
+----------+
1 row in set (0.00 sec)
mysql> insert into t1 values (6); select count(*) from t1;
+----------+
| count(*) |
+----------+
|        5 |
+----------+
```

```
1 row in set (0.00 sec)
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        6 |
+----------+
1 row in set (0.00 sec)
```

如果读取一致性设置为 session，则紧随 INSERT 后的一条 SELECT 语句会等待，直至 INSERT 语句中的更改可见。后续 SELECT 语句不会等待。

```
mysql> set aurora_replica_read_consistency = 'session';
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        6 |
+----------+
1 row in set (0.01 sec)
mysql> insert into t1 values (6); select count(*) from t1; select count(*) from t1;
Query OK, 1 row affected (0.08 sec)
+----------+
| count(*) |
+----------+
|        7 |
+----------+
1 row in set (0.37 sec)
+----------+
| count(*) |
+----------+
|        7 |
+----------+
1 row in set (0.00 sec)
```

当读取一致性设置仍设置为 session，如果在执行一条 INSERT 语句后引入简短的等待，则会使更新的行计数在下一条 SELECT 语句运行时可用。

```
mysql> insert into t1 values (6); select sleep(2); select count(*) from t1;
Query OK, 1 row affected (0.07 sec)
+----------+
| sleep(2) |
```

```
+----------+
|        0 |
+----------+
1 row in set (2.01 sec)
+----------+
| count(*) |
+----------+
|        8 |
+----------+
1 row in set (0.00 sec)
```

如果读取一致性设置为 global 时，每个 SELECT 语句将等待，以确保在执行该查询之前，该语句开始时的所有数据更改均可见。每个 SELECT 语句的等待量会有所不同，具体取决于主集群和辅助集群之间的复制滞后量。

```
mysql> set aurora_replica_read_consistency = 'global';
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        8 |
+----------+
1 row in set (0.75 sec)
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        8 |
+----------+
1 row in set (0.37 sec)
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        8 |
+----------+
1 row in set (0.66 sec)
```

使用 Aurora MySQL 中的写入转发运行多部分语句

DML 语句可能由多个部分组成，如 INSERT ... SELECT 语句或 DELETE ... WHERE 语句。在这种情况下，整个语句将转发到主集群并在此处运行。

使用 Aurora MySQL 中写入转发执行的事务

事务是否转发到主集群取决于事务的访问模式。您可以使用 SET TRANSACTION 语句或 START TRANSACTION 语句指定事务的访问模式。您还可以通过更改 transaction_read_only 会话变量的值来指定事务访问模式。只有在连接到启用了写入转发的数据库集群时，才能更改此会话值。

如果长时间运行的事务在很长一段时间内没有发出任何语句，则可能会超过空闲超时期限。此时段的默认值为一分钟。您最多可以将其增加到一天。超过空闲超时的事务将被主集群取消。您提交的下一个后续语句会收到超时错误。然后 Aurora 回滚事务。

在写入转发变得不可用的其他情况下，可能会发生此类错误。例如，如果重新启动主集群或关闭写入转发配置设置，则 Aurora 取消使用写入转发的任何事务。

Aurora MySQL 中写入转发的配置参数

Aurora 集群参数组包含写入转发功能的设置。由于这些是集群参数，因此每个集群中的所有数据库实例对于这些变量具有相同的值。下表汇总了有关这些参数的详细信息，表后附有使用说明。

名称	范围	类型	默认值	有效值
aurora_fwd_master_idle_timeout （Aurora MySQL 版本 2）	全局	无符号整数	60	1–86,400
aurora_fwd_master_max_connections_pct （Aurora MySQL 版本 2）	全局	无符号长整数	10	0–90
aurora_fwd_writer_idle_timeout （Aurora MySQL 版本 3）	全局	无符号整数	60	1–86,400
aurora_fwd_writer_max_connections_pct （Aurora MySQL 版本 3）	全局	无符号长整数	10	0–90
aurora_replica_read_consistency	对于版本 2 和版本 3（低于版本 3.04）为会话，对于版本	枚举	"（null）	EVENTUAL, SESSION, GLOBAL

名称	范围	类型	默认值	有效值
	3.04 及更高版本为全局			

要控制来自辅助集群的传入写入请求，请在主集群上使用以下设置：

- aurora_fwd_master_idle_timeout、aurora_fwd_writer_idle_timeout：主集群在关闭连接之前从辅助集群转发的连接上等待某个活动的秒数。如果会话在此期间之后仍处于空闲状态，则 Aurora 取消会话。
- aurora_fwd_master_max_connections_pct、aurora_fwd_writer_max_connections_pct：可以在写入器数据库实例上用于处理从读取器转发的查询的数据库连接的上限。它表示为主集群中写入器数据库实例的 max_connections 设置的百分比。例如，如果 max_connections 是 800，而 aurora_fwd_master_max_connections_pct 或 aurora_fwd_writer_max_connections_pct 是 10，则写入器最多允许 80 个同时转发会话。这些连接来自由 max_connections 设置管理的同一连接池。

  当一个或多个辅助集群启用了写入转发时，此设置仅适用于主集群。如果减小该值，不会影响现有连接。Aurora 尝试从辅助集群创建新连接时，会考虑帐户设置的新值。默认值为 10，表示 max_connections 值的 10%。如果在任何辅助集群上启用查询转发，则此设置必须具有非零值才能成功地从辅助集群执行写入操作。如果值为零，则写入操作会收到错误代码 ER_CON_COUNT_ERROR 以及消息 Not enough connections on writer to handle your request。

aurora_replica_read_consistency 参数启用写入转发。每个会话中都需使用该参数。您可将读取一致性级别指定为 EVENTUAL、SESSION 或 GLOBAL。要了解有关一致性级别的更多信息，请参阅Aurora MySQL 中写入转发的隔离和一致性。以下规则适用于此参数：

- 默认值是 "（空）。
- 仅当 aurora_replica_read_consistency 设置为 EVENTUAL、SESSION 或 GLOBAL 时，写入转发才可用。此参数仅适用于启用了写入转发且位于 Aurora Global Database 中的辅助集群的读取器实例。
- 您不能在多语句事务内部设置此变量（如果为空）或取消设置（如果已设置）。但是，在此类事务期间，您可以将其从一个有效值（EVENTUAL、SESSION 或 GLOBAL）更改为另一个有效值（EVENTUAL、SESSION 或 GLOBAL）。

- 当辅助集群上未启用写入转发时，变量不能为 SET。

Aurora MySQL 中写入转发的 Amazon CloudWatch 指标

在一个或多个辅助集群上使用写入转发时，以下 Amazon CloudWatch 指标适用于主集群。这些指标均在主集群中的写入器数据库实例上进行测量。

CloudWatch 指标	单位	描述
AuroraDMLRejectedMasterFull	计数	由于写入器数据库实例上会话已满而遭拒绝的转发查询数。 对于 Aurora MySQL 版本 2。
AuroraDMLRejectedWriterFull	计数	由于写入器数据库实例上会话已满而遭拒绝的转发查询数。 对于 Aurora MySQL 版本 3。
ForwardingMasterDMLLatency	毫秒	在写入器数据库实例上处理每个转发的 DML 语句的平均时间。 它不包括辅助集群转发写入请求的时间，也不包括将更改复制回辅助集群的时间。 对于 Aurora MySQL 版本 2。
ForwardingMasterDMLThroughput	每秒计数	此写入器数据库实例每秒处理的转发 DML 语句数。 对于 Aurora MySQL 版本 2。
ForwardingMasterOpenSessions	计数	写入器数据库实例上的转发会话数。 对于 Aurora MySQL 版本 2。

CloudWatch 指标	单位	描述
ForwardingWriterDMLLatency	毫秒	在写入器数据库实例上处理每个转发的 DML 语句的平均时间。 它不包括辅助集群转发写入请求的时间，也不包括将更改复制回辅助集群的时间。 对于 Aurora MySQL 版本 3。
ForwardingWriterDMLThroughput	每秒计数	此写入器数据库实例每秒处理的转发 DML 语句数。 对于 Aurora MySQL 版本 3。
ForwardingWriterOpenSessions	计数	写入器数据库实例上的转发会话数。 对于 Aurora MySQL 版本 3。

以下 CloudWatch 指标适用于每个辅助集群。这些指标在启用写入转发的辅助集群中的每个读取器数据库实例上进行测量。

CloudWatch 指标	单位	描述
ForwardingReplicaDMLLatency	毫秒	副本上转发 DML 的平均响应时间。
ForwardingReplicaDMLThroughput	每秒计数	每秒处理的转发 DML 语句数。
ForwardingReplicaOpenSessions	计数	在读取器数据库实例上使用写入转发的会话数。
ForwardingReplicaReadWaitLatency	毫秒	读取器数据库实例上的 SELECT 语句等待赶上主集群的平均等待时间。

CloudWatch 指标	单位	描述
		读取器数据库实例在处理查询之前等待的程度取决于 aurora_replica_read_consistency 设置。
ForwardingReplicaReadWaitThroughput	每秒计数	转发写入的所有会话中每秒处理的 SELECT 语句总数。
ForwardingReplicaSelectLatency	毫秒	已转发的 SELECT 延迟，监控期间内所有转发 SELECT 语句的平均值。
ForwardingReplicaSelectThroughput	每秒计数	已转发 SELECT 吞吐量，监控期内每秒平均值。

用于写入转发的 Aurora MySQL 状态变量

当您在一个或多个辅助集群上使用写入转发时，以下 Aurora MySQL 状态变量适用于主集群。这些指标均在主集群中的写入器数据库实例上进行测量。

Aurora MySQL 状态变量	单位	描述
Aurora_fwd_master_dml_stmt_count	计数	转发到此写入器数据库实例的 DML 语句总数。 对于 Aurora MySQL 版本 2。
Aurora_fwd_master_dml_stmt_duration	微秒	转发到此写入器数据库实例的 DML 语句的总持续时间。 对于 Aurora MySQL 版本 2。
Aurora_fwd_master_open_sessions	计数	写入器数据库实例上的转发会话数。 对于 Aurora MySQL 版本 2。

Aurora MySQL 状态变量	单位	描述
Aurora_fwd_master_select_stmt_count	计数	转发到此写入器数据库实例的 SELECT 语句总数。 对于 Aurora MySQL 版本 2。
Aurora_fwd_master_select_stmt_duration	微秒	转发到此写入器数据库实例的 SELECT 语句的总持续时间。 对于 Aurora MySQL 版本 2。
Aurora_fwd_writer_dml_stmt_count	计数	转发到此写入器数据库实例的 DML 语句总数。 对于 Aurora MySQL 版本 3。
Aurora_fwd_writer_dml_stmt_duration	微秒	转发到此写入器数据库实例的 DML 语句的总持续时间。
Aurora_fwd_writer_open_sessions	计数	写入器数据库实例上的转发会话数。 对于 Aurora MySQL 版本 3。
Aurora_fwd_writer_select_stmt_count	计数	转发到此写入器数据库实例的 SELECT 语句总数。 对于 Aurora MySQL 版本 3。
Aurora_fwd_writer_select_stmt_duration	微秒	转发到此写入器数据库实例的 SELECT 语句的总持续时间。 对于 Aurora MySQL 版本 3。

以下 Aurora MySQL 状态变量适用于每个辅助集群。这些指标在启用写入转发的辅助集群中的每个读取器数据库实例上进行测量。

Aurora MySQL 状态变量	单位	描述
Aurora_fwd_replica_dml_stmt_count	计数	从此读取器数据库实例转发的 DML 语句总数。
Aurora_fwd_replica_dml_stmt_duration	微秒	从此读取器数据库实例转发的所有 DML 语句的总持续时间。
Aurora_fwd_replica_errors_session_limit	计数	由于以下错误条件之一而被主集群拒绝的会话数。 • 写入器已满 • 正在转发的语句过多。
Aurora_fwd_replica_open_sessions	计数	在读取器数据库实例上使用写入转发的会话数。
Aurora_fwd_replica_read_wait_count	计数	此读取器数据库实例上的写入后读取等待的总数。
Aurora_fwd_replica_read_wait_duration	微秒	由于此读取器数据库实例上的读取一致性设置而导致的总等待持续时间。
Aurora_fwd_replica_select_stmt_count	计数	从此读取器数据库实例转发的 SELECT 语句总数。
Aurora_fwd_replica_select_stmt_duration	微秒	从此读取器数据库实例转发的 SELECT 语句的总持续时间。

在 Aurora PostgreSQL 全局数据库中使用写入转发

主题

- Aurora PostgreSQL 中写入转发的区域和版本可用性
- 在 Aurora PostgreSQL 中启用写入转发
- 检查辅助集群是否在 Aurora PostgreSQL 中启用了写入转发

- 应用程序和 SQL 与 Aurora PostgreSQL 中写入转发的兼容性
- Aurora PostgreSQL 中写入转发的隔离和一致性
- 具有写入转发的事务访问模式
- 使用 Aurora PostgreSQL 中的写入转发运行多部分语句
- Aurora PostgreSQL 中写入转发的配置参数
- Aurora PostgreSQL 中写入转发的 Amazon CloudWatch 指标
- 在 Aurora PostgreSQL 中使用写入转发的等待事件

Aurora PostgreSQL 中写入转发的区域和版本可用性

在 Aurora PostgreSQL 版本 16 及更高的主要版本中，所有次要版本都支持全局写入转发。对于更早的 Aurora PostgreSQL 版本，版本 15.4 及更高的次要版本和版本 14.9 及更高的次要版本支持全局写入转发。写入转发在提供基于 Aurora PostgreSQL 的全局数据库的每个 AWS 区域均可用。

有关 Aurora PostgreSQL 全局数据库的版本和区域可用性的更多信息，请参阅使用 Aurora PostgreSQL 的 Aurora 全局数据库。

在 Aurora PostgreSQL 中启用写入转发

默认情况下，在将辅助集群添加到 Aurora Global Database 时不启用写入转发。您可以在创建辅助数据库集群时或创建后的任何时候为其启用写入转发。如果需要，您可以稍后将其禁用。启用或禁用写入转发不会导致停机或重启。

> **Note**
>
> 对于偶尔写入且要求先写后读一致性（即能够读取事务中的最新写入内容）的应用程序，您可以使用本地写入转发。

控制台

在控制台中，您可以在创建或修改辅助数据库集群时启用或禁用写入转发。

创建辅助数据库集群时启用或禁用写入转发

创建新的辅助数据库集群时，您可以通过选中只读副本写入转发下方的开启全局写入转发复选框来启用写入转发。或者清除复选框即可将其禁用。要创建辅助数据库集群，请按 创建 Amazon Aurora 数据库集群 中数据库引擎的说明操作。

以下屏幕截图显示已选中开启全局写入转发复选框的只读副本写入转发部分。



修改辅助数据库集群时启用或禁用写入转发

在控制台中，您可以修改辅助数据库集群以启用或禁用写入转发。

使用控制台为辅助数据库集群启用或禁用写入转发

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 选择数据库。
3. 选择辅助数据库集群，然后选择修改。
4. 在只读副本写入转发部分，选中或清除开启全局写入转发复选框。
5. 选择继续。
6. 对于计划修改，选择立即应用。如果您选择在下一个计划的维护时段内应用，则 Aurora 将忽略此设置并立即开启写入转发。
7. 选择修改集群。

AWS CLI

要使用 AWS CLI 启用写入转发，请使用 `--enable-global-write-forwarding` 选项。当您使用 create-db-cluster 命令创建新的辅助集群时，此选项将起作用。当您使用 modify-db-cluster 命令修改现有辅助集群时，它也可以起作用。它要求全局数据库使用支持写入转发的 Aurora 版本。您可以结合这些相同的 CLI 命令使用 `--no-enable-global-write-forwarding` 选项来禁用写入转发。

以下过程介绍如何使用 AWS CLI 为全局集群中的辅助数据库集群启用或禁用写入转发。

为现有辅助数据库集群启用或禁用写入转发

- 调用 modify-db-cluster AWS CLI 命令并提供以下值：
  - `--db-cluster-identifier` – 数据库集群的名称。
  - `--enable-global-write-forwarding` 以打开或 `--no-enable-global-write-forwarding` 以关闭。

以下示例为数据库集群 sample-secondary-db-cluster 启用写入转发。

对于 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster \
    --db-cluster-identifier sample-secondary-db-cluster \
    --enable-global-write-forwarding
```

对于 Windows：

```
aws rds modify-db-cluster ^
    --db-cluster-identifier sample-secondary-db-cluster ^
    --enable-global-write-forwarding
```

RDS API

要使用 Amazon RDS API 启用写入转发，请将 EnableGlobalWriteForwarding 参数设置为 true。当您使用 CreateDBCluster 操作创建新的辅助集群时，此参数起作用。当您使用 ModifyDBCluster 操作修改现有辅助集群时，它也起作用。它要求全局数据库使用支持写入转发的 Aurora 版本。您可以通过将 EnableGlobalWriteForwarding 参数设置为 false 来禁用写入转发。

检查辅助集群是否在 Aurora PostgreSQL 中启用了写入转发

要确定是否可以使用辅助集群中的写入转发，可以检查集群是否具有属性 "GlobalWriteForwardingStatus": "enabled"。

在 AWS Management Console中，您可以在集群的详细信息页面的配置选项卡上看到 Read replica write forwarding。要查看所有集群的全局写入转发设置的状态，请运行以下 AWS CLI 命令。

辅助集群显示值 "enabled" 或 "disabled"，以指示写入转发是打开还是关闭。null 的值表示写入转发对该集群不可用。集群不是全局数据库的一部分，或者是主集群而不是辅助集群。如果写入转发处于打开或关闭的过程中，则该值也可能是 "enabling" 或 "disabling"。

Example

```
aws rds describe-db-clusters --query '*[].
{DBClusterIdentifier:DBClusterIdentifier,GlobalWriteForwardingStatus:GlobalWriteForwardingStatu
```

```
[
    {
        "GlobalWriteForwardingStatus": "enabled",
        "DBClusterIdentifier": "aurora-write-forwarding-test-replica-1"
    },
    {
        "GlobalWriteForwardingStatus": "disabled",
        "DBClusterIdentifier": "aurora-write-forwarding-test-replica-2"
    },
    {
        "GlobalWriteForwardingStatus": null,
        "DBClusterIdentifier": "non-global-cluster"
    }
]
```

要查找启用了全局写入转发功能的所有辅助集群，请运行以下命令。此命令还会返回集群的读取器终端节点。在 Aurora Global Database 中使用从辅助集群到主集群的写入转发时，可以使用辅助集群的读取器端点。

Example

```
aws rds describe-db-clusters --query 'DBClusters[].
{DBClusterIdentifier:DBClusterIdentifier,GlobalWriteForwardingStatus:GlobalWriteForwardingStatu
 | [?GlobalWriteForwardingStatus == `enabled`]'
[
    {
        "GlobalWriteForwardingStatus": "enabled",
        "ReaderEndpoint": "aurora-write-forwarding-test-replica-1.cluster-ro-
cnpexample.us-west-2.rds.amazonaws.com",
        "DBClusterIdentifier": "aurora-write-forwarding-test-replica-1"
    }
]
```

应用程序和 SQL 与 Aurora PostgreSQL 中写入转发的兼容性

在具有写入转发的全局数据库中使用某些语句时，不允许这些语句或它们可能产生过时的结果。此外，不支持用户定义的函数和用户定义的过程。因此，默认情况下，辅助集群的 `EnableGlobalWriteForwarding` 设置处于关闭状态。在启用它之前，请检查以确保您的应用程序代码不受任何这些限制的影响。

您可以将以下类型的 SQL 语句与写入转发一起使用：

- 数据操作语言（DML）语句，如 INSERT、DELETE 和 UPDATE。
- SELECT FOR { UPDATE | NO KEY UPDATE | SHARE | KEY SHARE } 语句
- PREPARE 和 EXECUTE 语句
- EXPLAIN 语句和此列表中语句

写入转发不支持以下类型的 SQL 语句：

- 数据定义语言（DDL）语句
- ANALYZE
- CLUSTER
- COPY
- 游标 - 不支持游标，因此在使用写入转发之前，请务必将其关闭。
- GRANT|REVOKE|REASSIGN OWNED|SECURITY LABEL
- LOCK
- SAVEPOINT 语句
- SELECT INTO
- SET CONSTRAINTS
- TRUNCATE
- VACUUM

Aurora PostgreSQL 中写入转发的隔离和一致性

在使用写入转发的会话中，您可以使用 REPEATABLE READ 和 READ COMMITTED 隔离级别。但是，不支持 SERIALIZABLE 隔离级别。

您可以控制辅助集群上的读取一致性程度。读取一致性级别确定辅助集群在每次读取操作之前要等待多少时间，以确保从主集群复制某些或所有更改。您可以调整读取一致性级别，以确保会话中的所有转发的写入操作在辅助集群中可见，然后再进行任何后续查询。您还可以使用此设置来确保辅助集群上的查询始终看到主集群中的最新更新，即使是由其他会话或其他集群提交的更新。要为应用程序指定此类行为，请为 apg_write_forward.consistency_mode 参数选择一个合适的值。apg_write_forward.consistency_mode 参数仅对启用写入转发的辅助集群产生影响。

> **Note**
>
> 对于 apg_write_forward.consistency_mode 参数，您可以指定值 SESSION、EVENTUAL、GLOBAL 和 OFF。默认情况下，值设为 SESSION。将该值设置为 OFF 将在会话中禁用写入转发。

随着您提高一致性级别，您的应用程序会花更多时间等待在 AWS 区域之间传播更改。您可以在快速响应时间与确保在运行查询之前在其他位置进行的更改完全可用之间选择平衡。

对于每种可用的一致性模式设置，效果如下所示：

- SESSION – 辅助 AWS 区域中使用写入转发的所有查询都会看到在该会话中所做的所有更改的结果。无论事务是否已提交，这些更改都是可见的。如有必要，查询将等待转发的写入操作的结果复制到当前区域。它不会等待在其他区域或当前区域内的其他会话中执行的写入操作的更新结果。
- EVENTUAL – 使用写入转发的辅助 AWS 区域中的查询可能会看到由于复制滞后而稍微过时的数据。在对主区域执行写入操作并将其复制到当前区域之前，看不到同一会话中写入操作的结果。查询不会等待更新的结果可用。因此，它可能会检索较旧的数据或更新的数据，具体取决于语句的时间和复制滞后量。
- GLOBAL – 辅助 AWS 区域中的会话会看到该会话所做的更改。它还可以查看来自主 AWS 区域和其他辅助 AWS 区域的所有已提交的更改。每个查询可能会等待一段时间，该时间取决于会话滞后量。从查询开始时，辅助集群与来自主集群的所有已提交数据处于最新状态时，查询将继续进行。
- OFF – 在会话中禁用写入转发。

有关写入转发涉及的所有参数的更多信息，请参阅 Aurora PostgreSQL 中写入转发的配置参数。

具有写入转发的事务访问模式

如果事务访问模式设置为只读，则不使用写入转发。在连接到启用了写入转发的数据库集群和会话时，可以将访问模式设置为只读写。

有关事务访问模式的更多信息，请参阅 SET TRANSACTION。

使用 Aurora PostgreSQL 中的写入转发运行多部分语句

DML 语句可能由多个部分组成，如 INSERT ... SELECT 语句或 DELETE ... WHERE 语句。在这种情况下，整个语句将转发到主集群并在此处运行。

Aurora PostgreSQL 中写入转发的配置参数

Aurora 集群参数组包含写入转发功能的设置。由于这些是集群参数，因此每个集群中的所有数据库实例对于这些变量具有相同的值。下表汇总了有关这些参数的详细信息，表后附有使用说明。

名称	范围	类型	默认值	有效值
apg_write_forward.connect_timeout	会话	秒	30	0–2147483647
apg_write_forward.consistency_mode	Session	enum	会话	SESSION, EVENTUAL, GLOBAL, OFF
apg_write_forward.idle_in_transaction_session_timeout	会话	毫秒	86400000	0–2147483647
apg_write_forward.idle_session_timeout	会话	毫秒	300000	0–2147483647
apg_write_forward.max_forwarding_connections_percent	全局	整数	25	1–100

apg_write_forward.max_forwarding_connections_percent 参数是可以用于处理从读取器转发的查询的数据库连接插槽上限。它表示为主集群中写入器数据库实例的 max_connections 设置的百分比。例如，如果 max_connections 是 800，而 apg_write_forward.max_forwarding_connections_percent 是 10，则写入器最多允许 80 个同时转发会话。这些连接来自由 max_connections 设置管理的同一连接池。当至少一个辅助集群启用了写入转发时，此设置仅适用于主集群。

在辅助集群上使用以下设置：

- apg_write_forward.consistency_mode – 会话级参数，用于控制辅助集群上的读取一致性程度。有效值为 SESSION、EVENTUAL、GLOBAL 或 OFF。默认情况下，值设为 SESSION。将该值设置为 OFF 将在会话中禁用写入转发。要了解有关一致性级别的更多信息，请参阅Aurora PostgreSQL 中写入转发的隔离和一致性。此参数仅适用于启用了写入转发且位于 Aurora Global Database 中的辅助集群的读取器实例。

- apg_write_forward.connect_timeout – 辅助集群在与主集群建立连接时在放弃之前等待的最大秒数。值 0 表示无限期等待。
- apg_write_forward.idle_in_transaction_session_timeout – 主集群在关闭从具有未处理事务的辅助集群转发的连接之前等待活动的毫秒数。如果在此段时间之后，会话在事务中仍处于空闲状态，则 Aurora 会终止会话。0 值禁用超时。
- apg_write_forward.idle_session_timeout – 主集群在关闭从辅助集群转发的连接之前等待活动的毫秒数。如果会话在这段时间之后仍处于空闲状态，则 Aurora 会终止会话。值 0 禁用超时。

Aurora PostgreSQL 中写入转发的 Amazon CloudWatch 指标

在一个或多个辅助集群上使用写入转发时，以下 Amazon CloudWatch 指标适用于主集群。这些指标均在主集群中的写入器数据库实例上进行测量。

CloudWatch 指标	单位和描述
AuroraForwardingWriterDMLThroughput	计数（每秒）。此写入器数据库实例每秒处理的转发 DML 语句数。
AuroraForwardingWriterOpenSessions	计数。此写入器数据库实例上处理转发查询的打开会话数。
AuroraForwardingWriterTotalSessions	计数。此写入器数据库实例上的转发会话总数。

以下 CloudWatch 指标适用于每个辅助集群。这些指标在启用写入转发的辅助集群中的每个读取器数据库实例上进行测量。

CloudWatch 指标	单位和描述
AuroraForwardingReplicaCommitThroughput	计数（每秒）。此副本每秒转发的会话中的提交数。
AuroraForwardingReplicaDMLLatency	毫秒。副本上转发 DML 的平均响应时间（以毫秒为单位）。

CloudWatch 指标	单位和描述
AuroraForwardingReplicaDMLThroughput	计数（每秒）。此副本每秒处理的转发 DML 语句数。
AuroraForwardingReplicaErrorSessionsLimit	计数。由于达到最大连接数或最大写入转发连接数限制而被主集群拒绝的会话数。
AuroraForwardingReplicaOpenSessions	计数。在副本实例上使用写入转发的会话数。
AuroraForwardingReplicaReadWaitLatency	毫秒。副本等待与主集群 LSN 一致的平均等待时间（以毫秒为单位）。读取器数据库实例等待的程度取决于 apg_write_forward.consistency_mode 设置。有关该设置的信息，请参阅 the section called “Aurora PostgreSQL 中写入转发的配置参数”。

在 Aurora PostgreSQL 中使用写入转发的等待事件

当您在 Aurora PostgreSQL 中使用写入转发时，Amazon Aurora 会生成以下等待事件。

主题

- IPC:AuroraWriteForwardConnect
- IPC:AuroraWriteForwardConsistencyPoint
- IPC:AuroraWriteForwardExecute
- IPC:AuroraWriteForwardGetGlobalConsistencyPoint
- IPC:AuroraWriteForwardXactAbort
- IPC:AuroraWriteForwardXactCommit
- IPC:AuroraWriteForwardXactStart

IPC:AuroraWriteForwardConnect

当辅助数据库集群上的后端进程在等待开启与主数据库集群写入器节点的连接时，就会发生该 IPC:AuroraWriteForwardConnect 事件。

等待次数增加的可能原因

随着尝试从辅助区域读取器节点连接到主数据库集群写入器节点的次数不断增加，此事件也会增加。

操作

减少从辅助节点到主区域写入器节点的同时连接数量。

IPC:AuroraWriteForwardConsistencyPoint

该 IPC:AuroraWriteForwardConsistencyPoint 事件描述在将转发写入操作的结果复制到当前区域之前，来自辅助数据库集群上节点的查询将等待的时间。仅当会话级别的参数 apg_write_forward.consistency_mode 设置为以下项之一时，才会生成此事件：

- SESSION – 辅助节点上的查询等待在该会话中所做的所有更改的结果。
- GLOBAL – 辅助节点上的查询等待该会话中所做更改以及全局集群中主区域和其它辅助区域中已提交的所有更改的结果。

有关 apg_write_forward.consistency_mode 参数设置的更多信息，请参阅 the section called “Aurora PostgreSQL 中写入转发的配置参数”。

等待次数增加的可能原因

等待时间较长的常见原因包括以下几点：

- 副本滞后延长，如 Amazon CloudWatch ReplicaLag 指标所衡量。有关该指标的更多信息，请参阅监控 Aurora PostgreSQL 复制。
- 增加了主区域写入器节点或辅助节点上的负载。

操作

根据应用程序的要求更改一致性模式。

IPC:AuroraWriteForwardExecute

当辅助数据库集群上的后端进程在等待转发查询完成并从主数据库集群写入器节点获取结果时，就会发生该 IPC:AuroraWriteForwardExecute 事件。

等待次数增加的可能原因

等待次数增加的常见原因包括以下几点：

- 从主区域的写入器节点获取大量行。
- 辅助节点和主区域写入器节点之间的网络延迟增加会增加辅助节点从写入器节点获得数据所花费的时间。
- 辅助节点的负载增加可能会延迟从辅助节点向主区域写入器节点传输查询请求的时间。
- 主区域写入器节点负载的增加可能会延迟从写入器节点向辅助节点传输数据的时间。

操作

根据等待事件的原因，我们建议采取不同的操作。

- 优化查询以仅检索必要的数据。
- 优化数据操纵语言（DML）操作，使其仅修改必要数据。
- 如果辅助节点或主区域写入器节点受到 CPU 或网络带宽的限制，请考虑将其更改为具有更大 CPU 容量或更多网络带宽的实例类型。

IPC:AuroraWriteForwardGetGlobalConsistencyPoint

当使用 GLOBAL 一致性模式的辅助数据库集群上的后端进程在执行查询之前等待从写入器节点获取全局一致性点时，就会发生该 `IPC:AuroraWriteForwardGetGlobalConsistencyPoint` 事件。

等待次数增加的可能原因

等待次数增加的常见原因包括以下几点：

- 辅助节点和主区域写入器节点之间的网络延迟增加会增加辅助节点从写入器节点获得数据所花费的时间。
- 辅助节点的负载增加可能会延迟从辅助节点向主区域写入器节点传输查询请求的时间。
- 主区域写入器节点负载的增加可能会延迟从写入器节点向辅助节点传输数据的时间。

操作

根据等待事件的原因，我们建议采取不同的操作。

- 根据应用程序的要求更改一致性模式。

- 如果辅助节点或主区域写入器节点受到 CPU 或网络带宽的限制，请考虑将其更改为具有更大 CPU 容量或更多网络带宽的实例类型。

IPC:AuroraWriteForwardXactAbort

当辅助数据库集群上的后端进程在等待远程清理查询的结果时，就会发生该 `IPC:AuroraWriteForwardXactAbort` 事件。在写入转发的事务中止后，会发出清理查询，以将进程恢复到适当的状态。Amazon Aurora 之所以执行这些操作，要么是因为发现了错误，要么是因为用户发出了明确的 `ABORT` 命令或取消了正在运行的查询。

等待次数增加的可能原因

等待次数增加的常见原因包括以下几点：

- 辅助节点和主区域写入器节点之间的网络延迟增加会增加辅助节点从写入器节点获得数据所花费的时间。
- 辅助节点的负载增加可能会延迟从辅助节点向主区域写入器节点传输清理查询请求的时间。
- 主区域写入器节点负载的增加可能会延迟从写入器节点向辅助节点传输数据的时间。

操作

根据等待事件的原因，我们建议采取不同的操作。

- 调查事务中止的原因。
- 如果辅助节点或主区域写入器节点受到 CPU 或网络带宽的限制，请考虑将其更改为具有更大 CPU 容量或更多网络带宽的实例类型。

IPC:AuroraWriteForwardXactCommit

当辅助数据库集群上的后端进程在等待转发提交事务命令的结果时，就会发生该 `IPC:AuroraWriteForwardXactCommit` 事件。

等待次数增加的可能原因

等待次数增加的常见原因包括以下几点：

- 辅助节点和主区域写入器节点之间的网络延迟增加会增加辅助节点从写入器节点获得数据所花费的时间。
- 辅助节点的负载增加可能会延迟从辅助节点向主区域写入器节点传输查询请求的时间。

- 主区域写入器节点负载的增加可能会延迟从写入器节点向辅助节点传输数据的时间。

操作

如果辅助节点或主区域写入器节点受到 CPU 或网络带宽的限制，请考虑将其更改为具有更大 CPU 容量或更多网络带宽的实例类型。

IPC:AuroraWriteForwardXactStart

当辅助数据库集群上的后端进程在等待转发开始事务命令的结果时，就会发生该 `IPC:AuroraWriteForwardXactStart` 事件。

等待次数增加的可能原因

等待次数增加的常见原因包括以下几点：

- 辅助节点和主区域写入器节点之间的网络延迟增加会增加辅助节点从写入器节点获得数据所花费的时间。
- 辅助节点的负载增加可能会延迟从辅助节点向主区域写入器节点传输查询请求的时间。
- 主区域写入器节点负载的增加可能会延迟从写入器节点向辅助节点传输数据的时间。

操作

如果辅助节点或主区域写入器节点受到 CPU 或网络带宽的限制，请考虑将其更改为具有更大 CPU 容量或更多网络带宽的实例类型。

在 Amazon Aurora Global Database 中使用切换或失效转移

与 Aurora 数据库集群在单个 AWS 区域中提供的标准高可用性相比，Aurora Global Database 特征提供了更高的业务连续性和灾难恢复（BCDR）保护。通过使用 Aurora Global Database，您可以计划从罕见、计划外的区域性灾难或完全的服务级别中断中更快地恢复。

您可以参阅以下指南和程序，使用 Aurora Global Database 特征来计划、测试和实施 BCDR 策略。

主题

- 计划业务连续性和灾难恢复
- 对 Amazon Aurora Global Database 执行切换
- 从计划外停机中恢复 Amazon Aurora Global Database
- 管理基于 Aurora PostgreSQL 的全局数据库的 RPO

- 全局数据库辅助集群的跨区域弹性

计划业务连续性和灾难恢复

要计划业务连续性和灾难恢复策略，了解以下行业术语以及这些术语与 Aurora Global Database 特征的关系会很有帮助。

从灾难中恢复通常由以下两个业务目标驱动：

- 恢复时间目标（RTO）– 灾难或服务中断后系统恢复工作状态所需的时间。换言之，RTO 用于衡量停机时间。对于 Aurora Global Database，RTO 大约为数分钟。
- 恢复点目标（RPO）– 灾难或服务中断后可能丢失的数据量（按时间衡量）。这种数据丢失通常是由于异步复制滞后造成的。对于 Aurora 全球数据库，RPO 通常以秒为单位进行测量。通过基于 Aurora PostgreSQL 的全局数据库，您可以使用 `rds.global_db_rpo` 参数设置和跟踪 RPO 上限，但这样做可能会影响主集群写入器节点上的事务处理。有关更多信息，请参阅 管理基于 Aurora PostgreSQL 的全局数据库的 RPO。

使用 Aurora Global Database 执行切换或失效转移时，需要将辅助数据库集群提升为主数据库集群。“区域性停机”一词通常用于描述各种故障情况。最坏的情况可能是影响数百平方英里的灾难性事件造成的广泛停电。但是，大多数停机的局部程度会高得多，仅影响一小部分云服务或客户系统。考虑停机的全部范围，以确保跨区域失效转移是正确的解决方案，并针对这种情况选择适当的失效转移方法。应使用失效转移还是切换方法取决于具体的停机情况：

- 失效转移 – 使用此方法从计划外停机中恢复。使用这种方法，您可以跨区域失效转移到 Aurora 全球数据库中的一个辅助数据库集群。这种方法的 RPO 通常是一个以秒为单位的非零值。数据丢失量取决于发生故障时跨 AWS 区域的 Aurora 全球数据库复制滞后。要了解更多信息，请参阅从计划外停机中恢复 Amazon Aurora Global Database。
- 切换 – 此操作以前称为“托管式计划内失效转移”。将此方法用于受控场景，例如操作维护和其他计划内操作过程，其中所有 Aurora 集群以及与之交互的其他服务都处于正常状态。由于此特征会在进行任何其他更改之前将辅助数据库集群与主数据库集群同步，因此 RPO 为 0（不会造成数据丢失）。要了解更多信息，请参阅对 Amazon Aurora Global Database 执行切换。

> **Note**
>
> 切换或失效转移到无外设辅助 Aurora 数据库集群之前，必须先向其添加一个数据库实例。有关无管控数据库集群的更多信息，请参阅在辅助区域中创建无管控 Aurora 数据库集群。

对 Amazon Aurora Global Database 执行切换

> **Note**
>
> 切换以前称为托管式计划内失效转移。

通过使用切换，您可以定期更改主集群的区域。此方法适用于受控场景，例如操作维护和其他计划内操作过程。

切换有三种常见使用案例。

- 适用于对特定行业施加的“区域轮换”要求。例如，金融服务法规可能要求第 0 层系统在几个月内切换到不同的区域，以确保定期执行灾难恢复过程。
- 适用于多区域“全天候”应用程序。例如，一家企业可能希望根据不同时区的工作时间在不同区域提供延迟更低的写入。
- 作为一种零数据丢失方法，可在失效转移后失效自动恢复到原始主区域。

> **Note**
>
> 切换旨在用于 Aurora Global Database，其中所有 Aurora 集群以及与之交互的其他服务都处于正常状态。要从计划外停机中进行恢复，请按照从计划外停机中恢复 Amazon Aurora Global Database中的相应过程操作。
>
> 仅当主数据库集群和辅助数据库集群具有相同的主要和次要引擎版本时，您才能对 Aurora Global Database 执行托管式跨区域切换。根据引擎和引擎版本，补丁级别可能需要相同，也可以不同。有关允许在具有不同补丁级别的主集群和辅助集群之间进行这些操作的引擎和引擎版本列表，请参阅托管式跨区域切换和失效转移的补丁级别兼容性。在开始切换之前，请检查全球集群中的引擎版本，以确保它们支持托管式跨区域切换，并在需要时对其进行升级。

在切换期间，Aurora 会使您选择的辅助区域中的集群成为主集群。切换机制维护全局数据库的现有复制拓扑：它在相同区域中仍有相同数量的 Aurora 集群。在 Aurora 开始切换过程之前，它会等待目标辅助区域集群与主区域集群完全同步。然后，主区域中的数据库集群变为只读状态。所选的辅助集群将其一个只读节点提升为完全写入器状态，从而允许该辅助集群代入主集群的角色。由于目标辅助集群在过程开始时都与主集群同步，因此新的主集群将继续执行 Aurora Global Database 的操作，而不会丢失任何数据。您的数据库在短时间内不可用，而主集群和所选的辅助集群将担任其新角色。

> **Note**
>
> 要在执行切换后管理 Aurora PostgreSQL 的复制时隙，请参阅管理 Aurora PostgreSQL 的逻辑插槽。

为了优化应用程序可用性，我们建议您在使用此特征之前执行以下操作：

- 在非高峰时间段，或在向主数据库集群写入操作最少的其他时间执行此操作。
- 检查 Aurora 全局数据库中所有辅助 Aurora 数据库集群的滞后时间。对于所有基于 Aurora PostgreSQL 的全球数据库以及从引擎版本 3.04.0 及更高版本或 2.12.0 及更高版本开始的基于 Aurora MySQL 的全球数据库，请使用 Amazon CloudWatch 查看所有辅助数据库集群的 `AuroraGlobalDBRPOLag` 指标。对于基于 Aurora MySQL 的全球数据库的较低次要版本，请改为查看 `AuroraGlobalDBReplicationLag` 指标。这些指标显示复制到辅助集群滞后于复制到主数据库集群的时间（以毫秒为单位）。该值与 Aurora 完成切换所需的时间成正比。因此，滞后值越大，切换所需的时间就越长。检查这些指标时，请从当前的主集群进行检查。

  有关 Aurora 的 CloudWatch 指标的更多信息，请参阅 Amazon Aurora 的集群级指标。
- 在切换期间提升的辅助数据库集群的配置设置可能与旧的主数据库集群不同。建议您在 Aurora Global Database 集群中的所有集群之间使以下类型的配置设置保持一致。这样做有助于更大限度地减少切换后的性能问题、工作负载不兼容问题和其他异常行为。
  - 为新的主数据库集群配置 Aurora 数据库集群参数组（如有必要）– 当您提升辅助数据库集群以接管主数据库集群的角色时，辅助数据库集群中参数组的配置可能与主数据库集群的配置不同。如果是这样，请修改提升后的辅助数据库集群的参数组，使其与主集群的设置一致。要了解如何操作，请参阅修改 Aurora 全局数据库的参数。
  - 配置监控工具和选项（例如 Amazon CloudWatch Events 和警报）– 根据全局数据库的需要，为提升后的数据库集群配置相同的日志记录能力、警报等等。与参数组一样，在切换过程中，这些特征的配置不会从主数据库集群继承。一些 CloudWatch 指标（例如复制滞后）仅适用于辅助区域。因此，切换会更改查看这些指标和对指标设置警报的方式，并且可能要求更改任何预定义的控制面板。有关 Aurora 数据库集群的更多信息，请参阅使用 Amazon CloudWatch 监控 Amazon Aurora 指标。
  - 配置与其他 AWS 服务的集成 - 如果您的 Aurora 全球数据库与 AWS 服务（例如 AWS Secrets Manager、AWS Identity and Access Management、Amazon S3 和 AWS Lambda）集成，则确保根据需要配置与这些服务的集成。有关将 Aurora Global Database 与 IAM、Amazon S3 和 Lambda 集成的更多信息，请参阅 将 Amazon Aurora Global Database 与其他AWS服务结合使

用。要了解有关 Secrets Manager 的更多信息，请参阅如何跨 AWS 区域 自动复制 AWS Secrets Manager 中的密钥。

如果使用的是 Aurora Global Database 写入器端点，则无需更改应用程序中的连接设置。验证 DNS 更改已传播，并且您可以连接新的主集群并对其执行写入操作。然后，您可以恢复应用程序的完整运行。

假设您的应用程序连接使用旧主集群的集群端点，而不是全局写入器端点。在这种情况下，请务必更改应用程序连接设置，以使用新主集群的集群端点。如果您在创建 Aurora 全局数据库时接受了提供的名称，则可以在应用程序中从提升集群的终端节点字符串中删除 `-ro` 以更改终端节点。例如，辅助集群的终端节点 `my-global.cluster-ro-aaaaaabbbbbb.us-west-1.rds.amazonaws.com` 将在该集群提升为主集群时变为终端节点 `my-global.cluster-aaaaaabbbbbb.us-west-1.rds.amazonaws.com`。

如果您使用的是 RDS 代理，请确保将应用程序的写入操作重定向到与新主集群关联的代理的相应读/写端点。此代理端点可能是默认端点或自定义读/写端点。有关更多信息，请参阅 RDS 代理端点如何与全局数据库配合使用。

您可以使用 AWS Management Console、AWS CLI 或 RDS API 对 Aurora Global Database 进行切换。

控制台

对您的 Aurora 全球数据库执行切换

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 选择数据库，然后找到要进行切换的 Aurora 全局数据库。
3. 从操作菜单中选择切换或失效转移全球数据库。



4. 选择切换。



5. 对于新的主集群，选择其中一个辅助 AWS 区域中的活动集群作为新的主集群。
6. 选择确认。

切换完成后，您可以在数据库列表中看到 Aurora 数据库集群及其当前角色，如下图所示。



AWS CLI

对您的 Aurora 全球数据库执行切换

使用 `switchover-global-cluster` CLI 命令对 Aurora Global Database 进行切换。使用命令，传递下列参数的值。

- `--region` - 指定 Aurora Global Database 的主数据库集群运行的 AWS 区域。
- `--global-cluster-identifier` – 指定 Aurora 全局数据库的名称。
- `--target-db-cluster-identifier` – 指定要提升为 Aurora 全局数据库的主数据库集群的 Aurora 数据库集群的 Amazon 资源名称 (ARN)。

对于 Linux、macOS 或 Unix：

```
aws rds --region region_of_primary \
   switchover-global-cluster --global-cluster-identifier global_database_id \
  --target-db-cluster-identifier arn_of_secondary_to_promote
```

对于 Windows：

```
aws rds --region region_of_primary ^
   switchover-global-cluster --global-cluster-identifier global_database_id ^
```

```
  --target-db-cluster-identifier arn_of_secondary_to_promote
```

RDS API

要对 Aurora Global Database 进行切换，请运行 SwitchoverGlobalCluster API 操作。

从计划外停机中恢复 Amazon Aurora Global Database

在极少数情况下，您的 Aurora Global Database 可能会在其主 AWS 区域中发生意外停机。如果发生这种情况，主 Aurora 数据库集群及其写入器节点将不可用，并且主数据库集群和辅助数据库集群之间的复制将停止。为了最大限度地减少停机时间（RTO）和数据丢失（RPO），您可以快速执行跨区域失效转移。

Aurora Global Database 有两种失效转移方法可供您在灾难恢复情况下使用：

- 托管式失效转移 - 建议使用此方法进行灾难恢复。使用此方法时，当旧的主区域再次变为可用时，Aurora 会自动将其作为辅助区域重新添加到全球数据库中。因此，您的全球集群的原始拓扑保持不变。要了解如何使用此方法，请参阅执行 Aurora 全球数据库的托管式失效转移。
- 手动失效转移 – 当无法选择托管式失效转移时，例如，当您的主区域和辅助区域运行不兼容的引擎版本时，可以使用这种替代方法。要了解如何使用此方法，请参阅执行 Aurora 全球数据库的手动失效转移。

> **Important**
>
> 这两种失效转移方法都可能导致在失效转移事件发生之前，未复制到选定辅助系统的写入事务数据丢失。但是，将所选辅助数据库集群上的数据库实例提升为主写入器数据库实例的恢复过程可确保数据处于事务一致状态。失效转移也容易受到脑裂问题的影响。

执行 Aurora 全球数据库的托管式失效转移

这种方法用于在发生真实的区域性灾难或完全的服务级别中断时实现业务连续性。

在托管式失效转移期间，所选的辅助区域中的辅助集群将成为新的主集群。所选的辅助集群将其一个只读节点提升为完全写入器状态。此步骤允许集群代入主集群的角色。在此集群代入其新角色期间，您的数据库在短时间内不可用。一旦该旧主区域正常运行并再次可用，Aurora 就会自动将其作为辅助区域重新添加到全局集群中。因此，Aurora Global Database 的现有复制拓扑保持不变。

> **Note**
>
> 要在执行失效转移后管理 Aurora PostgreSQL 的复制时隙，请参阅管理 Aurora PostgreSQL 的逻辑插槽 。

> **Note**
>
> 仅当主数据库集群和辅助数据库集群具有相同的主要和次要引擎版本时，您才能对 Aurora Global Database 执行托管式跨区域失效转移。根据引擎和引擎版本，补丁级别可能需要相同，也可以不同。有关允许在具有不同补丁级别的主集群和辅助集群之间进行这些操作的引擎和引擎版本列表，请参阅托管式跨区域切换和失效转移的补丁级别兼容性。在开始失效转移之前，请检查全球集群中的引擎版本，以确保它们支持托管式跨区域切换，并在需要时对其进行升级。如果您的引擎版本需要相同的补丁级别，但正在运行不同的补丁级别，您可以按照执行 Aurora 全球数据库的手动失效转移中的步骤手动执行失效转移。

托管式失效转移不会等待所选辅助区域和当前主区域之间的数据同步。由于 Aurora Global Database 以异步方式复制数据，因此在所选的辅助 AWS 区域升级为接受完整读/写功能之前，可能并非所有事务都复制到该区域。

为确保数据处于一致状态，Aurora 会在旧的主区域恢复后为其创建一个新的存储卷。在 AWS 区域中创建新的存储卷之前，Aurora 会尝试在出现故障时拍摄旧存储卷的快照。这样，您就可以还原快照并从中恢复任何丢失的数据。如果此操作成功，Aurora 会将这个名为 `rds:unplanned-global-failover-`*`name-of-old-primary-DB-cluster`*`-`*`timestamp`* 的快照放在 AWS Management Console的“快照”部分中。您也可以使用 `describe-db-cluster-snapshots` AWS CLI 命令或 `DescribeDBClusterSnapshots` API 操作来查看该快照的详细信息。

当您启动托管式失效转移时，Aurora 还会尝试停止流经高可用 Aurora 存储层的写入流量。我们将这种机制称为“写入屏蔽”。如果该过程成功，Aurora 会触发一个 RDS 事件，告知您写入已停止。在极少数情况下，如果某个区域有多个可用区出现故障，则写入屏蔽过程可能无法及时成功。在这种情况下，Aurora 会触发一个 RDS 事件，通知您停止写入的过程已超时。如果可在网络上访问旧的主集群，Aurora 会在那里记录这些事件。否则，Aurora 会将事件记录在新的主集群上。要了解有关这些事件的更多信息，请参阅数据库集群事件。由于屏蔽写入是一种尽力而为的尝试，因此旧的主区域可能会暂时接受写入，从而导致脑裂问题。

建议您在对 Aurora Global Database 执行失效转移之前，先完成以下任务。这样做可以更大限度地降低脑裂问题的可能性，或从旧主集群的快照中恢复未复制数据的可能性。

- 要防止写入内容被发送到 Aurora Global Database 的主集群，请使应用程序离线。
- 确保连接到主数据库集群的所有应用程序都使用全局写入器端点。即使新区域由于切换或失效转移而成为主集群，此端点的值也保持不变。Aurora 实施额外的保护措施，以更大限度地降低通过全局端点提交的写入操作数据丢失的可能性。有关全局写入器端点的更多信息，请参阅 连接到 Amazon Aurora Global Database。
- 如果您使用的是全局写入器端点，并且您的应用程序层或网络层缓存 DNS 值，请将 DNS 缓存的生存时间（TTL）缩短至较低值，例如 5 秒。这样，您的应用程序就可以快速向全局写入器端点注册 DNS 更改。尽管 Aurora 试图阻止在旧主区域中写入，但不能保证该操作会成功。缩短 DNS 缓存持续时间可进一步降低出现脑裂问题的可能性。或者，您也可以查看 RDS 事件，该事件会在 Aurora 观察到全局写入器端点的 DNS 更改时通知您。这样，您就可以在重新启动应用程序写入流量之前验证您的应用程序也注册了 DNS 更改。
- 检查 Aurora Global Database 中所有辅助 Aurora 数据库集群的滞后时间。选择复制滞后最小的辅助区域可以最大限度地减少当前出现故障的主区域的数据丢失。

  对于所有基于 Aurora PostgreSQL 的全局数据库版本，以及从引擎版本 3.04.0 及更高版本或 2.12.0 及更高版本开始的基于 Aurora MySQL 的全局数据库，请使用 Amazon CloudWatch 查看所有辅助数据库集群的 `AuroraGlobalDBRPOLag` 指标。对于基于 Aurora MySQL 的全球数据库的较低次要版本，请改为查看 `AuroraGlobalDBReplicationLag` 指标。这些指标显示复制到辅助集群滞后于复制到主数据库集群的时间（以毫秒为单位）。

  有关 Aurora 的 CloudWatch 指标的更多信息，请参阅 Amazon Aurora 的集群级指标。

在托管式失效转移期间，所选的辅助数据库集群将提升为新角色，即主数据库集群。但是，它不会继承主数据库集群的各种配置选项。配置不匹配可能会导致性能问题、工作负载不兼容和其他异常行为。为避免出现此类问题，我们建议您解决以下方面 Aurora 全局数据库集群之间的差异问题：

- 为新的主数据库集群配置 Aurora 数据库集群参数组（如有必要）– 您可以为 Aurora Global Database 中的每个 Aurora 集群单独配置 Aurora 数据库集群参数组。因此，当您提升辅助数据库集群以接管主数据库集群的角色时，辅助数据库集群中参数组的配置可能与主数据库集群的配置不同。如果是这样，请修改提升后的辅助数据库集群的参数组，使其与主集群的设置一致。要了解如何操作，请参阅修改 Aurora 全局数据库的参数。
- 配置监控工具和选项（例如 Amazon CloudWatch Events 和警报）– 根据全局数据库的需要，为提升后的数据库集群配置相同的日志记录能力、警报等等。与参数组一样，在故障转移过程中，这些特征的配置不会从主数据库集群继承。一些 CloudWatch 指标（例如复制滞后）仅适用于辅助区域。因此，失效转移会更改查看这些指标和对指标设置警报的方式，并且可能要求更改任何预定义的控

制面板。有关监控 Aurora 数据库集群的更多信息，请参阅使用 Amazon CloudWatch 监控 Amazon Aurora 指标。

- 配置与其他 AWS 服务的集成 - 如果您的 Aurora Global Database 与 AWS 服务（例如 AWS Secrets Manager、AWS Identity and Access Management、Amazon S3 和 AWS Lambda）集成，则需要确保根据需要对这些服务进行配置，以从任何辅助区域进行访问。有关将 Aurora Global Database 与 IAM、Amazon S3 和 Lambda 集成的更多信息，请参阅 将 Amazon Aurora Global Database 与其他AWS服务结合使用。要了解有关 Secrets Manager 的更多信息，请参阅如何跨 AWS 区域 自动复制 AWS Secrets Manager 中的密钥。

通常，所选的辅助集群会在几分钟内代入主角色。一旦新的主区域的写入器数据库实例可用，您就可以将应用程序连接到该实例并恢复工作负载。Aurora 提升新的主集群后，它会自动重建所有其他辅助区域集群。

由于 Aurora 全球数据库使用异步复制，因此每个辅助区域的复制滞后可能会有所不同。Aurora 重建这些辅助区域，使其具有与新的主区域集群完全相同的时间点数据。完成重建任务的持续时间可能需要几分钟到几小时，具体取决于存储卷的大小和区域之间的距离。当辅助区域集群从新的主区域完成重建后，它们就可供进行读取访问了。

一旦新的主写入器已提升并可用，新的主区域的集群就可以处理 Aurora 全球数据库的读取和写入操作。

如果使用的是全局端点，则无需更改应用程序中的连接设置。验证 DNS 更改已传播，并且您可以连接新的主集群并对其执行写入操作。然后，您可以恢复应用程序的完整运行。

如果不使用全局端点，请务必更改应用程序的端点，以使用新提升的主数据库集群的集群端点。如果您在创建 Aurora 全局数据库时接受了提供的名称，则可以在应用程序中从提升集群的终端节点字符串中删除 `-ro` 以更改终端节点。

例如，辅助集群的终端节点 `my-global.cluster-ro-aaaaaabbbbbb.us-west-1.rds.amazonaws.com` 将在该集群提升为主集群时变为终端节点 `my-global.cluster-aaaaaabbbbbb.us-west-1.rds.amazonaws.com`。

如果您使用的是 RDS 代理，请确保将应用程序的写入操作重定向到与新的主集群关联的代理的相应读取/写入端点。此代理端点可能是默认端点或自定义读/写端点。有关更多信息，请参阅 RDS 代理端点如何与全局数据库配合使用。

为了还原全球数据库集群的原始拓扑，Aurora 会监控旧主区域的可用性。一旦该区域正常运行并再次可用，Aurora 就会自动将其作为辅助区域重新添加到全球集群中。在旧的主区域中创建新的存储卷之前，Aurora 会尝试在出现故障时拍摄旧存储卷的快照。它这样做是为了让您可以用它来恢复任何丢失

的数据。如果此操作成功，Aurora 将创建一个名为 `rds:unplanned-global-failover-name-of-old-primary-DB-cluster-timestamp` 的快照。您可以在 AWS Management Console的快照部分中找到该快照。还可以在 DescribeDBClusterSnapshots API 操作返回的信息中看到列出的此快照。

> **Note**
>
> 旧存储卷的快照是系统快照，受旧的主集群上配置的备份保留期限制。要在保留期之外保留此快照，可以复制它以另存为手动快照。要了解有关复制快照的更多信息（包括定价），请参阅数据库集群快照复制。

还原原始拓扑后，您可以通过在对业务和工作负载最有意义的时候执行切换操作，将全球数据库失效自动恢复到原始主区域。为此，请按照对 Amazon Aurora Global Database 执行切换中的步骤进行操作。

您可以使用 AWS Management Console、AWS CLI 或 RDS API 对 Aurora Global Database 进行失效转移。

控制台

在 Aurora 全球数据库上执行托管式失效转移

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 选择数据库，然后找到要进行失效转移的 Aurora 全局数据库。
3. 从操作菜单中选择切换或失效转移全球数据库。



4. 选择失效转移（允许数据丢失）。



5. 对于新的主集群，选择其中一个辅助 AWS 区域中的活动集群作为新的主集群。
6. 输入 **`confirm`**，然后选择确认。

失效转移完成后，您可以在数据库列表中查看 Aurora 数据库集群及其当前状态，如下图所示。



AWS CLI

在 Aurora 全球数据库上执行托管式失效转移

使用 `failover-global-cluster` CLI 命令对 Aurora Global Database 进行失效转移。使用命令，传递下列参数的值。

- `--region` – 指定一个 AWS 区域，这是您希望成为 Aurora 全球数据库的新主数据库集群的辅助数据库集群正在运行的位置。
- `--global-cluster-identifier` – 指定 Aurora 全局数据库的名称。
- `--target-db-cluster-identifier` – 指定要提升为 Aurora 全球数据库的新主数据库集群的 Aurora 数据库集群的 Amazon 资源名称（ARN）。
- `--allow-data-loss` – 显式使其成为失效转移操作而不是切换操作。如果异步复制组件尚未完成将所有复制的数据发送到辅助区域，则失效转移操作可能会导致一些数据丢失。

对于 Linux、macOS 或 Unix：

```
aws rds --region region_of_selected_secondary \
   failover-global-cluster --global-cluster-identifier global_database_id \
   --target-db-cluster-identifier arn_of_secondary_to_promote \
   --allow-data-loss
```

对于 Windows：

```
aws rds --region region_of_selected_secondary ^
   failover-global-cluster --global-cluster-identifier global_database_id ^
   --target-db-cluster-identifier arn_of_secondary_to_promote ^
   --allow-data-loss
```

RDS API

要对 Aurora Global Database 进行失效转移，请运行 FailoverGlobalCluster API 操作。

执行 Aurora 全球数据库的手动失效转移

在某些情况下，您可能无法使用托管式失效转移过程。例如，如果您的主数据库集群和辅助数据库集群运行的引擎版本不兼容。在这种情况下，您可以按照此手动过程，失效转移到目标辅助区域。

> **Tip**
>
> 我们建议您在实施此过程之前先了解此过程。准备好计划，以便在区域级问题初见端倪时快速处理。通过定期使用 Amazon CloudWatch 来跟踪辅助集群的滞后时间，您就可以确定复制滞后最少的辅助区域。确保测试计划，以检查您的过程是否完整和准确，并确保员工在灾难恢复失效转移真正发生之前接受了相关培训以执行此失效转移过程。

在主区域发生计划外停机后，手动失效转移到辅助集群

1. 停止向停机的 AWS 区域 中的主 Aurora 数据库集群发出 DML 语句和其他写入操作。
2. 从辅助 AWS 区域 中标识作为新的主数据库集群的 Aurora 数据库集群。如果您的 Aurora 全球数据库中有两个或更多辅助 AWS 区域，请选择复制滞后最少的辅助集群。
3. 从 Aurora 全局数据库分离您所选的辅助数据库集群。

   从 Aurora Global Database 删除备用数据库集群会立即停止从主数据库集群到该备用数据库集群的复制过程，并会将其提升为拥有完全读/写功能的独立预置的 Aurora 数据库集群。与该停机区域中的主集群关联的任何其他辅助 Aurora 数据库集群仍然可用，并且可以接受应用程序的调用。它们还会消耗资源。由于您要重新创建 Aurora 全局数据库，请先删除其他备用数据库集群，再在后续步骤中创建新的 Aurora 全局数据库。这样做可以避免 Aurora Global Database 中数据库集群之间的数据不一致（脑裂问题）。

   有关分离的详细步骤，请参阅 从 Amazon Aurora Global Database 删除集群。

4. 重新配置应用程序，使用新的终端节点将所有写入操作发送到现在的独立 Aurora 数据库集群。如果您在创建 Aurora 全局数据库时接受了提供的名称，则可以在应用程序中从集群的终端节点字符串中删除 `-ro` 以更改终端节点。

   例如，辅助集群的终端节点 `my-global.cluster-ro-aaaaaabbbbbb.us-west-1.rds.amazonaws.com` 将在该集群从 Aurora 全局数据库分离时变为终端节点 `my-global.cluster-aaaaaabbbbbb.us-west-1.rds.amazonaws.com`。

   在下一步中，当您开始向 Aurora 数据库集群添加区域时，该数据库集群将成为新的 Aurora Global Database 的主集群。

   如果您使用的是 RDS 代理，请确保将应用程序的写入操作重定向到与新的主集群关联的代理的相应读取/写入端点。此代理端点可能是默认端点或自定义读/写端点。有关更多信息，请参阅 RDS 代理端点如何与全局数据库配合使用。
5. 向数据库集群添加 AWS 区域。执行此操作后，从主数据库集群到辅助数据库集群的复制过程将会开始。有关添加区域的详细步骤，请参阅 将 AWS 区域 添加到 Amazon Aurora Global Database。
6. 根据需要添加更多 AWS 区域，以重新创建为了支持应用程序所需的拓扑。

确保在进行这些更改之前、期间和之后，将应用程序写入发送到正确的 Aurora 数据库集群。这样做可以避免 Aurora Global Database 中数据库集群之间的数据不一致（脑裂问题）。

如果您为响应 AWS 区域中的中断进行了重新配置，则可以在解决中断后再次将 AWS 区域设置为主区域。为此，您需要将旧的 AWS 区域添加到新的全球数据库中，然后使用切换过程以切换其角色。您的 Aurora 全球数据库必须使用支持切换的 Aurora PostgreSQL 或 Aurora MySQL 版本。有关更多信息，请参阅 对 Amazon Aurora Global Database 执行切换。

管理基于 Aurora PostgreSQL 的全局数据库的 RPO

通过基于 Aurora PostgreSQL 的全球数据库，您可以使用 `rds.global_db_rpo` 参数管理 Aurora 全球数据库的恢复点目标（RPO）。RPO 表示在停机时可能丢失的最大数据量。

为基于 Aurora PostgreSQL 的全局数据库设置 RPO 后，Aurora 会监控所有辅助集群的 RPO 滞后时间，以确保至少有一个辅助集群保留在目标 RPO 窗口内。RPO 滞后时间是另一个基于时间的指标。

当您的数据库在发生故障转移后在新 AWS 区域 中恢复运行时，将使用 RPO。Aurora 会按如下方式评估 RPO 和 RPO 滞后时间，以便在主数据库集群上提交（或阻止）交易：

- 如果至少有一个辅助数据库集群的 RPO 滞后时间小于 RPO，则提交事务。

- 如果所有辅助数据库集群的 RPO 滞后时间大于 RPO，则阻止事务。它还会将事件记录到 PostgreSQL 日志文件中，并发出显示被阻止的会话的“等待”事件。

换言之，如果所有辅助集群都落后于目标 RPO，则 Aurora 会在主集群中暂停，直到至少一个辅助集群与目标 RPO 同步。一旦至少一个备用数据库集群的滞后时间小于 RPO，就会恢复并提交暂停的事务。结果是在满足 RPO 条件之后，才能提交事务。

`rds.global_db_rpo` 参数是动态的。如果您决定在滞后充分减少之前不希望所有写入事务都停滞不前，则可以快速将其重置。在这种情况下，Aurora 会在短暂延迟后识别并实施更改。

> **Important**
>
> 在只有两个 AWS 区域的全局数据库中，我们建议在辅助区域的参数组中保留 `rds.global_db_rpo` 参数的默认值。否则，由于主 AWS 区域丢失而执行失效转移可能会导致 Aurora 暂停事务。相反，请等待 Aurora 在旧的故障 AWS 区域中完成集群重建，然后更改此参数以实施最大 RPO。

如果按下述步骤设置此参数，则还可以监控其生成的指标。您可以通过使用 `psql` 或其他工具查询 Aurora 全局数据库的主数据库集群并获取基于 Aurora PostgreSQL 的全局数据库操作的详细信息来执行此操作。要了解如何操作，请参阅监控基于 Aurora PostgreSQL 的全局数据库。

主题

- 设置恢复点目标
- 查看恢复点目标
- 禁用恢复点目标

设置恢复点目标

该 `rds.global_db_rpo` 参数控制 PostgreSQL 数据库的 RPO 设置。Aurora PostgreSQL 支持此参数。`rds.global_db_rpo` 的有效值的范围是从 20 秒到 2147483647 秒（68 年）。请选择符合您的业务需求和使用案例的真实值。例如，您可能希望最多为 RPO 留出 10 分钟的时间，在这种情况下，可以将值设置为 600。

您可以使用AWS Management Console、AWS CLI 或 RDS API 为基于 Aurora PostgreSQL 的全局数据库设置此值。

控制台

设置 RPO

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 选择 Aurora 全局数据库的主集群，然后打开 Configuration（配置）选项卡以查找其数据库集群参数组。例如，运行 Aurora PostgreSQL 11.7 的主数据库集群的默认参数组为 `default.aurora-postgresql11`。

   参数组无法直接编辑。您可以改而执行以下操作：

   - 使用适当的默认参数组作为起点创建自定义数据库集群参数组。例如，基于 `default.aurora-postgresql11` 创建自定义数据库集群参数组。
   - 在自定义数据库参数组中，设置 rds.global_db_rpo 参数的值以符合您的使用案例。有效值的范围是从 20 秒到最大整数值 2147483647（68 年）。
   - 请将修改后的数据库集群参数组应用于 Aurora 数据库集群。

有关更多信息，请参阅“在 Amazon Aurora 中修改数据库集群参数组中的参数”。

AWS CLI

要设置 `rds.global_db_rpo` 参数，请使用 modify-db-cluster-parameter-group CLI 命令。在此命令中，请指定主集群参数组的名称和 RPO 参数的值。

以下示例将名为 `my_custom_global_parameter_group` 的主数据库集群参数组的 RPO 设置为 600 秒（10 分钟）。

对于 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster-parameter-group \
    --db-cluster-parameter-group-name my_custom_global_parameter_group \
    --parameters
 "ParameterName=rds.global_db_rpo,ParameterValue=600,ApplyMethod=immediate"
```

对于 Windows：

```
aws rds modify-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name my_custom_global_parameter_group ^
```

```
    --parameters
 "ParameterName=rds.global_db_rpo,ParameterValue=600,ApplyMethod=immediate"
```

RDS API

要修改 rds.global_db_rpo 参数，请使用 Amazon RDS ModifyDBClusterParameterGroup API 操作。

查看恢复点目标

全局数据库的恢复点目标 (RPO) 存储每个数据库集群的 rds.global_db_rpo 参数中。您可以连接到要查看的辅助集群的终端节点，并使用 psql 查询该值的实例。

```
db-name=>show rds.global_db_rpo;
```

如果未设置此参数，查询将返回以下内容：

```
rds.global_db_rpo
-------------------
 -1
(1 row)
```

下一个响应来自 RPO 设置为 1 分钟的辅助数据库集群。

```
rds.global_db_rpo
-------------------
 60
(1 row)
```

您还可以使用 CLI 获取集群的所有 rds.global_db_rpo 参数的值，以便了解 user 是否在任何 Aurora 数据库集群上处于活动状态。

对于 Linux、macOS 或 Unix：

```
aws rds describe-db-cluster-parameters \
 --db-cluster-parameter-group-name lab-test-apg-global \
 --source user
```

对于 Windows：

```
aws rds describe-db-cluster-parameters ^
```

```
  --db-cluster-parameter-group-name lab-test-apg-global *
  --source user
```

该命令会对所有 user 参数（非 default-engine 或 system 数据库集群参数）返回与以下内容类似的输出。

```
{
    "Parameters": [
        {
            "ParameterName": "rds.global_db_rpo",
            "ParameterValue": "60",
            "Description": "(s) Recovery point objective threshold, in seconds, that
 blocks user commits when it is violated.",
            "Source": "user",
            "ApplyType": "dynamic",
            "DataType": "integer",
            "AllowedValues": "20-2147483647",
            "IsModifiable": true,
            "ApplyMethod": "immediate",
            "SupportedEngineModes": [
                "provisioned"
            ]
        }
    ]
}
```

要了解有关查看集群参数组的参数的更多信息，请参阅 在 Amazon Aurora 中查看数据库集群参数组的参数值。

禁用恢复点目标

要禁用 RPO，请重置 rds.global_db_rpo 参数。您可以使用 AWS Management Console、AWS CLI 或 RDS API 重置参数。

控制台

禁用 RPO

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择参数组。

3. 在列表中，选择您的主数据库集群参数组。
4. 选择编辑参数。
5. 选择 rds.global_db_rpo 参数旁边的框。
6. 选择 Reset (重置)。
7. 当屏幕显示 Reset parameters in DB parameter group (重置数据库参数组中的参数) 时，选择 Reset parameters (重置参数)。

有关如何使用控制台重置参数的更多信息，请参阅在 Amazon Aurora 中修改数据库集群参数组中的参数。

AWS CLI

要重置 `rds.global_db_rpo` 参数，请使用 reset-db-cluster-parameter-group 命令。

对于 Linux、macOS 或 Unix：

```
aws rds reset-db-cluster-parameter-group \
    --db-cluster-parameter-group-name global_db_cluster_parameter_group \
    --parameters "ParameterName=rds.global_db_rpo,ApplyMethod=immediate"
```

对于 Windows：

```
aws rds reset-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name global_db_cluster_parameter_group ^
    --parameters "ParameterName=rds.global_db_rpo,ApplyMethod=immediate"
```

RDS API

要重置 `rds.global_db_rpo` 参数，请使用 Amazon RDS API ResetDBClusterParameterGroup 操作。

全局数据库辅助集群的跨区域弹性

Aurora PostgreSQL 16.6、15.10、14.15、13.18、12.22 或更高版本，以及 Aurora MySQL 版本 3.09 或更高版本包含可用性改进，使辅助区域只读副本能够在发生计划外事件（例如硬件故障、跨 AWS 区域网络中断、集群之间大量数据传输等）期间保持服务连续性。

尽管只读副本仍然可供您的应用程序请求使用，但在计划外事件得到解决之前，复制延迟可能会继续增加。可以使用 `AuroraGlobalDBProgressLag` CloudWatch 指标监控主集群和辅助集

群之间的延迟。要衡量端到端延迟，包括集群卷和辅助集群的数据库实例之间的任何延迟，请将 `AuroraGlobalDBProgressLag` 和 `AuroraReplicaLag` CloudWatch 指标的值相加。有关指标的更多信息，请参阅 Amazon Aurora 的指标参考。

在此类计划外事件中，Aurora MySQL 和早期版本的 Aurora PostgreSQL 的全局数据库读取可用性可能会受到影响。

有关 Aurora PostgreSQL 16.6、15.10、14.15、13.18 和 12.22 中新功能的更多信息，请参阅《Aurora PostgreSQL Release Notes》中的 PostgreSQL 16.6。

有关 Aurora MySQL 版本 3.09 及更高版本中新功能的更多信息，请参阅《Aurora MySQL Release Notes》中的 Database engine updates for Amazon Aurora MySQL version 3。

监控 Amazon Aurora Global Database

当您创建构成 Aurora 全局数据库的 Aurora 数据库集群时，您可以选择许多选项来监控数据库集群的性能。这些选项包含以下内容：

- Amazon RDS 性能详情 – 在底层 Aurora 数据库引擎中启用性能架构。要了解有关性能详情和 Aurora 全局数据库的更多信息，请参阅 使用 Amazon RDS 性能详情监控 Amazon Aurora Global Database。
- 增强监控 – 生成 CPU 上的进程或线程利用率的指标。要了解有关增强型监控的信息，请参阅使用增强监控来监控操作系统指标。
- Amazon CloudWatch Logs – 将指定日志类型发布到 CloudWatch Logs。默认情况下会发布错误日志，但您可以选择特定于 Aurora 数据库引擎的其他日志。
  - 对于基于 Aurora MySQL– 的 Aurora 数据库集群，您可以导出审核日志、常规日志和慢查询日志。
  - 对于基于 Aurora PostgreSQL 的 Aurora 数据库集群，您可以导出 PostgreSQL 日志。
- 对于基于 Aurora MySQL 的全局数据库，您可以查询特定的 `information_schema` 表来检查 Aurora 全局数据库及其实例的状态。要了解如何操作，请参阅监控基于 Aurora MySQL 的全局数据库。
- 对于基于 Aurora PostgreSQL 的全局数据库，您可以使用特定的函数来检查 Aurora 全局数据库及其实例的状态。要了解如何操作，请参阅监控基于 Aurora PostgreSQL 的全局数据库。

以下屏幕截图显示了 Aurora 全局数据库中的主 Aurora 数据库集群的 Monitoring (监控) 选项卡上的一些可用选项。



有关更多信息，请参阅 监控 Amazon Aurora 集群中的指标。

使用 Amazon RDS 性能详情监控 Amazon Aurora Global Database

您可以将 Amazon RDS 性能详情用于 Aurora 全局数据库。您可以为 Aurora 全局数据库中的每个数据 Aurora 库集群单独启用此特征。为此，您可以在 Create database (创建数据库) 页面的 Additional configuration (其他配置) 部分中选择 Enable Performance Insights (启用性能详情)。或者，您可以修改 Aurora 数据库集群，以便在其正常运行后使用此特征。您可以针对 Aurora 全局数据库中的每个集群启用或禁用性能详情功能。

性能详情创建的报告适用于全局数据库中的每个集群。在将新的辅助 AWS 区域 添加到已在使用 Performance Insights 的 Aurora Global Database 时，确保在新添加的集群中启用 Performance Insights。此集群不能从现有全局数据库继承 Performance Insights 设置。

在查看附加到全局数据库的数据库实例的 Performance Insights 页面时，您可以切换 AWS 区域。但是，您可能不能在切换 AWS 区域 后立即看到性能信息。在各个 AWS 区域 中，虽然数据库实例的名

称可能会相同，但每个数据库实例的相关 Performance Insights URL 不同。在切换 AWS 区域 后，可在 Performance Insights 导航窗格中重新选择数据库实例的名称。

对于与全局数据库关联的数据库实例，各 AWS 区域 中影响性能的系数可能不同。例如，各 AWS 区域中数据库实例的容量可能不同。

要了解有关使用性能详情的更多信息，请参阅 在 Amazon Aurora 上使用性能详情监控数据库负载。

使用数据库活动流监控 Aurora Global Database

利用数据库活动流特征，您可以监控全局数据库中数据库集群的审计活动并为其设置告警。请分别在每个数据库集群上启动数据库活动流。每个集群在其自己的 AWS 区域中将审计数据传送到其自己的 Kinesis 流。有关更多信息，请参阅 使用数据库活动流监控 Amazon Aurora。

监控基于 Aurora MySQL 的全局数据库

要查看基于 Aurora MySQL 的全局数据库的状态，请查询 information_schema.aurora_global_db_status 和 information_schema.aurora_global_db_instance_status 表。

> **Note**
>
> `information_schema.aurora_global_db_status` 和 `information_schema.aurora_global_db_instance_status` 表仅适用于 Aurora MySQL 版本 3.04.0 及更高版本的全局数据库。

监控基于 Aurora MySQL 的全局数据库

1. 使用 MySQL 客户端连接到全局数据库主集群端点。有关如何连接的更多信息，请参阅 连接到 Amazon Aurora Global Database。
2. 在 mysql 命令中查询 `information_schema.aurora_global_db_status` 表以列出主卷和辅助卷。此查询返回全局数据库辅助数据库集群的滞后时间，如以下示例所示。

```
mysql> select * from information_schema.aurora_global_db_status;
```

```
AWS_REGION | HIGHEST_LSN_WRITTEN | DURABILITY_LAG_IN_MILLISECONDS |
 RPO_LAG_IN_MILLISECONDS | LAST_LAG_CALCULATION_TIMESTAMP | OLDEST_READ_VIEW_TRX_ID
```

```
-----------+---------------------+--------------------------------
+------------------------+---------------------------------
+------------------------
us-east-1  |           183537946 |                              0   |
      0 |  1970-01-01 00:00:00.000000     |               0
us-west-2  |           183537944 |                            428 |
      0 |  2023-02-18 01:26:41.925000     |               20806982
(2 rows)
```

输出包含全局数据库的每个数据库集群行，其中包含以下列：

- AWS_REGION – 此数据库集群所在的 AWS 区域。有关按引擎列出 AWS 区域的表，请参阅 区域可用性。
- HIGHEST_LSN_WRITTEN – 当前在此数据库集群上写入的最高日志序列号（LSN）。

  日志序列号 (LSN) 是标识数据库事务日志中的记录的唯一序列号。对 LSN 进行排序，以便较大的 LSN 表示较晚的事务。
- DURABILITY_LAG_IN_MILLISECONDS – 辅助数据库集群上的 HIGHEST_LSN_WRITTEN 与主数据库集群上的 HIGHEST_LSN_WRITTEN 之间的时间戳值差异。在 Aurora 全局数据库的主数据库集群上，该值始终为 0。
- RPO_LAG_IN_MILLISECONDS – 恢复点目标（RPO）滞后。RPO 滞后是最近的用户事务在存储在 Aurora 全局数据库的主数据库集群上之后，执行 COMMIT 操作以便存储在辅助数据库集群上所需的时间。在 Aurora 全局数据库的主数据库集群上，该值始终为 0。

  简而言之，该指标计算 Aurora 全局数据库中每个 Aurora MySQL 数据库集群的恢复点目标，也就是说，如果发生中断，可能会丢失多少数据。与滞后一样，RPO 是按时间计量的。
- LAST_LAG_CALCULATION_TIMESTAMP – 最后为 DURABILITY_LAG_IN_MILLISECONDS 和 RPO_LAG_IN_MILLISECONDS 计算值时指定的时间戳。时间值（如 1970-01-01 00:00:00+00）表示这是主数据库集群。
- OLDEST_READ_VIEW_TRX_ID – 写入器数据库实例可以清除到的最早事务的 ID。

3. 查询 information_schema.aurora_global_db_instance_status 表，以列出主数据库集群和辅助数据库集群的所有辅助数据库实例。

```
mysql> select * from information_schema.aurora_global_db_instance_status;
```

```
SERVER_ID            |              SESSION_ID              | AWS_REGION
 | DURABLE_LSN | HIGHEST_LSN_RECEIVED | OLDEST_READ_VIEW_TRX_ID |
 OLDEST_READ_VIEW_LSN | VISIBILITY_LAG_IN_MSEC
```

```
--------------------+--------------------------------------+-----------
+------------+---------------------+------------------------
+----------------------+------------------------
ams-gdb-primary-i2   | MASTER_SESSION_ID                    | us-east-1  |
 183537698   |                   0 |                      0 |
 0 |                      0
ams-gdb-secondary-i1 | cc43165b-bdc6-4651-abbf-4f74f08bf931 | us-west-2  |
 183537689   |           183537692 |                20806928 |
 183537682 |                     0
ams-gdb-secondary-i2 | 53303ff0-70b5-411f-bc86-28d7a53f8c19 | us-west-2  |
 183537689   |           183537692 |                20806928 |
 183537682 |                   677
ams-gdb-primary-i1   | 5af1e20f-43db-421f-9f0d-2b92774c7d02 | us-east-1  |
 183537697   |           183537698 |                20806930 |
 183537691 |                    21
(4 rows)
```

输出包含全局数据库的每个数据库实例行，其中包含以下列：

- SERVER_ID – 数据库实例的服务器标识符。
- SESSION_ID – 当前会话的唯一标识符。MASTER_SESSION_ID 的值标识写入器（主要）数据库实例。
- AWS_REGION – 此数据库实例所在的 AWS 区域。有关按引擎列出 AWS 区域的表，请参阅 区域可用性。
- DURABLE_LSN – 存储中持久的 LSN。
- HIGHEST_LSN_RECEIVED – 数据库实例从写入器数据库实例收到的最高 LSN。
- OLDEST_READ_VIEW_TRX_ID – 写入器数据库实例可以清除到的最早事务的 ID。
- OLDEST_READ_VIEW_LSN – 数据库实例从存储中读取所用的最早 LSN。
- VISIBILITY_LAG_IN_MSEC – 对于主数据库集群中的读取器，此数据库实例滞后于写入器数据库实例的时间（以毫秒为单位）。对于辅助数据库集群中的读取器，此数据库实例滞后于辅助卷的时间（以毫秒为单位）。

要查看这些值如何随时间变化，请考虑以下事务块，其中表插入需要一个小时。

```
mysql> BEGIN;
mysql> INSERT INTO table1 SELECT Large_Data_That_Takes_1_Hr_To_Insert;
mysql> COMMIT;
```

在某些情况下，在 BEGIN 语句之后主数据库集群和辅助数据库集群之间的网络会断开连接。如果是这样，则辅助数据库集群的 DURABILITY_LAG_IN_MILLISECONDS 值开始增加。在 INSERT 语句结束时，DURABILITY_LAG_IN_MILLISECONDS 值为 1 小时。但是，RPO_LAG_IN_MILLISECONDS 值为 0，因为在主数据库集群和辅助数据库集群之间提交的所有用户数据仍然相同。一旦 COMMIT 语句完成，RPO_LAG_IN_MILLISECONDS 值就会增加。

监控基于 Aurora PostgreSQL 的全局数据库

要查看基于 Aurora PostgreSQL 的全局数据库的状态，请使用 aurora_global_db_status 和 aurora_global_db_instance_status 函数。

> **Note**
>
> 仅 Aurora PostgreSQL 支持 aurora_global_db_status 和 aurora_global_db_instance_status 函数。

监控基于 Aurora PostgreSQL 的全局数据库

1. 使用 PostgreSQL 实用工具（如 psql）连接到全局数据库主集群终端节点。有关如何连接的更多信息，请参阅 连接到 Amazon Aurora Global Database。
2. 在 psql 命令中使用 aurora_global_db_status 函数列出主卷和辅助卷。这显示全局数据库辅助数据库集群的滞后时间。

```
postgres=> select * from aurora_global_db_status();
```

```
aws_region | highest_lsn_written | durability_lag_in_msec | rpo_lag_in_msec |
 last_lag_calculation_time  | feedback_epoch | feedback_xmin
------------+---------------------+------------------------+-----------------
+----------------------------+----------------+---------------
us-east-1  |         93763984222 |                     -1 |              -1 |
 1970-01-01 00:00:00+00     |              0 |              0
us-west-2  |         93763984222 |                    900 |            1090 |
 2020-05-12 22:49:14.328+00 |              2 |    3315479243
(2 rows)
```

输出包含全局数据库的每个数据库集群行，其中包含以下列：

- aws_region - 此数据库集群所在的 AWS 区域。有关按引擎列出 AWS 区域的表，请参阅 区域可用性。
- highest_lsn_written – 当前在此数据库集群上写入的最高日志序列号 (LSN)。

  日志序列号 (LSN) 是标识数据库事务日志中的记录的唯一序列号。对 LSN 进行排序，以便较大的 LSN 表示较晚的事务。
- durability_lag_in_msec – 辅助数据库集群上写入的最高日志序列号 (`highest_lsn_written`) 与主数据库集群上的 `highest_lsn_written` 之间的时间戳差异。
- rpo_lag_in_msec – 恢复点目标 (RPO) 滞后。此滞后是存储在辅助数据库集群上的最新用户事务提交与存储在主数据库集群上的最新用户事务提交之间的时差。
- last_lag_calculation_time – 最后为 `durability_lag_in_msec` 和 `rpo_lag_in_msec` 计算值的时间戳。
- feedback_epoch – 辅助数据库集群在生成热备用服务器信息时使用的纪元。

  热备用 是指在服务器处于恢复或备用模式时，数据库集群可以进行连接和查询。热备用反馈是有关处于热备用状态的数据库集群的信息。有关更多信息，请参阅 PostgreSQL 文档中的热备用。
- feedback_xmin – 辅助数据库集群使用的最小（最早）活动事务 ID。

3. 使用 `aurora_global_db_instance_status` 函数列出主数据库集群和辅助数据库集群的所有辅助数据库实例。

```
postgres=> select * from aurora_global_db_instance_status();
```

```
server_id                                    |              session_id
  | aws_region | durable_lsn | highest_lsn_rcvd | feedback_epoch | feedback_xmin |
 oldest_read_view_lsn | visibility_lag_in_msec
----------------------------------------------+--------------------------------------
+------------+-------------+------------------+----------------+--------------
+----------------------+-----------------------
apg-global-db-rpo-mammothrw-elephantro-1-n1 | MASTER_SESSION_ID
 | us-east-1  | 93763985102 |                  |                |              |
                      |
apg-global-db-rpo-mammothrw-elephantro-1-n2 | f38430cf-6576-479a-b296-dc06b1b1964a
 | us-east-1  | 93763985099 |      93763985102 |              2 |    3315479243 |
        93763985095 |                      10
```

```
apg-global-db-rpo-elephantro-mammothrw-n1   | 0d9f1d98-04ad-4aa4-8fdd-e08674cbbbfe
 | us-west-2  | 93763985095 |      93763985099 |              2 |    3315479243 |
        93763985089 |                   1017
(3 rows)
```

输出包含全局数据库的每个数据库实例行，其中包含以下列：

- server_id – 数据库实例的服务器标识符。
- session_id – 当前会话的唯一标识符。
- aws_region - 此数据库实例所在的 AWS 区域。有关按引擎列出 AWS 区域的表，请参阅 区域可用性。
- durable_lsn – 存储中持久的 LSN。
- highest_lsn_rcvd – 数据库实例从写入器数据库实例收到的最高 LSN。
- feedback_epoch – 数据库实例在生成热备用信息时使用的纪元。

  热备用服务器是指在服务器处于恢复或备用模式时，数据库实例可以进行连接和查询。热备用反馈是有关处于热备用状态的数据库实例的信息。有关更多信息，请参阅有关热备用的 PostgreSQL 文档。
- feedback_xmin – 数据库实例使用的最小（最早）活动事务 ID。
- oldest_read_view_lsn – 数据库实例用于从存储中读取的最早 LSN。
- visibility_lag_in_msec – 此数据库实例落后于写入器数据库实例的程度。

要查看这些值如何随时间变化，请考虑以下事务块，其中表插入需要一个小时。

```
psql> BEGIN;
psql> INSERT INTO table1 SELECT Large_Data_That_Takes_1_Hr_To_Insert;
psql> COMMIT;
```

在某些情况下，在 BEGIN 语句之后主数据库集群和辅助数据库集群之间的网络会断开连接。如果是这样，辅助数据库集群的 durability_lag_in_msec 值开始增加。在 INSERT 语句的末尾，durability_lag_in_msec 值为 1 小时。但是，rpo_lag_in_msec 值为 0，因为在主数据库集群和辅助数据库集群之间提交的所有用户数据仍然相同。在 COMMIT 语句完成后，rpo_lag_in_msec 值会立即增加。

将 Amazon Aurora Global Database 与其他AWS服务结合使用

您可以将 Aurora Global Database 与其他AWS服务结合使用，例如 Amazon S3 和 AWS Lambda。这样做要求全局数据库中的所有 Aurora 数据库集群在各自的 AWS 区域 中具有相同的权限、外部函数等。由于可以将 Aurora 全局数据库中的只读 Aurora 辅助数据库集群提升为主数据库的角色，因此我们建议您提前在所有 Aurora 数据库集群上为计划用于 Aurora 全局数据库的任何服务设置写入权限。

下面的程序总结了对每项 AWS 服务 执行的操作。

从 Aurora Global Database 调用 AWS Lambda 函数

1. 对于构成 Aurora 全局数据库的所有 Aurora 集群，请执行从 Amazon Aurora MySQL 数据库集群中调用 Lambda 函数中的过程。
2. 对于 Aurora 全局数据库中的每个集群，设置新 IAM (IAM) 角色的 (ARN)。
3. 要允许 Aurora 全局数据库中的数据库用户调用 Lambda 函数，请将您在创建 IAM 角色以允许 Amazon Aurora 访问AWS服务中创建的角色与 Aurora 全局数据库中的每个集群关联。
4. 配置 Aurora 全局数据库中的每个集群，以允许建立到 Lambda 的出站连接。有关说明，请参阅启用从 Amazon Aurora 到其它 AWS 服务的网络通信。

从 Amazon S3 加载数据

1. 对于构成 Aurora 全局数据库的所有 Aurora 集群，请执行将数据从 Amazon S3 存储桶中的文本文件加载到 Amazon Aurora MySQL 数据库集群中的过程。
2. 对于全局数据库中的每个 Aurora 集群，将 `aurora_load_from_s3_role` 或 `aws_default_s3_role` 数据库集群参数设置为新 IAM 角色的 Amazon Resource Name (ARN)。如果没有为 `aurora_load_from_s3_role` 指定 IAM 角色，则 Aurora 使用在 `aws_default_s3_role` 中指定的 IAM 角色。
3. 要允许 Aurora 全局数据库中的数据库用户访问 S3，请将您在创建 IAM 角色以允许 Amazon Aurora 访问AWS服务中创建的角色与全局数据库中的每个 Aurora 集群关联。
4. 配置 Aurora 全局数据库中的每个集群，以允许建立到 S3 的出站连接。有关说明，请参阅启用从 Amazon Aurora 到其它 AWS 服务的网络通信。

将查询的数据保存到 Amazon S3

1. 对于构成 Aurora 全局数据库的所有 Aurora 集群，请执行将数据从 Amazon Aurora MySQL 数据库集群保存到 Amazon S3 存储桶中的文本文件或将数据从 Aurora PostgreSQL 数据库集群导出到 Amazon S3中的过程。
2. 对于全局数据库中的每个 Aurora 集群，将 `aurora_select_into_s3_role` 或 `aws_default_s3_role` 数据库集群参数设置为新 IAM 角色的 Amazon Resource Name (ARN)。如果没有为 `aurora_select_into_s3_role` 指定 IAM 角色，则 Aurora 使用在 `aws_default_s3_role` 中指定的 IAM 角色。
3. 要允许 Aurora 全局数据库中的数据库用户访问 S3，请将您在创建 IAM 角色以允许 Amazon Aurora 访问AWS服务中创建的角色与全局数据库中的每个 Aurora 集群关联。
4. 配置 Aurora 全局数据库中的每个集群，以允许建立到 S3 的出站连接。有关说明，请参阅启用从 Amazon Aurora 到其它 AWS 服务的网络通信。

升级 Amazon Aurora Global Database

按照与升级 Aurora 数据库集群相同的过程升级 Aurora Global Database。但是，以下是在开始该过程之前需要注意的一些重要区别。

建议您将主数据库集群和辅助数据库集群升级到相同版本。仅当主数据库集群和辅助数据库集群具有相同的主要、次要和补丁级别引擎版本时，您才能对 Aurora Global Database执行托管式跨区域数据库失效转移。但是，补丁级别可能会有所不同，具体取决于次要引擎版本。有关更多信息，请参阅 托管式跨区域切换和失效转移的补丁级别兼容性。

主要版本升级

当您对 Amazon Aurora Global Database 执行主要版本升级时，将升级全局数据库集群，而不是它包含的单个集群。

要了解如何将 Aurora PostgreSQL 全局数据库升级到更高的主要版本，请参阅 全局数据库的主要版本升级。

> **Note**
>
> 使用基于 Aurora PostgreSQL 的 Aurora Global Database时，如果启用了恢复点目标 (RPO) 功能，则无法对 Aurora 数据库引擎执行主要版本升级。有关 RPO 功能的信息，请参阅 管理基于 Aurora PostgreSQL 的全局数据库的 RPO。

要了解如何将 Aurora MySQL 全局数据库升级到更高的主要版本，请参阅 全局数据库的就地主要版本升级。

> **Note**
>
> 使用基于 Aurora MySQL 的 Aurora Global Database 时，只有当您将 `lower_case_table_names` 参数设置为默认值并且重启全球数据库之后，才能执行从 Aurora MySQL 版本 2 到版本 3 的就地升级。
> 要在使用 `lower_case_table_names` 时执行到 Aurora MySQL 版本 3 的主要版本升级，请使用以下过程：
>
> 1. 从全局集群中删除所有辅助区域。按照从 Amazon Aurora Global Database 删除集群中的步骤操作。
> 2. 将主区域的引擎版本升级到 Aurora MySQL 版本 3。按照如何执行就地升级中的步骤操作。
> 3. 向全局集群添加辅助区域。按照将 AWS 区域 添加到 Amazon Aurora Global Database中的步骤操作。
>
> 还可以改用快照还原方法。有关更多信息，请参阅 从数据库集群快照还原。

次要版本升级

对于 Aurora Global Database 的次要升级，请先升级所有辅助集群，然后再升级主集群。

要了解如何将 Aurora PostgreSQL 全局数据库升级到更高的次要版本，请参阅 如何执行次要版本升级和应用补丁。要了解如何将 Aurora MySQL 全局数据库升级到更高的次要版本，请参阅 通过修改引擎版本升级 Aurora MySQL。

在执行升级之前，请查看以下注意事项：

- 升级辅助集群的次要版本不会以任何方式影响主集群的可用性或使用情况。
- 辅助集群必须具有至少一个数据库实例才能执行次要版本升级。
- 如果您将 Aurora MySQL Global Database 升级到版本 2.11.*，则必须将主数据库集群和辅助数据库集群升级到完全相同的版本（包括补丁级别）。
- 如果您升级 Aurora PostgreSQL Global Database，则必须将主数据库集群和辅助数据库集群升级到完全相同的版本和补丁级别。要更新补丁级别，请在辅助集群上应用所有待处理的维护操作。

- 要支持托管式跨区域切换或失效转移，您可能需要将主数据库集群和辅助数据库集群升级到完全相同的版本（包括补丁级别）。此要求适用于 Aurora MySQL 和某些 Aurora PostgreSQL 版本。有关允许在运行不同补丁级别的集群之间进行切换和失效转移的版本列表，请参阅托管式跨区域切换和失效转移的补丁级别兼容性。

托管式跨区域切换和失效转移的补丁级别兼容性

如果您的 Aurora Global Database 正在运行以下次要引擎版本之一，那么即使主数据库集群和辅助数据库集群的补丁级别不匹配，您也可以执行托管式跨区域切换或失效转移。如果次要引擎版本低于此列表中的版本，您的主数据库集群和辅助数据库集群必须运行相同的主要版本、次要版本和补丁级别，才能执行托管式跨区域切换或失效转移。计划主集群和/或辅助集群的升级时，请务必查看下表中的版本信息和注释。

> **Note**
>
> 对于手动跨区域失效转移，只要目标辅助数据库集群运行的主要和次要引擎版本与主数据库集群相同，就可以执行失效转移过程。在这种情况下，补丁级别不需要匹配。
> 如果您的引擎版本需要相同的补丁级别，您可以按照执行 Aurora 全球数据库的手动失效转移中的步骤手动执行失效转移。

数据库引擎	次要引擎版本	备注
Aurora MySQL	没有次要版本	当主数据库集群和备用数据库集群之间的补丁级别不同时，所有 Aurora MySQL 次要版本都不允许执行托管式跨区域切换或失效转移。
Aurora PostgreSQL	• 版本 15 或更高的主要版本 • 版本 14.5 或更高的次要版本 • 版本 13.8 或更高的次要版本 • 版本 12.12 或更高的次要版本 • 版本 11.17 或更高的次要版本	使用上一列中列出的引擎版本，您可以执行以下托管式跨区域切换或失效转移：从具有一个补丁级别的主数据库集群到具有不同补丁级别的辅助数据库集群。 如果次要版本低于这些版本，仅当主数据库集群和辅助数据库集群的补丁

数据库引擎	次要引擎版本	备注
		级别匹配时，您才能执行托管式跨区域数据库切换或失效转移。 **Warning** 当您将全局数据库中的集群更新为以下任何补丁版本时，在全局数据库中的所有集群都运行这些补丁版本之一或更高的补丁版本之前，您将无法执行跨区域切换或失效转移。 • 补丁版本 16.1.6、16.2.4、16.3.2 和 16.4.2 • 补丁版本 15.3.8、15.4.9、15.5.6、15.6.4、15.7.2 和 15.8.2 • 补丁版本 14.8.8、14.9.9、14.10.6、14.11.4、14.12.2 和 14.13.2

适用于 Aurora 的 Amazon RDS 代理

通过使用 Amazon RDS 代理，您可以允许您的应用程序池化和共享数据库连接，以提高其扩展能力。RDS 代理通过在保留应用程序连接的同时自动连接到备用数据库实例，使应用程序能够更好地抵御数据库故障。使用 RDS 代理还使您能够为数据库强制执行 AWS Identity and Access Management (IAM) 身份验证，并将凭证安全地存储在 AWS Secrets Manager。

使用 RDS 代理，您可以处理不可预测的数据库流量突增。否则，这些突增情况可能会由于超额订阅连接或快速创建新连接而导致问题。RDS 代理建立数据库连接池，并重用该池中的连接。这种方法可避免每次打开新数据库连接所产生的内存和 CPU 开销。要防止数据库超额订阅，您可以控制创建的数据库连接数。

RDS 代理对无法立即从连接池提供服务的应用程序连接进行排队或限制。尽管延迟可能会增加，但您的应用程序可以继续扩展，而不会导致数据库突然出现故障或不堪重负。如果连接请求超出您指定的限制，则 RDS 代理会拒绝应用程序连接（即，减轻负载）。与此同时，它为 RDS 可通过可用容量提供服务的负载维护可预测的性能。

您可以减少处理凭证的开销，并为每个新连接建立安全连接。RDS 代理可以代表数据库处理其中的一些工作。

RDS 代理与它支持的引擎版本完全兼容。您可以在不更改代码的情况下为大多数应用程序启用 RDS 代理。有关支持的引擎版本列表，请参阅支持 Amazon RDS 代理的区域和 Aurora 数据库引擎。

主题

- 区域和版本可用性
- RDS 代理的配额和限制
- 规划在哪里使用 RDS 代理
- RDS 代理概念和术语
- 开始使用 RDS 代理
- 管理 RDS 代理
- 使用 Amazon RDS Proxy 终端节点
- 使用 Amazon CloudWatch 监控 RDS Proxy 指标
- 使用 RDS 代理事件
- RDS Proxy 命令行示例
- RDS 代理故障排除

- 将 RDS Proxy 与 AWS CloudFormation 一起使用
- 在 Aurora Global Database 中使用 RDS 代理

区域和版本可用性

有关给定 AWS 区域中的 RDS 代理的数据库引擎版本支持和可用性的信息，请参阅 支持 Amazon RDS 代理的区域和 Aurora 数据库引擎。

RDS 代理的配额和限制

以下配额和限制适用于 RDS 代理：

- 每个 AWS 账户 ID 限制为 20 个代理。如果您的应用程序需要更多代理，请通过 AWS Management Console中的服务配额页面申请增加代理。在服务配额页面中，选择 Amazon Relational Database Service（Amazon RDS），然后找到代理以申请增加配额。AWS 可以自动增加您的配额，或者等待支持 审核您的申请。
- 每个代理最多可拥有 200 个关联的 Secrets Manager 密钥。因此，每个代理可以在任何给定时间连接多达 200 个不同的用户账户。
- 每个代理都具有一个默认端点。您还可以为每个代理添加最多 20 个代理端点。您可以创建、查看、修改和删除这些端点。
- 在 Aurora 集群中，使用默认代理端点的所有连接都由 Aurora 写入器实例处理。要对读取密集型工作负载执行负载均衡，您可以为代理创建只读端点。该端点会将连接传递到集群的读取器端点。这样，您的代理连接就可以利用 Aurora 读取可扩展性。有关更多信息，请参阅“代理终端节点概述”。
- 您可以将 RDS 代理用于 Aurora Serverless v2 集群，但不能用于 Aurora Serverless v1 集群。
- 您的 RDS 代理必须与数据库位于同一 Virtual Private Cloud (VPC) 中。代理无法公开访问，但数据库可以。例如，如果在本地主机上进行数据库原型设计，则无法连接到代理，除非设置必要的网络要求来允许连接到代理。这是因为您的本地主机不在代理的 VPC 范围内。

  > **Note**
  >
  > 对于 Aurora 数据库集群，您可以启用跨 VPC 访问。为此，请为代理创建额外的端点，并为该端点指定其他 VPC、子网和安全组。有关更多信息，请参阅 访问 VPC 中的 Aurora 数据库。

- 您不能将 RDS 代理用于其租户设置为 `dedicated` 的 VPC。

- 如果您将 RDS 代理与启用了 IAM 身份验证的 Aurora 数据库集群结合使用，请检查用户身份验证。通过代理连接的用户必须通过登录凭证进行身份验证。有关 RDS 代理中 Secrets Manager 和 IAM 支持的详细信息，请参阅在 AWS Secrets Manager 中设置 RDS 代理的数据库凭证和设置 RDS 代理的 AWS Identity and Access Management（IAM）策略。
- 使用 SSL 主机名验证时，不能将 RDS 代理与自定义 DNS 一起使用。
- 每个代理都可以与单个目标数据库集群相关联。不过，您可以将多个代理与同一个数据库集群关联。
- 文本大小大于 16 KB 的任何语句都会导致代理将会话固定到当前连接。
- 某些区域在创建代理时需要考虑可用区（AZ）限制。美国东部（弗吉尼亚州北部）区域在 `use1-az3` 可用区中不支持 RDS 代理。美国西部（加利福尼亚北部）区域在 `usw1-az2` 可用区中不支持 RDS 代理。如果在创建代理时选择子网，请确保不要在上述可用区中选择子网。
- 目前，RDS 代理不支持任何全局条件上下文键。

  有关全局条件上下文键的更多信息，请参阅《IAM 用户指南》中的 AWS 全局条件上下文键。
- 不能将 RDS 代理与 RDS Custom for SQL Server 一起使用。
- 要将任何数据库参数组修改反映到您的代理，需要重启实例，即使您选择立即应用更改也不例外。对于集群级参数，需要在集群范围内重启。
- 当注册代理目标时，您的代理会自动创建 `rdsproxyadmin` 数据库用户。这是受保护的用户，对于代理功能至关重要。您应该避免以任何身份篡改 `rdsproxyadmin` 用户。删除或修改 `rdsproxyadmin` 用户或其权限可能导致应用程序完全无法使用代理。

有关每个数据库引擎的更多信息，请参阅下面几节：

- Aurora MySQL 的额外限制
- Aurora PostgreSQL 的额外限制

Aurora MySQL 的额外限制

以下附加限制应用于适用于 Aurora MySQL 数据库的 RDS 代理：

- RDS 代理对 `caching_sha2_password` 身份验证的支持需要安全（TLS）连接。
- 对 `caching_sha2_password` 的 RDS 代理支持已知与某些 go-sql 驱动程序版本存在兼容性问题。
- 使用 MySQL 8.4 C 驱动程序时，如果预编译语句中的参数计数超过占位符计数，`mysql_stmt_bind_named_param` API 可能会形成格式错误的数据包。这会导致错误的响应。有关更多信息，请参阅 MySQL 错误报告。

- 目前，所有代理均在端口 3306 上侦听 MySQL。代理仍使用您在数据库设置中指定的端口连接到您的数据库。
- 不能将 RDS 代理与 EC2 实例中自管理的 MySQL 数据库一起使用。
- 您不能将 RDS 代理与数据库参数组中的 `read_only` 参数设置为 1 的 RDS for MySQL 数据库实例一起使用。
- RDS 代理不支持 MySQL 压缩模式。例如，它不支持 `mysql` 命令的 `--compress` 或 `-C` 选项使用的压缩。
- 当 RDS 代理重用相同的数据库连接运行其他查询时，处理 `GET DIAGNOSTIC` 命令的数据库连接可能会返回不准确的信息。当 RDS 代理多路复用数据库连接时，可能会发生这种情况。
- 某些 SQL 语句和函数（如 `SET LOCAL`）可以更改连接状态而不会引起固定。有关最新的固定行为，请参阅 避免固定 RDS 代理。
- 不支持在多语句查询中使用 `ROW_COUNT()` 函数。
- RDS 代理不支持无法在一个 TLS 记录中处理多条响应消息的客户端应用程序。
- RDS 代理不支持 MySQL 双密码。

> **Important**
>
> 对于与 MySQL 数据库关联的代理，请勿在初始化查询中将配置参数 `sql_auto_is_null` 设置为 `true` 或非零值。否则，可能会导致不正确的应用程序行为。

Aurora PostgreSQL 的额外限制

以下附加限制应用于适用于 Aurora PostgreSQL 数据库的 RDS 代理：

- RDS 代理不支持 PostgreSQL 的会话固定筛选条件。
- 目前，所有代理均在端口 5432 上侦听 PostgreSQL。
- 对于 PostgreSQL，RDS 代理目前不支持通过发出 `CancelRequest` 来取消来自客户端的查询。例如，使用 Ctrl+C 取消交互式 psql 会话中长时间运行的查询时，就会出现这种情况。
- PostgreSQL 函数 lastval 的结果并不总是准确的。作为解决方法，请将 INSERT 语句与 `RETURNING` 子句一起使用。
- RDS 代理目前不支持流式复制模式。
- `default` 数据库必须存在。

- 如果您使用 ALTER ROLE 通过 SET ROLE 更改用户角色，但后续以该用户身份连接到代理时若遇到固定问题，则这些后续连接可能不会使用此角色设置。为防止出现这种情况，在使用代理时，请在代理的初始化查询中使用 SET ROLE。有关更多信息，请参阅 为 Amazon Aurora 创建代理中的初始化查询。

> **Important**
>
> 对于适用于 PostgreSQL 数据库的现有代理，如果您将数据库身份验证修改为仅使用 SCRAM，代理将在长达 60 秒的时间内不可用。要避免此问题，请执行以下操作之一：
>
> - 确保数据库同时允许 SCRAM 和 MD5 身份验证。
> - 要仅使用 SCRAM 身份验证，请创建一个新代理，将应用程序流量迁移到新代理，然后删除先前与数据库关联的代理。

规划在哪里使用 RDS 代理

您可以确定哪些数据库实例、集群和应用程序可能从使用 RDS Proxy 中受益匪浅。为此，请考虑以下因素：

- 任何出现“连接过多”错误的数据库集群都很适合与代理相关联。这通常以 ConnectionAttempts CloudWatch 指标的值较高为特征。代理允许应用程序打开许多客户端连接，而代理管理与数据库集群的少量长期连接。
- 对于使用较小 AWS 实例类（如 T2 或 T3）的数据库集群，使用代理可以帮助避免内存不足情况。它还可以帮助减少建立连接时的 CPU 开销。处理大量连接时可能会发生这些情况。
- 您可以监控某些 Amazon CloudWatch 指标，以确定数据库集群是否接近某些类型的限制。这些限制针对连接数以及与连接管理关联的内存。您还可以监控某些 CloudWatch 指标，以确定数据库集群是否正在处理许多短期连接。打开和关闭此类连接可能会给数据库带来性能开销。有关要监控的指标的信息，请参阅 使用 Amazon CloudWatch 监控 RDS Proxy 指标。
- AWS Lambda 函数也可以很好地使用代理。这些函数进行频繁的短数据库连接，可受益于 RDS Proxy 提供的连接池。您可以利用已为 Lambda 函数提供的任何 IAM 身份验证，而不是在 Lambda 应用程序代码中管理数据库凭证。
- 代理服务器非常适合那些通常需要打开和关闭大量数据库连接，并且没有内置的连接池机制的应用程序。

- 长时间保持大量连接打开的应用程序通常可以很好地使用代理。诸如软件即服务 (SaaS) 或电子商务等行业中的应用程序通常会使连接保持打开状态，从而最大限度地减少数据库请求的延迟。通过 RDS 代理，与直接连接到数据库集群相比，应用程序可以将更多连接保持为打开状态。
- 由于为所有数据库集群设置 IAM 身份验证和 Secrets Manager 较为复杂，因此，您可能尚未采用此类身份验证。如果是这样，您可以保留现有的身份验证方法，并将身份验证委派给代理。代理可以为特定应用程序的客户端连接强制执行身份验证策略。您可以利用已为 Lambda 函数提供的任何 IAM 身份验证，而不是在 Lambda 应用程序代码中管理数据库凭证。
- RDS 代理可以帮助使应用程序更具弹性，对数据库故障更透明。RDS 代理绕过域名系统（DNS）缓存，以将 Aurora 多可用区数据库的故障转移时间缩短多达 66%。RDS 代理还会自动将流量路由到新的数据库实例，同时保留应用程序连接。这使得失效转移对应用程序更加透明。

RDS 代理概念和术语

您可以通过使用 RDS 代理来简化 Amazon Aurora 数据库集群的连接管理。

RDS 代理处理客户端应用程序和数据库之间的网络流量。为实现这一点，它首先了解数据库协议。然后，它会根据应用程序中的 SQL 操作和数据库中的结果集来调整其行为。

RDS 代理减少了数据库上的连接管理的内存和 CPU 开销。当应用程序打开多个同时连接时，数据库所需的内存和 CPU 资源较少。而且也不需要应用程序中的逻辑来关闭和重新打开长时间处于空闲状态的连接。同样，在出现数据库问题时，需要较少的应用程序逻辑来重新建立连接。

RDS 代理的基础设施高度可用，部署在多个可用区 (AZ) 上。RDS 代理的计算、内存和存储独立于您的 Aurora 数据库集群。这种分离有助于减少数据库服务器的开销，使它们能够将资源用于处理数据库工作负载。RDS 代理计算资源是无服务器的，根据您的数据库工作负载自动进行扩展。

主题

- RDS 代理概念概述
- 连接池
- RDS 代理安全性
- 失效转移
- 事务

RDS 代理概念概述

RDS 代理处理基础设施以执行连接池和以下各节中所述的其他功能。您可以在代理页面上看到 RDS 控制台中表示的代理。

每个代理处理与单个 Aurora 数据库集群的连接。对于 Aurora 预调配集群，代理自动确定当前的写入器实例。

代理保持打开状态且可供数据库应用程序用于形成连接池的连接。

默认情况下，RDS 代理可在会话中的每个事务之后重用连接。这种事务级别的重用称为多路复用。当 RDS 代理临时从连接池中删除某个连接以重用该连接时，该操作称为借用连接。如果这样做是安全的，RDS 代理会将该连接返回到连接池。

在某些情况下，RDS 代理不能确定在当前会话之外重用数据库连接是否安全。在这些情况下，它将会话保持在同一个连接上，直到会话结束。此回退行为称为固定。

代理具有默认端点。使用 Amazon Aurora 数据库集群时，您将连接到该端点。您可以这样做，而不是连接到与集群直接连接的读/写端点。Aurora 集群的专用端点仍可供您使用。对于 Aurora 数据库集群，您还可以创建其它读/写和只读端点。有关更多信息，请参阅 代理终端节点概述。

例如，您仍可连接到集群端点以实现读/写连接，而无需连接池。您仍可连接到读取器端点以实现负载均衡的只读连接。您仍可连接到实例端点，以便对集群中的特定数据库实例进行诊断和故障排除。如果您使用其他 AWS 服务（例如 AWS Lambda）连接到 RDS 数据库，请将其连接设置更改为使用代理端点。例如，指定代理端点，以允许 Lambda 函数在利用 RDS 代理功能的同时访问数据库。

每个代理都包含一个目标组。该目标组包含代理可连接到的 Aurora 数据库集群。对于 Aurora 集群，默认情况下，目标组与该集群中的所有数据库实例相关联。这样，代理就可以连接到被提升为集群中的写入器实例的任何 Aurora 数据库实例。与代理关联的 Aurora 数据库集群称为该代理的目标。为方便起见，当您通过控制台创建代理时，RDS 代理也会创建相应的目标组并自动注册关联的目标。

引擎系列是使用相同数据库协议的相关数据库引擎集。您可以为创建的每个代理选择引擎系列。

连接池

每个代理都为其关联的 Aurora 数据库的写入器和读取器实例建立连接池。连接池 是一项优化操作，可减少与打开和关闭连接相关的开销，并可保持许多连接同时打开。该开销包括处理每个新连接所需的内存。关闭每个连接并打开一个新连接还涉及 CPU 开销。示例包括传输层安全性协议/安全套接字层（TLS/SSL）握手、身份验证、协商功能等。连接池可简化应用程序逻辑。无需编写应用程序代码，即可最大限度地减少同时打开的连接数量。

每个代理还执行连接多路复用，也称为连接重用。对于多路复用，RDS 代理使用一个底层数据库连接对事务执行所有操作。然后，RDS 可以对下一个事务使用不同的连接。您可以同时打开许多到代理的连接，而代理可确保打开的到数据库实例或集群的连接数量较少。这样做可进一步减少数据库服务器上连接的内存开销。这项技术还降低了出现“连接过多”错误的可能性。

RDS 代理安全性

RDS 代理使用现有的 RDS 安全机制，如 TLS/SSL 和 AWS Identity and Access Management (IAM)。有关这些安全功能的一般信息，请参阅 Amazon Aurora 中的安全性。此外，请务必熟悉 Aurora 如何处理身份验证、授权和其它安全领域。

RDS 代理可作为客户端应用程序和底层数据库之间的附加安全层。例如，即使底层数据库实例支持旧版 TLS，您也可以使用 TLS 1.3 连接到代理。您可以使用 IAM 角色连接到代理。即使代理使用本机用户和密码身份验证方法连接到数据库，也是如此。通过采用这项技术，您可以对数据库应用程序强制执行强身份验证要求，而无需对数据库实例本身进行成本高昂的迁移。

您将 RDS 代理使用的数据库凭证存储在 AWS Secrets Manager 中。代理访问的 Aurora 数据库集群的每个数据库用户都必须在 Secrets Manager 中拥有相应的密钥。您还可为 RDS 代理的用户设置 IAM 身份验证。这样一来，即使数据库使用本机密码身份验证，您也可以对数据库访问强制执行 IAM 身份验证。我们建议使用这些安全功能，而不是在应用程序代码中嵌入数据库凭证。

将 TLS/SSL 与 RDS 代理结合使用

您可以使用 TLS/SSL 协议连接到 RDS Proxy。

> **Note**
>
> RDS 代理使用来自 AWS Certificate Manager (ACM) 的证书。如果您正在使用 RDS Proxy，则无需下载 Amazon RDS 证书或更新使用 RDS 代理连接的应用程序。

要对代理与数据库之间的所有连接强制执行 TLS，您可以在 AWS Management Console中创建或修改代理时，指定需要传输层安全性设置。

RDS 代理还可以确保您的会话在客户端与 RDS 代理端点之间使用 TLS/SSL。要让 RDS 代理这样做，请在客户端上指定此要求。没有为使用 RDS 代理的数据库 SSL 连接设置 SSL 会话变量。

- 对于 Aurora MySQL，在运行 `mysql` 命令时使用 `--ssl-mode` 参数在客户端上指定此要求。
- 对于 和 Aurora PostgreSQL，在运行 `psql` 命令时将 `sslmode=require` 指定为 `conninfo` 字符串的一部分。

RDS 代理支持 TLS 协议版本 1.0、1.1、1.2 和 1.3。您可以使用比底层数据库中使用的更高版本的 TLS 连接到代理。

默认情况下，客户端程序会与 RDS 代理建立加密连接，并通过 `--ssl-mode` 选项进行进一步控制。从服务器端，RDS 代理支持所有 SSL 模式。

对于客户端，SSL 模式如下：

PREFERRED

SSL 是首选项，但不是必需项。

DISABLED

不允许使用 SSL。

REQUIRED

强制 SSL。

VERIFY_CA

强制 SSL 并验证证书颁发机构 (CA)。

VERIFY_IDENTITY

强制 SSL 并验证 CA 和 CA 主机名。

将客户端与 `--ssl-mode`VERIFY_CA 或 VERIFY_IDENTITY 结合使用时，以 `--ssl-ca` 格式指定指向 CA 的 `.pem` 选项。对于要使用的 `.pem` 文件，请从 Amazon Trust Services 下载所有根 CA PEM，然后把它们放入一个 `.pem` 文件。

RDS 代理使用通配符证书，这些证书应用到域及其子域。如果您使用 `mysql` 客户端以 SSL 模式 VERIFY_IDENTITY 进行连接，则您当前必须使用与 MySQL 8.0 兼容的 `mysql` 命令。

失效转移

失效转移是一项高可用性功能，可在原始实例变得不可用时将数据库实例替换为另一个数据库实例。可能会因为数据库实例出现问题而发生失效转移。失效转移也可能是正常维护程序的一部分，例如在数据库升级期间。失效转移还适用于除写入器实例外还具有一个或多个读取器实例的 Aurora 数据库集群。

通过代理进行连接可使您的应用程序对数据库失效转移更具弹性。当原始数据库实例变得不可用时，RDS 代理将连接到备用数据库，而不删除空闲应用程序连接。这有助于加快和简化失效转移过程。这与典型的重启或数据库问题相比，对应用程序造成的干扰更小。

如果没有 RDS Proxy，失效转移会导致短暂中断。在中断期间，您无法在失效转移中对该数据库执行写入操作。任何现有数据库连接都会中断，您的应用程序必须重新打开这些连接。当一个只读数据库实例被提升以替代不可用的数据库实例时，数据库将可用于新连接和写入操作。

在数据库失效转移期间，RDS Proxy 将继续接受相同 IP 地址的连接，并自动将连接定向到新的主数据库实例。通过 RDS 代理连接的客户端不会受到以下情况的影响：

- 失效转移时的域名系统 (DNS) 传播延迟。
- 本地 DNS 缓存。
- 连接超时。
- 不确定哪个数据库实例是当前的写入器。
- 等待来自以前写入器的查询响应，该写入器在未关闭连接的情况下变得不可用。

对于维护自身连接池的应用程序，完成 RDS Proxy 意味着大多数连接在失效转移或其他中断期间保持活动状态。只有处于事务或 SQL 语句中间的连接才会被取消。RDS 代理会立即接受新连接。当数据库写入器不可用时，RDS 代理将对传入的请求进行排队。

对于不维护自身连接池的应用程序，RDS 代理提供更快的连接速度和更多的打开连接。它减轻了频繁从数据库重新连接而产生的昂贵开销。这是通过重用 RDS 代理连接池中维护的数据库连接来实现。此方法对于 TLS 连接尤其重要，因为这些连接的设置成本很高。

事务

单个事务中的所有语句始终使用相同的底层数据库连接。当事务结束时，连接可供另一会话使用。使用事务作为粒度单位会产生以下后果：

- 开启 Aurora MySQL `autocommit` 设置后，每个单独的语句之后都会发生连接重用。
- 相反，关闭 `autocommit` 设置后，您在会话中发出的第一个语句会开始新的事务。例如，假设您输入 `SELECT`、`INSERT`、`UPDATE` 序列和其他数据操作语言（DML）语句。在这种情况下，在您发出 `COMMIT`、`ROLLBACK` 或以其他方式结束事务之前，不会发生连接重用。
- 输入数据定义语言 (DDL) 语句会导致事务在该语句完成后结束。

RDS 代理通过数据库客户端应用程序使用的网络协议检测事务何时结束。事务检测不依赖于关键字，例如 SQL 语句文本中显示的 `COMMIT` 或 `ROLLBACK`。

在某些情况下，RDS 代理可能会检测到使得无法将会话移动到其他连接的数据库请求。在这些情况下，它会在会话的剩余时间关闭该连接的多路复用。当 RDS 代理不能确定多路复用是否适用于会话时，这一规则也适用。该操作称为固定。有关检测和最小化固定的方法，请参阅 避免固定 RDS 代理。

开始使用 RDS 代理

使用以下页面中的信息来设置、管理 适用于 Aurora 的 Amazon RDS 代理以及设置相关的安全选项。这些安全选项控制哪些人可以访问每个代理，以及每个代理如何连接到数据库实例。

如果您不熟悉 RDS 代理，我们建议您按照我们展示页面的顺序进行操作。

主题

- 设置 RDS 代理的网络先决条件
- 在 AWS Secrets Manager 中设置 RDS 代理的数据库凭证
- 设置 RDS 代理的 AWS Identity and Access Management（IAM）策略
- 为 Amazon Aurora 创建代理
- 查看代理
- 通过 RDS Proxy 连接到数据库

设置 RDS 代理的网络先决条件

使用 RDS 代理需要您在 Aurora 数据库集群和 RDS 代理之间拥有通用的虚拟私有云（VPC）。此 VPC 应至少有两个位于不同可用区中的子网。您的账户可以拥有这些子网，或与其他账户共享它们。有关 VPC 共享的信息，请参阅使用共享 VPC。

您的客户端应用程序资源（例如 Amazon EC2、Lambda 或 Amazon ECS）可以与代理位于同一 VPC 中。它们也可以位于与代理不同的 VPC 中。如果已成功连接到任何 Aurora 数据库集群，则您已拥有所需的网络资源。

主题

- 获取有关您的子网的信息
- 计划 IP 地址容量

获取有关您的子网的信息

如果您刚刚开始使用 Aurora，您可以遵循为 Amazon Aurora 设置环境中的过程，了解连接到数据库的基础知识。您也可以按照 开始使用 Amazon Aurora 中的教程进行操作。

要创建代理，必须提供代理在其中运行的子网和 VPC。以下 Linux 示例展示了检查 AWS 账户 拥有的 VPC 和子网的 AWS CLI 命令。尤其是，在使用 CLI 创建代理时，您可以将子网 ID 作为参数传递。

```
aws ec2 describe-vpcs
aws ec2 describe-internet-gateways
aws ec2 describe-subnets --query '*[].[VpcId,SubnetId]' --output text | sort
```

以下 Linux 示例展示了用于确定与特定 Aurora 数据库集群对应的子网 ID 的 AWS CLI 命令。

对于 Aurora 集群，首先，您可以找到其中一个关联数据库实例的 ID。您可以提取该数据库实例使用的子网 ID。为此，请在数据库实例的描述输出中检查 DBSubnetGroup 和 Subnets 属性中的嵌套字段。在为该数据库服务器设置代理时，您可以指定部分或全部这些子网 ID。

```
$ # Find the ID of any DB instance in the cluster.
$  aws rds describe-db-clusters --db-cluster-identifier my_cluster_id --query '*[].
[DBClusterMembers]||[0]||[0][*].DBInstanceIdentifier' --output text
```

```
my_instance_id
instance_id_2
instance_id_3
```

找到数据库实例标识符后，检查关联的 VPC 以查找其子网。以下 Linux 示例展示了操作步骤。

```
$ #From the DB instance, trace through the DBSubnetGroup and Subnets to find the subnet
 IDs.
$ aws rds describe-db-instances --db-instance-identifier my_instance_id --query '*[].
[DBSubnetGroup]||[0]||[0]||[Subnets]||[0]||[*].SubnetIdentifier' --output text
```

```
subnet_id_1
subnet_id_2
subnet_id_3
...
```

```
$ #From the DB instance, find the VPC.
```

```
$ aws rds describe-db-instances --db-instance-identifier my_instance_id --query '*[].
[DBSubnetGroup]|[0]|[0].VpcId' --output text
```

```
my_vpc_id
```

```
$ aws ec2 describe-subnets --filters Name=vpc-id,Values=my_vpc_id --query '*[].
[SubnetId]' --output text
```

```
subnet_id_1
subnet_id_2
subnet_id_3
subnet_id_4
subnet_id_5
subnet_id_6
```

计划 IP 地址容量

RDS 代理会根据向它注册的数据库实例的大小和数量，视需要自动调整其容量。某些操作可能还需要更多代理容量，例如增加注册数据库的大小或内部 RDS 代理维护操作。在这些操作期间，您的代理可能需要更多 IP 地址来预调配额外的容量。这些额外的地址使您的代理可以在不影响工作负载的情况下进行扩展。您的子网中缺少可用的 IP 地址会阻止代理纵向扩展。这可能导致查询延迟更长或客户端连接故障。当您的子网中没有足够的可用 IP 地址时，RDS 会通过事件 RDS-EVENT-0243 通知您。有关此事件的信息，请参阅 使用 RDS 代理事件。

根据数据库实例类大小，您应在子网中为代理保留以下最少可用的 IP 地址数。

数据库实例类	最少可用 IP 地址数
db.*.xlarge 或更小	10
db.*.2xlarge	15
db.*.4xlarge	25
db.*.8xlarge	45
db.*.12xlarge	60
db.*.16xlarge	75

数据库实例类	最少可用 IP 地址数
db.*.24xlarge	110

这些建议的 IP 地址数是针对仅具有默认端点的代理的估计数。具有更多端点或只读副本的代理可能需要更多可用的 IP 地址。对于每个其他端点，建议您另外预留三个 IP 地址。对于每个只读副本，建议您根据该只读副本的大小，保留表中指定的额外 IP 地址。

> **Note**
>
> RDS 代理在一个 VPC 中支持的 IP 地址数不超过 215 个。

例如，假设您要估算与 Aurora 数据库集群关联的代理所需的 IP 地址数。

在这种情况下，假设如下：

- 您的 Aurora 数据库集群有 1 个大小为 db.r5.8xlarge 的写入器实例和 1 个大小为 db.r5.2xlarge 的读取器实例。
- 连接到此数据库集群的代理具有默认端点和 1 个具有只读角色的自定义端点。

在这种情况下，代理需要大约 63 个可用 IP 地址（写入器实例需要 45 个，读取器实例需要 15 个，附加的自定义端点需要 3 个）。

在 AWS Secrets Manager 中设置 RDS 代理的数据库凭证

Amazon RDS 中的 RDS 代理使用 AWS Secrets Manager 安全地存储和管理数据库凭证。您无需在应用程序中嵌入凭证，而是将代理与包含必要身份验证详细信息的 Secrets Manager 密钥相关联。您可以在 Aurora 数据库集群上，为代理连接到的每个数据库用户账户分别创建 Secrets Manager 密钥。

主题

- 创建与 RDS 代理结合使用的密钥

创建与 RDS 代理结合使用的密钥

在创建代理之前，必须先创建至少一个用于存储数据库凭证的密钥。

控制台

创建密钥

1. 通过 https://console.aws.amazon.com/secretsmanager/ 打开 Secrets Manager 控制台。
2. 选择存储新密钥。
3. 选择 Amazon RDS 数据库的凭证。
4. 输入用户名和密码。您输入的凭证必须与关联的 RDS 数据库中存在的数据库用户的凭证相匹配。RDS 代理使用这些凭证代表应用程序对数据库进行身份验证和建立连接。

   如果不匹配，您可以更新密钥来与数据库密码匹配。在更新密钥之前，尝试使用该密钥通过代理进行连接会失败，但使用其它有效密钥的连接仍然有效。

   > **Note**
   >
   > 对于 RDS For SQL Server，无论数据库实例排序规则设置如何，RDS 代理都需要在 Secrets Manager 中设置一个区分大小写的密钥。如果应用程序支持用户名使用不同的大小写，例如“Admin”和“admin”，则必须为每个用户名创建单独的密钥。RDS 代理不支持在客户端和代理之间进行不区分大小写的用户名身份验证。
   > 有关 SQL Server 中的排序规则的更多信息，请参阅 Microsoft SQL Server 文档。

5. 对于数据库，选择密钥将访问的 Amazon RDS 数据库。
6. 填写密钥的其它设置，然后选择存储。有关全面的说明，请参阅《AWS Secrets Manager User Guide》中的 Creating an AWS Secrets Manager secret。

AWS CLI

通过 AWS CLI 创建代理时，请指定相应密钥的 Amazon 资源名称（ARN）。请为代理可以访问的所有数据库用户账户执行此操作。在 AWS Management Console 中，根据描述性名称选择密钥。

- 要创建与 RDS 代理结合使用的 Secrets Manager 密钥，请使用 create-secret 命令：

```
aws secretsmanager create-secret \
  --name "secret_name" \
  --description "secret_description" \
  --region region_name \
  --secret-string '{"username":"db_user","password":"db_user_password"}'
```

- 您也可以创建自定义密钥来加密 Secrets Manager 密钥。以下命令创建一个密钥示例。

```
aws kms create-key --description "test-key" --policy '{
  "Id":"kms-policy",
  "Version":"2012-10-17",
  "Statement":
    [
      {
        "Sid":"Enable IAM User Permissions",
        "Effect":"Allow",
        "Principal":{"AWS":"arn:aws:iam::account_id:root"},
        "Action":"kms:*","Resource":"*"
      },
      {
        "Sid":"Allow access for Key Administrators",
        "Effect":"Allow",
        "Principal":
          {
            "AWS":
              ["$USER_ARN","arn:aws:iam:account_id::role/Admin"]
          },
        "Action":
          [
            "kms:Create*",
            "kms:Describe*",
            "kms:Enable*",
            "kms:List*",
            "kms:Put*",
            "kms:Update*",
            "kms:Revoke*",
            "kms:Disable*",
            "kms:Get*",
            "kms:Delete*",
            "kms:TagResource",
            "kms:UntagResource",
            "kms:ScheduleKeyDeletion",
            "kms:CancelKeyDeletion"
          ],
        "Resource":"*"
      },
      {
        "Sid":"Allow use of the key",
        "Effect":"Allow",
        "Principal":{"AWS":"$ROLE_ARN"},
        "Action":["kms:Decrypt","kms:DescribeKey"],
```

```
        "Resource":"*"
      }
    ]
}'
```

例如，以下命令为两个数据库用户创建 Secrets Manager 密钥：

```
aws secretsmanager create-secret \
  --name secret_name_1 --description "db admin user" \
  --secret-string '{"username":"admin","password":"choose_your_own_password"}'

aws secretsmanager create-secret \
  --name secret_name_2 --description "application user" \
  --secret-string '{"username":"app-user","password":"choose_your_own_password"}'
```

要创建使用自定义 AWS KMS 密钥加密的这些密钥，请使用以下命令：

```
aws secretsmanager create-secret \
  --name secret_name_1 --description "db admin user" \
  --secret-string '{"username":"admin","password":"choose_your_own_password"}' \
  --kms-key-id arn:aws:kms:us-east-2:account_id:key/key_id

aws secretsmanager create-secret \
  --name secret_name_2 --description "application user" \
  --secret-string '{"username":"app-user","password":"choose_your_own_password"}' \
  --kms-key-id arn:aws:kms:us-east-2:account_id:key/key_id
```

要查看您的 AWS 账户拥有的密钥，请使用 list-secrets 命令：

```
aws secretsmanager list-secrets
```

使用 CLI 创建代理时，将一个或多个密钥的 Amazon Resource Name (ARN) 传递给了 `--auth` 参数。以下示例展示了如何准备报告，其中仅包含 AWS 账户所拥有的每个密钥的名称和 ARN。此示例使用了 `--output table` 版本 2 中提供的 AWS CLI 参数。如果您使用的是 AWS CLI 版本 1，请改用 `--output text`。

```
aws secretsmanager list-secrets --query '*[].[Name,ARN]' --output table
```

要确认密钥包含格式正确的凭证，请使用 get-secret-value 命令。将 *your_secret_name* 替换为密钥的短名称或 ARN。

```
aws secretsmanager get-secret-value --secret-id your_secret_name
```

输出包含一行 JSON 编码值，类似于以下内容：

```
...
"SecretString": "{\"username\":\"your_username\",\"password\":\"your_password\"}",
...
```

设置 RDS 代理的 AWS Identity and Access Management（IAM）策略

在 Secrets Manager 中创建密钥后，请创建一个可以访问这些密钥的 IAM 策略。有关使用 IAM 的一般信息，请参阅Amazon Aurora 的 Identity and Access Management。

> **Tip**
>
> 如果您使用 IAM 控制台，则应用下面的过程。如果您使用 AWS Management Console for RDS，RDS 可以自动为您创建 IAM 策略。在这种情况下，您可以跳过下面的过程。

创建用于访问您的 Secrets Manager 密钥以便与代理一起使用的 IAM 策略

1. 登录到 IAM 控制台。按照创建 IAM 角色中所述的创建角色过程，选择创建角色以将权限委派给 AWS 服务。

   对于可信实体类型，选择 AWS 服务。在使用案例下，从其他 AWS 服务的使用案例下拉列表中选择 RDS。选择 RDS – 将角色添加到数据库。

2. 对于新角色，请执行添加内联策略步骤。使用与编辑 IAM 策略中相同的一般过程。将下面的 JSON 粘贴到 JSON 文本框中。替换您自己的账户 ID。将您的 AWS 区域替换为 `us-east-2`。用 Amazon 资源名称（ARN）替换您创建的密钥，请参阅在 IAM 策略语句中指定 KMS 密钥。对于 `kms:Decrypt` 操作，请替换为默认 AWS KMS key 的 ARN 或您自己的 KMS 密钥。具体使用哪一项取决于您使用哪一项来加密 Secrets Manager 密钥。

   ```
   {
       "Version": "2012-10-17",
       "Statement": [
           {
               "Sid": "VisualEditor0",
               "Effect": "Allow",
   ```

```
            "Action": "secretsmanager:GetSecretValue",
            "Resource": [
                "arn:aws:secretsmanager:us-east-2:account_id:secret:secret_name_1",
                "arn:aws:secretsmanager:us-east-2:account_id:secret:secret_name_2"
            ]
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": "kms:Decrypt",
            "Resource": "arn:aws:kms:us-east-2:account_id:key/key_id",
            "Condition": {
                "StringEquals": {
                    "kms:ViaService": "secretsmanager.us-east-2.amazonaws.com"
                }
            }
        }
    ]
}
```

3. 编辑此 IAM 角色的信任策略。将下面的 JSON 粘贴到 JSON 文本框中。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "rds.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

以下命令通过 AWS CLI 执行相同的操作。

```
PREFIX=my_identifier
USER_ARN=$(aws sts get-caller-identity --query "Arn" --output text)

aws iam create-role --role-name my_role_name \
```

```
  --assume-role-policy-document '{"Version":"2012-10-17","Statement":
[{"Effect":"Allow","Principal":{"Service":
["rds.amazonaws.com"]},"Action":"sts:AssumeRole"}]}'

ROLE_ARN=arn:aws:iam::account_id:role/my_role_name

aws iam put-role-policy --role-name my_role_name \
  --policy-name $PREFIX-secret-reader-policy --policy-document '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "secretsmanager:GetSecretValue",
            "Resource": [
                "arn:aws:secretsmanager:us-east-2:account_id:secret:secret_name_1",
                "arn:aws:secretsmanager:us-east-2:account_id:secret:secret_name_2"
            ]
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": "kms:Decrypt",
            "Resource": "arn:aws:kms:us-east-2:account_id:key/key_id",
            "Condition": {
                "StringEquals": {
                    "kms:ViaService": "secretsmanager.us-east-2.amazonaws.com"
                }
            }
        }
    ]
}
```

为 Amazon Aurora 创建代理

您可以使用 Amazon RDS 代理，通过更高效地建立连接池和管理数据库失效转移，来提高数据库应用程序的可扩展性、可用性和安全性。本主题将引导您完成创建代理的过程。在开始之前，请确保您的数据库满足必要的先决条件，包括 IAM 权限和 VPC 配置。

您可以将代理与 Aurora MySQL 或 Aurora PostgreSQL 数据库集群相关联。

控制台

创建代理

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Proxies (代理)。
3. 选择 Create proxy (创建代理)。
4. 为代理配置以下设置。

设置	描述
引擎系列	代理在解释进出数据库的网络流量时识别的数据库网络协议。 **Note** 要使用 Aurora PostgreSQL，请确保将 `postgres` 数据库保留在实例中。请参阅对已删除的 `postgres` 数据库进行故障排除。
代理标识符	在您的 AWS 账户 ID 和当前 AWS 区域内具有唯一性的名称。
空闲客户端连接超时	如果客户端连接在设定的时间段内保持空闲，则代理会关闭客户端连接。默认情况下，这是 1800 秒（30 分钟）。当应用程序在完成上一个请求后的指定时间内未提交新请求时，连接处于空闲状态。代理将底层数据库连接保持为打开状态，并将其返回到连接池，使其可用于新的客户端连接。 要主动移除过时的连接，请减少空闲客户端连接超时。要最大限度地降低工作负载高峰期间的连接成本，请增加此超时。
数据库	要通过此代理访问的 Aurora 数据库集群。该列表仅包含具有兼容数据库引擎、引擎版本和其他设置的数据库实例和集群。如果列表为空，请创建与 RDS Proxy 兼容的新数据库实例或集群。为此，请按照创建 Amazon Aurora 数据库集群中的过程操作 然后，尝试再次创建代理。

设置	描述
连接池最大连接数	一个介于 1 到 100 之间的值，用于定义 RDS 代理可以使用的 max_connections 限制的百分比。如果您只打算对此数据库实例或集群使用一个代理，请将此值设置为 100。有关 RDS 代理如何使用此设置的更多信息，请参阅 the section called “MaxConnectionsPercent”。
会话绑定筛选条件	防止 RDS 代理绑定某些检测到的会话状态来绕过多路复用连接的默认安全措施。目前，PostgreSQL 不支持此设置，唯一可用的选项是 EXCLUDE_VARIABLE_SETS 。启用它可能会导致来自一个连接的会话变量影响其它连接，如果查询依赖于在当前事务之外设置的会话变量，则会导致错误或正确性问题。只有在确认您的应用程序可以安全地共享数据库连接后，才能使用此选项。 以下模式被认为是安全的： • SET 语句，其中没有更改有效会话变量值。换句话说，会话变量没有变化。 • 您可以更改会话变量值并在同一个事务中执行一条语句。 有关更多信息，请参阅 避免固定 RDS 代理。
连接借用超时	如果您希望代理使用所有可用的数据库连接，请设置在代理返回超时错误之前的等待时间。最多可以指定五分钟。仅当代理已达到最大连接数并且所有连接都在使用中时，此设置才适用。

设置	描述
初始化查询	（可选）添加初始化查询或修改当前查询。您可以为代理指定一个或多个 SQL 语句，以便在打开每个新数据库连接时运行。此设置通常与 SET 语句一起使用，以确保每个连接具有完全相同的设置。确保您添加的查询有效。要在单个 SET 语句中包含多个变量，请使用逗号分隔符。例如： `SET variable1 =value1, variable2 =value2` 对于多个语句，请使用分号作为分隔符。 **Important** 由于您可以将初始化查询作为目标组配置的一部分进行访问，因此初始化查询不受身份验证或加密方法的保护。有权查看或管理代理目标组配置的任何人都可以查看初始化查询。您不应将敏感数据（例如密码或长期保存的加密密钥）添加至此选项。
AWS Identity and Access Management（IAM）角色	一个有权访问 Secrets Manager 密钥的 IAM 角色，这些密钥代表代理可以使用的数据库用户账户的凭证。或者，您可以从 AWS Management Console创建新的 IAM 角色。
Secrets Manager 密钥	至少选择一个 Secrets Manager 密钥，该密钥包含允许代理访问 Aurora 数据库集群的数据库用户凭证。
客户端身份验证类型	代理对来自客户端的连接使用的身份验证类型。您的选择适用于与此代理关联的所有 Secrets Manager 密钥。如果您需要为每个密钥指定不同的客户端身份验证类型，请使用 AWS CLI 或改用 API 创建代理。
IAM 身份验证	是要求还是禁止对代理连接进行 IAM 身份验证。您的选择适用于与此代理关联的所有 Secrets Manager 密钥。如果您需要为每个密钥指定不同的 IAM 身份验证，请使用 AWS CLI 或改用 API 创建代理。

设置	描述
需要传输层安全性	对所有客户端连接强制执行 TLS/SSL。代理对其与底层数据库的连接使用相同的加密设置，无论客户端连接是已加密还是未加密。
子网	此字段预填充与您的 VPC 关联的所有子网。您可以移除代理不需要的任何子网，但必须至少保留两个子网。
VPC 安全组	选择现有 VPC 安全组或者从 AWS Management Console创建一个新的安全组。配置入站规则以支持应用程序访问代理，并配置出站规则以支持来自数据库目标的流量。 **Note** 此安全组必须支持从代理到数据库的连接。它既用于从应用程序到代理的入口，也用于从代理到数据库的出口。例如，如果您对数据库和代理使用相同的安全组，则确保该安全组中的资源可以相互通信。 使用共享 VPC 时，避免使用 VPC 的默认安全组或与另一个账户关联的安全组。而是选择属于您的账户的安全组。如果不存在安全组，则创建一个安全组。有关更多信息，请参阅使用共享 VPC。 RDS 跨多个可用区部署代理以确保高可用性。要启用跨可用区通信，代理子网的网络访问控制列表（ACL）必须支持引擎端口上的出口和所有端口上的入口。有关网络 ACL 的更多信息，请参阅使用网络 ACL 控制指向子网的流量。如果代理和目标的网络 ACL 相同，则必须添加 TCP 协议入口规则，其中源设置为 VPC CIDR。您还必须添加特定于引擎端口的 TCP 协议出口规则，其中目标设置为 VPC CIDR。

设置	描述
激活增强的日志记录	启用该设置可解决代理兼容性或性能问题。启用后，RDS 代理会记录详细的性能信息，有助于您调试 SQL 行为或代理连接性能和可扩展性。 仅启用此设置来进行调试，并确保已采取适当的安全措施来保护日志中的敏感信息。为了最大限度地减少开销，RDS 代理会在激活后 24 小时自动禁用此设置。可以临时使用它来解决特定问题。

5. 选择 Create proxy (创建代理)。

AWS CLI

要使用 AWS CLI 创建代理，请使用以下必需的参数调用 create-db-proxy 命令：

- `--db-proxy-name`
- `--engine-family`
- `--role-arn`
- `--auth`
- `--vpc-subnet-ids`

`--engine-family` 值区分大小写。

Example

对于 Linux、macOS 或 Unix：

```
aws rds create-db-proxy \
    --db-proxy-name proxy_name \
    --engine-family { MYSQL | POSTGRESQL | SQLSERVER } \
    --auth ProxyAuthenticationConfig_JSON_string \
    --role-arn iam_role \
    --vpc-subnet-ids space_separated_list \
    [--vpc-security-group-ids space_separated_list] \
    [--require-tls | --no-require-tls] \
    [--idle-client-timeout value] \
    [--debug-logging | --no-debug-logging] \
    [--tags comma_separated_list]
```

对于 Windows：

```
aws rds create-db-proxy ^
    --db-proxy-name proxy_name ^
    --engine-family { MYSQL | POSTGRESQL | SQLSERVER } ^
    --auth ProxyAuthenticationConfig_JSON_string ^
    --role-arn iam_role ^
    --vpc-subnet-ids space_separated_list ^
    [--vpc-security-group-ids space_separated_list] ^
    [--require-tls | --no-require-tls] ^
    [--idle-client-timeout value] ^
    [--debug-logging | --no-debug-logging] ^
    [--tags comma_separated_list]
```

以下是 `--auth` 选项的 JSON 值的示例。此示例对每个密钥应用不同的客户端身份验证类型。

```
[
  {
    "Description": "proxy description 1",
    "AuthScheme": "SECRETS",
    "SecretArn": "arn:aws:secretsmanager:us-
west-2:123456789123:secret/1234abcd-12ab-34cd-56ef-1234567890ab",
    "IAMAuth": "DISABLED",
    "ClientPasswordAuthType": "POSTGRES_SCRAM_SHA_256"
  },

  {
    "Description": "proxy description 2",
    "AuthScheme": "SECRETS",
    "SecretArn": "arn:aws:secretsmanager:us-
west-2:111122223333:secret/1234abcd-12ab-34cd-56ef-1234567890cd",
    "IAMAuth": "DISABLED",
    "ClientPasswordAuthType": "POSTGRES_MD5"

  },

  {
    "Description": "proxy description 3",
    "AuthScheme": "SECRETS",
    "SecretArn": "arn:aws:secretsmanager:us-
west-2:111122221111:secret/1234abcd-12ab-34cd-56ef-1234567890ef",
    "IAMAuth": "REQUIRED"
  }
```

```
]
```

> **Tip**
>
> 如果您尚不知道要用于 `--vpc-subnet-ids` 参数的子网 ID，请参阅 设置 RDS 代理的网络先决条件 获取有关如何查找它们的示例。

> **Note**
>
> 安全组必须允许访问代理连接到的数据库。同一安全组用于从应用程序到代理的入口以及从代理到数据库的出口。例如，假设您对数据库和代理使用同一安全组。在这种情况下，请确保您指定该安全组中的资源可以与同一安全组中的其他资源进行通信。
> 使用共享 VPC 时，您不能使用 VPC 的默认安全组，也不能使用属于其他账户的安全组。选择属于您账户的安全组。如果不存在，请创建一个。有关此限制的详细信息，请参阅使用共享 VPC。

要为代理创建正确的关联，您还可以使用 register-db-proxy-targets 命令。指定目标组名称 `default`。创建每个代理时，RDS Proxy 会自动创建一个此名称的目标组。

```
aws rds register-db-proxy-targets
    --db-proxy-name value
    [--target-group-name target_group_name]
    [--db-instance-identifiers space_separated_list]  # rds db instances, or
    [--db-cluster-identifiers cluster_id]        # rds db cluster (all instances)
```

RDS API

要创建 RDS 代理，请调用 Amazon RDS API 操作 CreateDBProxy。传递具有 AuthConfig 数据结构的参数。

创建每个代理时，RDS Proxy 会自动创建一个名为 `default` 的目标组。通过调用函数 RegisterDBProxyTargets，将 Aurora 数据库集群与目标组关联。

查看代理

创建一个或多个 RDS 代理后，您可以在 AWS Management Console、AWS CLI 或 RDS API 中查看和管理它们。您可以查看其配置详细信息，监控性能，并根据需要确定要修改或删除哪些代理。

要使数据库应用程序能够通过代理路由流量，必须在连接字符串中指定代理端点。

控制台

在控制台中查看代理

1. 登录AWS Management Console并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Proxies (代理)。
3. 选择代理名称以查看其详细信息。
4. 在详细信息页面上，目标组部分显示了代理与特定 Aurora 数据库集群关联的方式。您可以导航到默认目标组页面，以便更深入地查看此关联，包括在创建代理期间定义的配置设置。这些设置包括最大连接百分比、连接借用超时、引擎系列和会话绑定筛选条件。

CLI

要使用 CLI 查看您的代理，请使用 describe-db-proxies 命令。默认情况下，该请求会返回由您的 AWS 账户拥有的所有代理。要查看单个代理的详细信息，请通过 `--db-proxy-name` 参数指定其名称。

```
aws rds describe-db-proxies [--db-proxy-name proxy_name]
```

要查看与代理关联的其它信息，请使用以下命令。

```
aws rds describe-db-proxy-target-groups  --db-proxy-name proxy_name

aws rds describe-db-proxy-targets --db-proxy-name proxy_name
```

使用以下命令序列查看有关与代理关联的内容的更多详细信息：

1. 要获取代理列表，请运行 describe-db-proxies。
2. 要显示连接参数（如代理可以使用的最大连接百分比），请运行 describe-db-proxy-target-groups `--db-proxy-name`。使用代理的名称作为参数值。
3. 要查看与返回的目标组关联的 Aurora 数据库集群的详细信息，请运行 describe-db-proxy-targets。

RDS API

要使用 RDS API 查看您的代理，请使用 DescribeDBProxies 操作。该操作将返回 DBProxy 数据类型的值。

要查看代理连接设置的详细信息，请将此返回值中的代理标识符与 DescribeDBProxyTargetGroups 操作一起使用。该操作将返回 DBProxyTargetGroup 数据类型的值。

要查看与代理关联的 RDS 实例或 Aurora 数据库集群，请使用 DescribeDBProxyTargets 操作。该操作将返回 DBProxyTarget 数据类型的值。

通过 RDS Proxy 连接到数据库

通过代理连接到 Aurora 数据库集群或使用 Aurora Serverless v2 的集群的方式通常与直接连接到数据库的方式相同。主要区别在于您指定代理端点而不是集群端点。默认情况下，所有代理连接都具有读/写功能，并使用写入器实例。如果您通常使用读取器端点进行只读连接，则可以为代理创建额外的只读端点。您可以通过相同的方式使用该端点。有关更多信息，请参阅 代理终端节点概述。

主题

- 使用本机身份验证连接到代理
- 使用 IAM 身份验证连接到代理
- 使用 PostgreSQL 连接到代理的注意事项

使用本机身份验证连接到代理

执行以下步骤，使用本机身份验证连接到代理：

1. 查找代理端点。在 AWS Management Console 中，您可以在相应代理的详细信息页面上找到端点。通过 AWS CLI，您可以使用 describe-db-proxies 命令。下面的示例演示如何操作。

```
# Add --output text to get output as a simple tab-separated list.
$ aws rds describe-db-proxies --query '*[*].
{DBProxyName:DBProxyName,Endpoint:Endpoint}'
[
    [
        {
            "Endpoint": "the-proxy.proxy-demo.us-east-1.rds.amazonaws.com",
            "DBProxyName": "the-proxy"
        },
        {
            "Endpoint": "the-proxy-other-secret.proxy-demo.us-
east-1.rds.amazonaws.com",
            "DBProxyName": "the-proxy-other-secret"
        },
        {
```

```
            "Endpoint": "the-proxy-rds-secret.proxy-demo.us-
east-1.rds.amazonaws.com",
            "DBProxyName": "the-proxy-rds-secret"
        },
        {
            "Endpoint": "the-proxy-t3.proxy-demo.us-east-1.rds.amazonaws.com",
            "DBProxyName": "the-proxy-t3"
        }
    ]
]
```

2. 在客户端应用程序的连接字符串中指定该端点作为主机参数。例如，指定代理端点作为 `mysql -h` 选项或 `psql -h` 选项的值。
3. 提供您通常使用的相同数据库用户名和密码。

使用 IAM 身份验证连接到代理

在将 IAM 身份验证与RDS Proxy结合使用时，请将数据库用户设置为使用常规用户名和密码进行身份验证。IAM 身份验证适用于从 Secrets Manager 中检索用户名和密码凭证的RDS Proxy。从RDS 代理至底层数据库的连接不经由 IAM。

要使用 IAM 身份验证连接到 RDS 代理，请使用与对 Aurora 数据库集群使用 IAM 身份验证相同的常规连接过程。有关使用 IAM 的一般信息，请参阅Amazon Aurora 中的安全性。

RDS Proxy 在 IAM 用法方面的主要区别包括：

- 您不会使用授权插件配置每个单独的数据库用户。数据库用户在数据库中仍有常规的用户名和密码。您可以设置包含这些用户名和密码的 Secrets Manager 密钥，并授权 RDS Proxy 从 Secrets Manager 中检索凭证。

  IAM 身份验证应用于您的客户端程序与代理之间的连接。然后，代理使用从 Secrets Manager 中检索的用户名和密码凭证对数据库进行身份验证。
- 指定代理端点，而不是实例、集群或读取器端点。有关代理端点的详细信息，请参阅 使用 IAM 身份验证连接到数据库集群。
- 在直接数据库 IAM 身份验证情况下，您可以有选择地选取数据库用户并将其配置为使用特殊身份验证插件进行标识。然后，您便可以使用 IAM 身份验证连接到这些用户。

  在代理使用案例中，您需要为代理提供包含某些用户的用户名和密码（本机身份验证）的密钥。然后，您便可以使用 IAM 身份验证连接到代理。在这里，您可以通过使用代理端点而非数据库端点生成身份验证令牌来实现此目的。您还可以使用与您所提供密钥的用户名之一匹配的用户名。

- 当使用 IAM 身份验证连接到代理时，确保您使用传输层安全性 (TLS)/安全套接字层 (SSL)。

您可以通过修改 IAM 策略授予特定用户对代理的访问权限。下面是一个示例。

```
"Resource": "arn:aws:rds-db:us-east-2:1234567890:dbuser:prx-ABCDEFGHIJKL01234/db_user"
```

使用 PostgreSQL 连接到代理的注意事项

如果您创建新的 PostgreSQL 数据库用户来连接到 RDS 代理，请确保授予该用户对数据库的 CONNECT 权限。否则，用户将无法建立连接。有关更多信息，请参阅 the section called “使用 RDS 代理时向 PostgreSQL 数据库添加新数据库用户”。

当客户端启动到 PostgreSQL 数据库的连接时，它会发送一条启动消息。此消息包含参数名称/值字符串对。有关详细信息，请参阅 PostgreSQL 文档中的 PostgreSQL 消息格式中的 StartupMessage。

当您通过 RDS 代理进行连接时，启动消息可以包含以下当前识别的参数：

- user
- database

启动消息还可以包含以下其他运行时参数：

- application_name
- client_encoding
- DateStyle
- TimeZone
- extra_float_digits
- search_path

有关 PostgreSQL 消息收发的更多信息，请参阅 PostgreSQL 文档中的前端/后端协议。

对于 PostgreSQL，如果您使用 JDBC，我们建议您执行以下操作以避免固定：

- 将 JDBC 连接参数 assumeMinServerVersion 至少设置为 9.0 以避免固定。这可阻止 JDBC 驱动程序在运行 SET extra_float_digits = 3 时，在连接启动期间执行额外的往返行程。
- 将 JDBC 连接参数 ApplicationName 设置为 *any/your-application-name* 以避免固定。这样做会阻止 JDBC 驱动程序在运行 SET application_name = "PostgreSQL JDBC

Driver" 时，在连接启动期间执行额外的往返行程。请注意，JDBC 参数为 ApplicationName，但 PostgreSQL StartupMessage 参数为 application_name。

有关更多信息，请参阅 避免固定 RDS 代理。有关使用 JDBC 进行连接的更多信息，请参阅 PostgreSQL 文档中的连接到数据库。

管理 RDS 代理

本节提供有关如何管理 RDS 代理操作和配置的信息。这些过程可帮助您的应用程序最有效地利用数据库连接，并实现最大程度的连接重用。您越多地利用连接重用，就可以节省越多的 CPU 和内存开销。这进而减少了应用程序的延迟，使数据库能够将更多资源用于处理应用程序请求。

主题

- 修改 RDS 代理
- 使用 RDS 代理时添加新数据库用户
- RDS 代理连接注意事项
- 避免固定 RDS 代理
- 删除 RDS 代理

修改 RDS 代理

您可以在创建代理后更改与代理关联的特定设置。可通过修改代理本身和/或其关联的目标组来执行此操作。每个代理都有一个关联的目标组。

AWS Management Console

> **Important**
>
> Client authentication type（客户端身份验证类型）和 IAM authentication（IAM 身份验证）字段中的值适用于与此代理关联的所有 Secrets Manager 密钥。要为每个密钥指定不同的值，请使用 AWS CLI 或改用 API 来修改代理。

修改代理的设置

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。

2. 在导航窗格中，选择 Proxies (代理)。
3. 在代理列表中，选择要修改其设置的代理或转到其详细信息页面。
4. 对于 Actions (操作)，选择 Modify (修改)。
5. 输入或选择要修改的属性。您可以指定如下内容：
   - 代理标识符 – 通过输入新标识符来重命名代理。
   - 空闲客户端连接超时 - 输入空闲客户端连接超时的时间段。
   - IAM 角色 – 更改用于从 Secrets Manager 中检索密钥的 IAM 角色。
   - Secrets Manager 密钥 - 添加或删除 SSecrets Manager 密钥。这些密钥对应于数据库用户名和密码。
   - Client authentication type（客户端身份验证类型）-（仅限 PostgreSQL）更改客户端与代理的连接的身份验证类型。
   - IAM authentication（IAM 身份验证）– 要求或禁止对与代理的连接进行 IAM 身份验证。
   - 需要传输层安全性 - 打开或关闭传输层安全性 (TLS) 的要求。
   - VPC 安全组 - 添加或删除供代理使用的 VPC 安全组。
   - 启用增强型日志记录 - 启用或禁用增强型日志记录。
6. 选择修改。

如果您找不到要更改的列出设置，请使用以下过程更新代理的目标组。与代理关联的目标组控制与物理数据库连接相关的设置。每个代理都有一个名为 `default` 的关联目标组，该目标组与代理一起自动创建。您无法重命名默认目标组。

您只能从代理详细信息页面修改目标组，而不能从 Proxies (代理) 页面上的列表中进行修改。

修改代理目标组的设置

1. 在 Proxies (代理) 页面上，转到代理的详细信息页面。
2. 对于 Target groups (目标组)，选择 `default` 链接。目前，所有代理都有一个名为 `default` 的目标组。
3. 在默认目标组的详细信息页面上，选择 Modify (修改)。
4. 为您可以修改的属性选择新设置：
   - 数据库 – 选择不同的 Aurora 集群。
   - 连接池最大连接数 – 调整代理可使用的最大可用连接的百分比。

- 会话固定筛选条件 - (可选) 选择会话固定筛选条件。这绕过了跨客户端多路复用数据库连接的默认安全措施。目前，PostgreSQL 不支持该设置。唯一的选择是 EXCLUDE_VARIABLE_SETS。

  启用此设置可能会导致一个连接的会话变量影响其它连接。如果您的查询依赖于在当前事务之外设置的会话变量值，则可能会导致错误或正确性问题。在确认您的应用程序可以安全地跨客户端连接共享数据库连接之后，请考虑使用此选项。

  可以认为以下模式是安全的：

  - SET 语句，其中有效会话变量值没有更改，即没有更改会话变量。
  - 您可以更改会话变量值并在同一个事务中执行一条语句。

  有关更多信息，请参阅 避免固定 RDS 代理。
- 连接借用超时 - 调整连接借用超时间隔。当最大连接数已用于代理时，此设置适用。该设置确定在返回超时错误之前代理等待连接可用的时间。
- 初始化查询。(可选) 添加初始化查询或修改当前查询。您可以为代理指定一个或多个 SQL 语句，以便在打开每个新数据库连接时运行。此设置通常与 SET 语句一起使用，以确保每个连接具有完全相同的设置。确保您添加的查询有效。要在单个 SET 语句中包含多个变量，请使用逗号分隔符。例如：

  ```
  SET variable1=value1, variable2=value2
  ```

  对于多个语句，请使用分号作为分隔符。

不能更改某些属性，例如，目标组标识符和数据库引擎。

5. 选择 Modify target group (修改目标组)。

AWS CLI

要使用 AWS CLI 修改代理，请使用 modify-db-proxy、modify-db-proxy-target-group、deregister-db-proxy-targets 和 register-db-proxy-targets 命令。

使用 modify-db-proxy 命令，您可以更改以下属性：

- 代理使用的一组 Secrets Manager 密钥。
- 是否需要 TLS。

- 空闲客户端超时。
- 是否记录 SQL 语句中的其他信息以进行调试。
- 用于检索 Secrets Manager 密钥的 IAM 角色。
- 代理使用的安全组。

以下示例演示了如何重命名现有代理。

```
aws rds modify-db-proxy --db-proxy-name the-proxy --new-db-proxy-name the_new_name
```

要修改与连接相关的设置或重命名目标组，请使用 modify-db-proxy-target-group 命令。目前，所有代理都有一个名为 default 的目标组。在您使用此目标组时，请指定代理的名称，并指定 default 作为目标组的名称。您无法重命名默认目标组。

以下示例演示了如何首先检查代理的 MaxIdleConnectionsPercent 设置，然后使用目标组对其进行更改。

```
aws rds describe-db-proxy-target-groups --db-proxy-name the-proxy

{
    "TargetGroups": [
        {
            "Status": "available",
            "UpdatedDate": "2019-11-30T16:49:30.342Z",
            "ConnectionPoolConfig": {
                "MaxIdleConnectionsPercent": 50,
                "ConnectionBorrowTimeout": 120,
                "MaxConnectionsPercent": 100,
                "SessionPinningFilters": []
            },
            "TargetGroupName": "default",
            "CreatedDate": "2019-11-30T16:49:27.940Z",
            "DBProxyName": "the-proxy",
            "IsDefault": true
        }
    ]
}

aws rds modify-db-proxy-target-group --db-proxy-name the-proxy --target-group-name
 default --connection-pool-config '
{ "MaxIdleConnectionsPercent": 75 }'
```

```
{
    "DBProxyTargetGroup": {
        "Status": "available",
        "UpdatedDate": "2019-12-02T04:09:50.420Z",
        "ConnectionPoolConfig": {
            "MaxIdleConnectionsPercent": 75,
            "ConnectionBorrowTimeout": 120,
            "MaxConnectionsPercent": 100,
            "SessionPinningFilters": []
        },
        "TargetGroupName": "default",
        "CreatedDate": "2019-11-30T16:49:27.940Z",
        "DBProxyName": "the-proxy",
        "IsDefault": true
    }
}
```

使用 `deregister-db-proxy-targets` 和 `register-db-proxy-targets` 命令，您可以通过相应的目标组来更改代理关联的 Aurora 数据库集群。目前，每个代理都可以连接到一个 Aurora 数据库集群。目标组可跟踪 Aurora 集群中所有数据库实例的连接详细信息。

以下示例首先介绍了与名为 `cluster-56-2020-02-25-1399` 的 Aurora MySQL 集群关联的代理。该示例演示了如何更改代理，以便连接到名为 `provisioned-cluster` 的不同集群。

使用 Aurora 数据库集群时，指定 `--db-cluster-identifier` 选项。

以下示例修改 Aurora MySQL 代理。Aurora PostgreSQL 代理具有端口 5432。

```
aws rds describe-db-proxy-targets --db-proxy-name the-proxy

{
    "Targets": [
        {
            "Endpoint": "instance-9814.demo.us-east-1.rds.amazonaws.com",
            "Type": "RDS_INSTANCE",
            "Port": 3306,
            "RdsResourceId": "instance-9814"
        },
        {
            "Endpoint": "instance-8898.demo.us-east-1.rds.amazonaws.com",
            "Type": "RDS_INSTANCE",
            "Port": 3306,
            "RdsResourceId": "instance-8898"
```

```
        },
        {
            "Endpoint": "instance-1018.demo.us-east-1.rds.amazonaws.com",
            "Type": "RDS_INSTANCE",
            "Port": 3306,
            "RdsResourceId": "instance-1018"
        },
        {
            "Type": "TRACKED_CLUSTER",
            "Port": 0,
            "RdsResourceId": "cluster-56-2020-02-25-1399"
        },
        {
            "Endpoint": "instance-4330.demo.us-east-1.rds.amazonaws.com",
            "Type": "RDS_INSTANCE",
            "Port": 3306,
            "RdsResourceId": "instance-4330"
        }
    ]
}

aws rds deregister-db-proxy-targets --db-proxy-name the-proxy --db-cluster-identifier
 cluster-56-2020-02-25-1399

aws rds describe-db-proxy-targets --db-proxy-name the-proxy

{
    "Targets": []
}

aws rds register-db-proxy-targets --db-proxy-name the-proxy --db-cluster-identifier
 provisioned-cluster

{
    "DBProxyTargets": [
        {
            "Type": "TRACKED_CLUSTER",
            "Port": 0,
            "RdsResourceId": "provisioned-cluster"
        },
        {
            "Endpoint": "gkldje.demo.us-east-1.rds.amazonaws.com",
            "Type": "RDS_INSTANCE",
            "Port": 3306,
```

```
            "RdsResourceId": "gkldje"
        },
        {
            "Endpoint": "provisioned-1.demo.us-east-1.rds.amazonaws.com",
            "Type": "RDS_INSTANCE",
            "Port": 3306,
            "RdsResourceId": "provisioned-1"
        }
    ]
}
```

RDS API

要使用 RDS API 修改代理，请使用 ModifyDBProxy、ModifyDBProxyTargetGroup、DeregisterDBProxyTargets 和 RegisterDBProxyTargets 操作。

使用 ModifyDBProxy，您可以更改以下属性：

- 代理使用的一组 Secrets Manager 密钥。
- 是否需要 TLS。
- 空闲客户端超时。
- 是否记录 SQL 语句中的其他信息以进行调试。
- 用于检索 Secrets Manager 密钥的 IAM 角色。
- 代理使用的安全组。

使用 ModifyDBProxyTargetGroup，您可以修改与连接相关的设置。目前，所有代理都有一个名为 default 的目标组。在您使用此目标组时，请指定代理的名称，并指定 default 作为目标组的名称。您无法重命名默认目标组。

使用 DeregisterDBProxyTargets 和 RegisterDBProxyTargets，您可以通过相应的目标组来更改代理关联的 Aurora 集群。目前，每个代理都可以连接到一个 Aurora 集群。目标组可跟踪Aurora 集群中数据库实例的连接详细信息。

使用 RDS 代理时添加新数据库用户

在某些情况下，您可能会将新数据库用户添加到与代理关联的 Aurora 集群。如果是这样，请添加 Secrets Manager 密钥或调整其用途，以存储该用户的凭证。为此，请运行以下步骤：

1. 使用 在 AWS Secrets Manager 中设置 RDS 代理的数据库凭证 中描述的过程创建新 Secrets Manager 密钥。
2. 更新 IAM 角色以授予 RDS Proxy 对新 Secrets Manager 密钥的访问权限。为此，请更新 IAM 角色策略的资源部分。
3. 修改 RDS 代理，以在 Secrets Manager 密钥下添加新的 Secrets Manager 密钥。
4. 如果新用户取代了现有用户，请更新存储在代理的 Secrets Manager 密钥中的现有用户的凭证。

使用 RDS 代理时向 PostgreSQL 数据库添加新数据库用户

将新用户添加到 PostgreSQL 数据库时，如果有，请运行以下命令：

```
REVOKE CONNECT ON DATABASE postgres FROM PUBLIC;
```

向 `rdsproxyadmin` 用户授予 `CONNECT` 权限，以便用户可以监控目标数据库上的连接。

```
GRANT CONNECT ON DATABASE postgres TO rdsproxyadmin;
```

还可以通过将上述命令中的 `rdsproxyadmin` 更改为其他目标数据库用户，来允许该数据库用户执行运行状况检查。

使用 RDS 代理时更改数据库用户的密码

在某些情况下，您可能会更改与代理关联的 Aurora 集群中的数据库用户的密码。如果是这样，请使用新密码更新相应的 Secrets Manager 密钥。

RDS 代理连接注意事项

配置连接设置

若要调整 RDS Proxy 的连接池，您可以修改以下设置：

- IdleClientTimeout
- MaxConnectionsPercent
- MaxIdleConnectionsPercent
- ConnectionBorrowTimeout

IdleClientTimeout

您可以指定在代理关闭客户端连接之前，客户端连接可以处于空闲状态的时长。默认值为 1800 秒（30 分钟）。

如果应用程序未在上一请求完成后的指定时间内提交新请求，则将客户端连接视为空闲。底层数据库连接保持打开状态并返回到连接池。因此，它可以重复用于新的客户端连接。如果希望代理主动删除过时的连接，则降低空闲客户端连接超时。如果工作负载与代理建立频繁的连接，则增加空闲客户端连接超时，以便节省建立连接的成本。

此设置由 RDS 控制台中的 Idle client connection timeout（空闲客户端连接超时）字段以及 AWS CLI 和 API 中的 `IdleClientTimeout` 设置表示。要了解如何在 RDS 控制台中更改 Idle client connection timeout（空闲客户端连接超时）字段的值，请参阅 AWS Management Console。要了解如何更改 `IdleClientTimeout` 设置的值，请参阅 CLI 命令 modify-db-proxy 或 API 操作 ModifyDBProxy。

MaxConnectionsPercent

您可以限制 RDS 代理可与目标数据库建立的连接数。请将限制指定为数据库可用的最大连接数的百分比。此设置由 RDS 控制台中的 Connection pool maximum connections（连接池最大连接数）字段以及 AWS CLI 和 API 中的 `MaxConnectionsPercent` 设置表示。

对于目标组使用的 Aurora 数据库集群，`MaxConnectionsPercent` 值以 `max_connections` 设置的百分比表示。代理不会提前创建所有连接。此设置允许代理在工作负载需要时建立这些连接。

例如，对于 `max_connections` 设置为 1000 且 `MaxConnectionsPercent` 设置为 95 的注册数据库目标，RDS 代理将 950 个连接设置为与该数据库目标的并行连接的上限。

工作负载达到允许的最大数据库连接数的一个常见副作用是总体查询延迟增加，同时 `DatabaseConnectionsBorrowLatency` 指标也会增加。您可以通过比较 `DatabaseConnections` 和 `MaxDatabaseConnectionsAllowed` 指标来监控当前使用的数据库连接数和允许的数据库连接总数。

设置此参数时，请注意以下最佳实践：

- 为工作负载模式的变化留出足够的连接余量。建议将该参数设置为比最近监控的最大使用量高出至少 30%。由于 RDS 代理在多个节点之间重新分配数据库连接限额，因此，内部容量更改可能需要至少 30% 的余量来增加连接，以避免增加借用延迟。
- RDS 代理保留一定数量的连接用于主动监控，以支持快速失效转移、流量路由和内部操作。`MaxDatabaseConnectionsAllowed` 指标不包括这些预留连接。该指标表示可用于为工作负载提供服务的连接数，可以低于从 `MaxConnectionsPercent` 设置得出的值。

建议的最小 MaxConnectionsPercent 值如下：

- db.t3.small：100
- db.t3.medium：55
- db.t3.large：35
- db.r3.large 或更大：20

如果在 RDS 代理中注册了多个目标实例，例如带有读取器节点的 Aurora 集群，请根据最小的注册实例设置最小值。

要了解如何在 RDS 控制台中更改 Connection pool maximum connections（连接池最大连接数）字段的值，请参阅 AWS Management Console。要了解如何更改 MaxConnectionsPercent 设置的值，请参阅 CLI 命令 modify-db-proxy-target-group 或 API 操作 ModifyDBProxyTargetGroup。

> **Important**
>
> 如果数据库集群是开启写入转发的全局数据库的一部分，则按照分配给写入转发的限额来减小代理的 MaxConnectionsPercent 值。写入转发限额在数据库集群参数 aurora_fwd_writer_max_connections_pct 中设置。有关写入转发的信息，请参阅 在 Amazon Aurora Global Database 中使用写入转发。

有关数据库连接限制的更多信息，请参阅 Aurora MySQL 数据库实例的最大连接数和 Aurora PostgreSQL 数据库实例的最大连接数。

MaxIdleConnectionsPercent

您可以控制 RDS Proxy 可在连接池中保留的空闲数据库连接数。默认情况下，如果连接五分钟内没有活动，RDS 代理会将其池中的数据库连接视为空闲。

对于 RDS 数据库实例目标组，MaxIdleConnectionsPercent 值以 max_connections 设置的百分比表示。原定设置值为 MaxConnectionsPercent 的 50%，上限为 MaxConnectionsPercent 的值。例如，如果 MaxConnectionsPercent 为 80，则 MaxIdleConnectionsPercent 的默认值为 40。

对于高值，代理会将较高百分比的空闲数据库连接保持在打开状态。对于低值，代理会关闭较高百分比的空闲数据库连接。如果您的工作负载不可预测，请考虑为 MaxIdleConnectionsPercent 设置较高的值。这样做意味着 RDS 代理可以在不打开大量新数据库连接的情况下适应活动的突增。

此设置由 AWS CLI 和 API 中 DBProxyTargetGroup 的 MaxIdleConnectionsPercent 设置表示。要了解如何更改 MaxIdleConnectionsPercent 设置的值，请参阅 CLI 命令 modify-db-proxy-target-group 或 API 操作 ModifyDBProxyTargetGroup。

有关数据库连接限制的更多信息，请参阅 Aurora MySQL 数据库实例的最大连接数和 Aurora PostgreSQL 数据库实例的最大连接数。

ConnectionBorrowTimeout

您可以选择 RDS Proxy 在返回超时错误之前等待连接池中数据库连接可用的时间。默认值为 120 秒。此设置适用于连接数达到最大而导致连接池中没有可用连接的情况。当没有适当的数据库实例可用于处理请求时，例如当失效转移操作正在进行时，它也适用。通过使用此设置，您可以为应用程序设置最佳等待期，而无需更改应用程序代码中的查询超时。

此设置由 RDS Proxy 控制台中的 Connection borrow timeout（连接借用超时）字段、ConnectionBorrowTimeout 中 DBProxyTargetGroup 的 AWS CLI 设置或 API 表示。要了解如何在 RDS 控制台中更改 Connection borrow timeout（连接借用超时）字段的值，请参阅 AWS Management Console。要了解如何更改 ConnectionBorrowTimeout 设置的值，请参阅 CLI 命令 modify-db-proxy-target-group 或 API 操作 ModifyDBProxyTargetGroup。

客户端和数据库连接

从您的应用程序到 RDS 代理的连接称为客户端连接。从代理到数据库的连接则称为数据库连接。使用 RDS 代理时，客户端连接在代理处终止，而数据库连接则在 RDS 代理中进行管理。

应用程序端连接池可以减少在应用程序与 RDS 代理之间重复建立连接。

在实现应用程序端连接池之前，请考虑以下配置方面：

- 客户端连接最长期限：RDS 代理强制规定客户端连接的最长期限为 24 小时。此值不可配置。将池配置为最长连接期限小于 24 小时，以免客户端连接意外中断。
- 客户端连接空闲超时：RDS 代理强制规定客户端连接的最长空闲时间。为池配置的空闲连接超时值要小于 RDS 代理的客户端连接空闲超时设置，以免连接意外中断。

在应用程序端连接池中配置的最大客户端连接数不必限制为 RDS 代理的 max_connections 设置。

客户端连接池可延长客户端连接期限。如果您的连接遇到固定问题，则池化客户端连接可能会降低多路复用效率。已固定但在应用程序端连接池中处于空闲状态的客户端连接会继续保持数据库连接，并防止数据库连接被其它客户端连接重复使用。查看您的代理日志，以检查您的连接是否遇到固定问题。

> **Note**
>
> 当数据库连接不再使用时，RDS 代理会在 24 小时后的某个时间关闭数据库连接。无论最大空闲连接设置的值如何，代理都会执行此操作。

避免固定 RDS 代理

当数据库请求不依赖于先前请求的状态信息时，多路复用效率更高。在这种情况下，RDS Proxy 可以在每个事务结束时重用连接。此类状态信息的示例包括可通过 SET 或 SELECT 语句更改的大多数变量和配置参数。默认情况下，客户端连接上的 SQL 事务可以在底层数据库连接之间多路复用。

与代理的连接可以进入一种称为固定的状态。固定连接后，每个后续事务将使用相同的底层数据库连接，直到会话结束。在会话结束之前，其他客户端连接也不能重用该数据库连接。客户端连接断开时，会话结束。

当 RDS Proxy 检测到不适合其他会话的会话状态更改时，它会自动将客户端连接固定到特定的数据库连接。固定降低了连接重用的有效性。如果您的所有连接或几乎所有连接都遇到固定，请考虑修改应用程序代码或工作负载，以减少导致固定的条件。

例如，您的应用程序更改了会话变量或配置参数。在这种情况下，后面的语句可能依赖于新变量或参数来生效。因此，当 RDS 代理处理更改会话变量或配置设置的请求时，它会将该会话固定到数据库连接。这样，会话状态对于同一会话中的所有后续事务仍然有效。

对于某些数据库引擎，此规则并不适用于您可以设置的全部参数。RDS 代理会跟踪某些语句和变量。因此，在您修改这些内容时，RDS 代理不会固定会话。在这种情况下，RDS 代理仅将连接重用于具有相同设置值的其他会话。有关 Aurora MySQL 的跟踪语句和变量的列表，请参阅 RDS 代理针对 Aurora MySQL 数据库跟踪的内容。

RDS 代理针对 Aurora MySQL 数据库跟踪的内容

以下是 RDS 代理跟踪的 MySQL 语句：

- DROP DATABASE
- DROP SCHEMA
- USE

以下是 RDS 代理跟踪的 MySQL 变量：

- AUTOCOMMIT
- AUTO_INCREMENT_INCREMENT
- CHARACTER SET (or CHAR SET)
- CHARACTER_SET_CLIENT
- CHARACTER_SET_DATABASE
- CHARACTER_SET_FILESYSTEM
- CHARACTER_SET_CONNECTION
- CHARACTER_SET_RESULTS
- CHARACTER_SET_SERVER
- COLLATION_CONNECTION
- COLLATION_DATABASE
- COLLATION_SERVER
- INTERACTIVE_TIMEOUT
- NAMES
- NET_WRITE_TIMEOUT
- QUERY_CACHE_TYPE
- SESSION_TRACK_SCHEMA
- SQL_MODE
- TIME_ZONE
- TRANSACTION_ISOLATION (or TX_ISOLATION)
- TRANSACTION_READ_ONLY (or TX_READ_ONLY)
- WAIT_TIMEOUT

最大限度地减少固定

RDS 代理的性能优化包括尝试通过最小化固定来最大化事务级别连接重用（多路复用）。

您可以执行以下步骤以最大限度地减少固定：

- 避免可能导致固定的不必要的数据库请求。
- 在所有连接中一致地设置变量和配置设置。这样，后续会话更有可能重用具有这些特定设置的连接。

  但是，对于 PostgreSQL 设置，变量会导致会话固定。

- 对于 MySQL 引擎系列数据库，可将会话固定筛选条件应用于代理。您可以免除某些类型的操作，使其不固定会话（如果您知道这样做不会影响应用程序的正确操作）。
- 通过监控 Amazon CloudWatch 指标 `DatabaseConnectionsCurrentlySessionPinned` 来查看固定的发生频率。有关该指标和其他 CloudWatch 指标的信息，请参阅 使用 Amazon CloudWatch 监控 RDS Proxy 指标。
- 如果您使用 `SET` 语句为每个客户端连接执行相同的初始化，则可以在保留事务级别多路复用的同时执行此操作。在这种情况下，您将设置初始会话状态的语句移动到由代理使用的初始化查询中。该属性是一个字符串，包含一个或多个 SQL 语句（用分号分隔）。

  例如，您可以为设置特定配置参数的代理定义初始化查询。然后，每当为该代理设置新连接时，RDS Proxy 都会应用这些设置。您可以从应用程序代码中删除相应的 `SET` 语句，这样，它们就不会干扰事务级别多路复用。

  有关代理的固定发生频率的指标，请参阅 使用 Amazon CloudWatch 监控 RDS Proxy 指标。

导致对所有引擎系列进行固定的条件

对于以下情况（其中多路复用可能会导致意外行为），代理将会话固定到当前连接：

- 文本大小大于 16 KB 的任何语句都会导致代理固定会话。

导致对 Aurora MySQL 进行固定的条件

对于 MySQL，以下交互也会导致固定：

- 显式表锁定语句 `LOCK TABLE`、`LOCK TABLES` 或 `FLUSH TABLES WITH READ LOCK` 会导致代理固定会话。
- 通过使用 `GET_LOCK` 创建命名锁会导致代理固定会话。
- 设置用户变量或系统变量（有些例外）会导致代理固定会话。如果这种情况过多地减少了连接重用，则选择让 `SET` 操作不导致固定。有关如何通过设置会话固定筛选条件属性来执行此操作的信息，请参阅为 Amazon Aurora 创建代理和修改 RDS 代理。
- 创建临时表会导致代理固定会话。这样，无论事务边界如何，临时表的内容都会在整个会话期间保留。
- 调用函数 `ROW_COUNT`、`FOUND_ROWS` 和 `LAST_INSERT_ID` 有时会导致固定。

  这些函数导致固定的确切情况在与 MySQL 5.7 兼容的 Aurora MySQL 版本间可能有所不同。

- 预编译语句会导致代理固定会话。无论预编译语句使用 SQL 文本还是二进制协议，这项规则都适用。
- 使用 SET LOCAL 时，RDS 代理无法固定连接。
- 调用存储过程和存储函数不会导致固定。RDS 代理不会检测此类调用导致的任何会话状态更改。如果您希望跨事务持久保持存储例程中的会话状态，请确保您的应用程序不会更改该状态。例如，RDS 代理目前与创建跨所有事务持久存在的临时表的存储过程不兼容。

如果您拥有关于应用程序行为的专业知识，则可以跳过某些应用程序语句的固定行为。为此，请在创建代理时选择会话固定筛选条件选项。当前，您可以选择退出会话固定，以设置会话变量和配置设置。

导致对 Aurora PostgreSQL 进行固定的条件

对于 PostgreSQL，以下交互也会导致固定：

- 使用 SET 命令。
- 使用 PREPARE、DISCARD、DEALLOCATE、或 EXECUTE 命令管理准备好的语句。
- 创建临时序列、表或视图。
- 声明游标。
- 丢弃会话状态。
- 监听通知频道。
- 加载库模块，如 auto_explain。
- 使用函数操作序列，例如 nextval 和 setval。
- 使用函数与锁定交互，例如 pg_advisory_lock 和 pg_try_advisory_lock。

> **Note**
>
> RDS 代理未锁定事务级咨询锁，特别是 pg_advisory_xact_lock、pg_advisory_xact_lock_shared、pg_try_advisory_xact_l 和 pg_try_advisory_xact_lock_shared。

- 设置参数或将参数重置为其默认值。具体而言，就是使用 SET 和 set_config 命令为会话变量分配默认值。
- 调用存储过程和存储函数不会导致固定。RDS 代理不会检测此类调用导致的任何会话状态更改。如果您希望跨事务持久保持存储例程中的会话状态，请确保您的应用程序不会更改该状态。例如，RDS 代理目前与创建跨所有事务持久存在的临时表的存储过程不兼容。

删除 RDS 代理

如果您不再需要某个代理，可以将其删除。或者，如果您停止使用数据库实例或与其关联的集群，则可以删除代理。

AWS Management Console

删除代理

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Proxies (代理)。
3. 从列表中选择要删除的代理。
4. 选择 Delete Proxy (删除代理)。

AWS CLI

要删除数据库代理，请使用 AWS CLI 命令 delete-db-proxy。要删除相关的关联，还可以使用 deregister-db-proxy-targets 命令。

```
aws rds delete-db-proxy --name proxy_name
```

```
aws rds deregister-db-proxy-targets
    --db-proxy-name proxy_name
    [--target-group-name target_group_name]
    [--target-ids comma_separated_list]       # or
    [--db-instance-identifiers instance_id]       # or
    [--db-cluster-identifiers cluster_id]
```

RDS API

要删除数据库代理，请调用 Amazon RDS API 函数 DeleteDBProxy。要删除相关的项目和关联，您还可以调用函数 DeleteDBProxyTargetGroup 和 DeregisterDBProxyTargets。

使用 Amazon RDS Proxy 终端节点

RDS 代理端点提供了灵活且高效地管理数据库连接的方式，从而提高了可扩展性、可用性和安全性。使用代理端点，您可以：

- 简化监控和故障排除：使用多个端点来独立跟踪和管理来自不同应用程序的连接。
- 增强读取可扩展性：对于 Aurora 数据库集群利用读取器端点来高效地分配读取流量，从而优化查询密集型工作负载的性能。
- 启用跨 VPC 访问：使用跨 VPC 端点来跨 VPC 连接数据库，这可让一个 VPC 中的 Amazon EC2 实例等资源访问另一个 VPC 中的数据库。

主题

- 代理终端节点概述
- 代理端点的限制
- 将读取器终端节点与 Aurora 集群结合使用
- 访问 VPC 中的 Aurora 数据库
- 创建代理终端节点
- 查看代理终端节点
- 修改代理终端节点
- 删除代理终端节点

代理终端节点概述

使用 RDS 代理终端节点所涉及的过程与使用 Aurora 数据库集群和读取器终端节点相同。如果您不熟悉 Aurora 终端节点，请参阅 Amazon Aurora 端点连接。

当您将 RDS 代理与 Aurora 集群结合使用时，默认端点同时支持读取和写入操作。这意味着它将所有请求路由到集群的写入器实例，从而影响其 `max_connections` 限制。为了更好地分配流量，您可以为代理创建额外的读/写或只读端点，这样可以更高效地管理工作负载并提高可扩展性。

将只读端点与代理结合使用来处理读取查询，就像您将读取器端点用于 Aurora 预置集群一样。这种方法通过将查询分布在一个或多个读取器数据库实例上，可最大限度地提高 Aurora 的读取可扩展性。通过使用只读端点并根据需要添加更多读取器实例，您可以增加集群可以处理的同时查询和连接的数量。

> **Tip**
>
> 使用 AWS Management Console为 Aurora 集群创建代理时，可以让 RDS 代理自动创建读取器端点。有关读取器终端节点优点的信息，请参阅 将读取器终端节点与 Aurora 集群结合使用。

当您创建代理端点时，您可以将它与不同于代理虚拟私有云（VPC）的 VPC 相关联。这样，您就可以从其它 VPC 连接到代理，例如组织内其它应用程序使用的 VPC。

有关与代理终端节点关联的限制的信息，请参阅 代理端点的限制。

RDS 代理日志在每个条目前添加关联代理端点的名称作为前缀。这可以是您为用户定义的端点指定的名称，也可以是代理默认读/写端点的特殊名称 `default`。

每个代理终端节点都有自己的 CloudWatch 指标集合。监控所有代理端点、特定端点的指标，或者代理的所有读/写或只读端点的指标。有关更多信息，请参阅 使用 Amazon CloudWatch 监控 RDS Proxy 指标。

代理终端节点使用与其关联的代理相同的身份验证机制。RDS Proxy 会自动为用户定义的终端节点设置权限和授权，并与关联代理的属性保持一致。

要了解代理端点如何用于 Aurora Global Database 中的数据库集群，请参阅 RDS 代理端点如何与全局数据库配合使用。

代理端点的限制

RDS 代理端点具有以下限制：

- 无法修改 RDS 代理默认端点。
- 代理的用户定义终端节点的最大数量为 20 个。因此，代理最多可以有 21 个终端节点：默认终端节点加上您创建的 20 个终端节点。
- 当您将其他终端节点与代理关联时，RDS Proxy 会自动确定集群中的哪些数据库实例用于每个终端节点。您无法像使用 Aurora 自定义终端节点那样选择特定实例。

创建代理时，RDS 会自动创建 VPC 端点，以用于应用程序和数据库之间的安全通信。VPC 端点是可见的，可从 Amazon VPC 控制台访问该端点。

添加新的代理端点会预置一个 AWS PrivateLink 接口端点。如果您向代理添加一个或多个端点，则需支付额外费用。有关更多信息，请参阅 RDS 代理定价。

将读取器终端节点与 Aurora 集群结合使用

在将 RDS Proxy 与 Aurora 集群结合使用时，可以创建并连接到称为读取器终端节点的只读终端节点。这些读取器终端节点有助于提高查询密集型应用程序的读取可扩展性。如果集群中的读取器数据库实例变得不可用，读取器终端节点还有助于提高连接的可用性。

> **Note**
>
> 当您将新的终端节点指定为只读时，RDS Proxy 要求 Aurora 集群有一个或多个读取器数据库实例。在某些情况下，可以将代理的目标更改为只包含单个写入器的 Aurora 集群。如果您这样做，则对读取器端点的任何请求都会失败并显示错误。如果代理的目标是 RDS 实例而不是 Aurora 集群，则请求也会失败。
>
> 如果 Aurora 集群具有读取器实例，但这些实例不可用，则 RDS Proxy 会等待发送请求，而不是立即返回错误。如果在连接借用超时期间没有读取器实例变为可用，则请求将失败并显示错误。

读取器终端节点如何帮助提高应用程序可用性

在某些情况下，集群中的一个或多个读取器实例可能会变得不可用。如果是这样，使用数据库代理的读取器终端节点的连接比使用 Aurora 读取器终端节点的连接恢复得更快。RDS Proxy 仅将连接路由到集群中的可用读取器实例。当实例变为不可用时，不会由于 DNS 缓存而造成延迟。

如果连接是多路复用的，则 RDS Proxy 会将后续查询定向到其他读取器数据库实例，而不会中断您的应用程序。在自动切换到新的读取器实例的过程中，RDS Proxy 会检查新旧读取器实例的复制滞后时间。RDS Proxy 确保新的读取器实例是最新的，并且与先前的读取器实例具有相同的更改。这样，当 RDS Proxy 从一个读取器数据库实例切换到另一个读取器数据库实例时，您的应用程序将永远不会看到过时的数据。

如果连接是固定的，则该连接中的下一个查询将返回错误。但是，您的应用程序可以立即重新连接到同一终端节点。RDS Proxy 会将连接路由到处于 `available` 状态的其他读取器数据库实例。在手动重新连接时，RDS Proxy 不会检查新旧读取器实例之间的复制滞后。

如果您的 Aurora 集群没有可用的读取器实例，则 RDS Proxy 会检查此情况是暂时的还是永久的。每种情况下的行为如下：

- 假设您的集群有一个或多个读取器数据库实例，但它们都没有处于 `Available` 状态。例如，所有读取器实例可能都正在重启或遇到问题。在这种情况下，连接到读取器终端节点的尝试将等待读取器实例变为可用。如果在连接借用超时期间没有读取器实例变为可用，则连接尝试将失败。如果读取器实例变为可用，则连接尝试成功。
- 假设您的集群没有读取器数据库实例。在这种情况下，RDS Proxy 会在您尝试连接到读取器终端节点时立即返回错误。要解决此问题，请在连接到读取器终端节点之前将一个或多个读取器实例添加到集群中。

读取器终端节点如何帮助提高查询可扩展性

代理的读取器终端节点通过以下方式帮助提高 Aurora 查询可扩展性：

- 当您将读取器实例添加到 Aurora 集群时，RDS Proxy 可以将到任何读取器终端节点的新连接路由到其他读取器实例。这样，使用一个读取器终端节点连接执行的查询不会减慢使用另一个读取器终端节点连接执行的查询。查询在单独的数据库实例上运行。每个数据库实例都有其自己的计算资源、缓冲区缓存等。
- 在可行的情况下，RDS Proxy 使用特定的读取器终端节点连接对所有查询问题使用相同的读取器数据库实例。这样，对相同表的一组相关查询就可以利用特定数据库实例上的缓存、计划优化等优势。
- 如果读取器数据库实例变为不可用，则对应用程序的影响取决于会话是多路复用的还是固定的。如果会话是多路复用的，则 RDS Proxy 会将所有后续查询路由到其他读取器数据库实例，而无需您执行任何操作。如果会话是固定的，则您的应用程序将出现错误，必须重新连接。您可以立即重新连接到读取器终端节点，RDS Proxy 会将连接路由到可用的读取器数据库实例。有关代理会话的多路复用和固定的更多信息，请参阅 RDS 代理概念概述。
- 集群中拥有的读取器数据库实例越多，使用读取器终端节点可以同时建立的连接越多。例如，假设您的集群有四个读取器数据库实例，每个实例都配置为支持 200 个同时连接。另外假设您的代理配置为使用最大连接数的 50%。在这种情况下，对于读取器 1，您可以通过代理中的读取器终端节点建立的最大连接数为100（200 的 50%）。对于读取器 2 来说，也是 100，依此类推，总共是 400。如果将集群读取器数据库实例的数量增加一倍至 8，则通过读取器终端节点的最大连接数也将增加一倍，达到 800。

使用读取器终端节点的示例

以下 Linux 示例显示了如何确认您已通过读取器终端节点连接到 Aurora MySQL 集群。`innodb_read_only` 配置设置已启用。执行写操作（例如 `CREATE DATABASE` 语句）的尝试失败并显示错误。您可以通过使用 `aurora_server_id` 变量检查数据库实例名称来确认您已连接到读取器数据库实例。

> **Tip**
>
> 不要只依靠检查数据库实例名称来确定连接是读/写还是只读。请记住，发生故障转移时，Aurora 集群中的数据库实例可以在写入器和读取器之间更改角色。

```
$ mysql -h endpoint-demo-reader.endpoint.proxy-demo.us-east-1.rds.amazonaws.com -u
 admin -p
```

```
...
mysql> select @@innodb_read_only;
+--------------------+
| @@innodb_read_only |
+--------------------+
|                  1 |
+--------------------+
mysql> create database shouldnt_work;
ERROR 1290 (HY000): The MySQL server is running with the --read-only option so it
 cannot execute this statement

mysql> select @@aurora_server_id;
+---------------------------------------+
| @@aurora_server_id                    |
+---------------------------------------+
| proxy-reader-endpoint-demo-instance-3 |
+---------------------------------------+
```

以下示例显示您与代理读取器终端节点的连接如何在删除读取器数据库实例时仍正常工作。在本示例中，Aurora 集群具有两个读取器实例，instance-5507 和 instance-7448。读取器终端节点的连接开始时使用其中一个读取器实例。在该示例中，此读取器实例被 delete-db-instance 命令删除。RDS Proxy 会切换到其他读取器实例以进行后续查询。

```
$ mysql -h reader-demo.endpoint.proxy-demo.us-east-1.rds.amazonaws.com
  -u my_user -p
...
mysql> select @@aurora_server_id;
+--------------------+
| @@aurora_server_id |
+--------------------+
| instance-5507      |
+--------------------+

mysql> select @@innodb_read_only;
+--------------------+
| @@innodb_read_only |
+--------------------+
|                  1 |
+--------------------+

mysql> select count(*) from information_schema.tables;
+----------+
| count(*) |
```

```
+----------+
|      328 |
+----------+
```

当 mysql 会话仍在运行时，以下命令将删除读取器终端节点连接到的读取器实例。

```
aws rds delete-db-instance --db-instance-identifier instance-5507 --skip-final-snapshot
```

mysql 会话中的查询会继续工作，无需重新连接。RDS Proxy 会自动切换到其他读取器数据库实例。

```
mysql> select @@aurora_server_id;
+--------------------+
| @@aurora_server_id |
+--------------------+
| instance-7448      |
+--------------------+

mysql> select count(*) from information_schema.TABLES;
+----------+
| count(*) |
+----------+
|      328 |
+----------+
```

访问 VPC 中的 Aurora 数据库

默认情况下，Aurora 技术堆栈的组件都在同一个 Amazon VPC 中。例如，假设一个在 Amazon EC2 实例上运行的应用程序连接到 Aurora 数据库集群。在这种情况下，应用程序服务器和数据库必须都位于同一 VPC 内。

通过 RDS 代理，您可以设置从一个 VPC 中的资源（例如，EC2 实例）访问另一个 VPC 中的 Aurora 数据库集群。例如，您的组织可能有多个应用程序访问同一数据库资源。每个应用程序都可能位于自己的 VPC 中。

要启用跨 VPC 访问，您需要为代理创建一个新的终端节点。代理本身与 Aurora 数据库集群位于同一个 VPC 中。但是，跨 VPC 终端节点与 EC2 实例等其他资源一起位于另一个 VPC 中。跨 VPC 终端节点与 EC2 和其他资源所在的 VPC 中的子网和安全组关联。通过这些关联，您可以从原本由于 VPC 限制而无法访问数据库的应用程序连接到终端节点。

以下步骤说明了如何通过 RDS Proxy 创建和访问跨 VPC 终端节点：

1. 创建两个 VPC，或者选择已用于 Aurora 工作的两个 VPC。每个 VPC 都应该有自己的关联网络资源，例如互联网网关、路由表、子网和安全组。如果您只有一个 VPC，可以查阅 开始使用 Amazon Aurora，了解设置另一个 VPC 以成功使用 Aurora 的步骤。您还可以查看 Amazon EC2 控制台中的现有 VPC，以了解要将哪些类型的资源连接起来。
2. 创建一个与要连接到的 Aurora 数据库集群关联的数据库代理。按照 为 Amazon Aurora 创建代理 中的过程操作。
3. 在 RDS 控制台中的代理的 Details（详细信息）页面上，在 Proxy endpoints（代理终端节点）部分下，选择 Create endpoint（创建终端节点）。按照 创建代理终端节点 中的过程操作。
4. 选择将跨 VPC 终端节点设置为读/写还是只读。
5. 不接受默认的与 Aurora 数据库集群使用相同的 VPC，而是选择其它 VPC。此 VPC 必须与代理所在的 VPC 位于同一 AWS 区域中。
6. 现在不接受来自 Aurora 数据库集群所在 VPC 中的默认子网和安全组，而是进行新的选择。根据您选择的 VPC 中的子网和安全组进行这些选择。
7. 您无需更改 Secrets Manager 密钥的任何设置。相同的凭证适用于代理的所有终端节点，无论每个终端节点位于哪个 VPC 中均是如此。
8. 等待新终端节点达到 Available（可用）状态。
9. 记下终端节点的完整名称。这是以 *Region_name*`.rds.amazonaws.com` 为结尾的值，您提供此值作为数据库应用程序连接字符串的一部分。
10. 从与该终端节点位于同一 VPC 中的资源访问新的终端节点。测试此过程的一种简单方法是在此 VPC 中创建一个新的 EC2 实例。然后，登录 EC2 实例并运行 `mysql` 或 `psql` 命令，以使用连接字符串中的端点值进行连接。

创建代理终端节点

要创建代理端点，请按照以下说明操作：

控制台

创建代理端点

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Proxies (代理)。
3. 单击您要为其创建新终端节点的代理的名称。

此时将显示该代理的详细信息页面。

4. 在 Proxy endpoints（代理终端节点）部分中，选择 Create proxy endpoint（创建代理终端节点）。

   此时将显示 Create proxy endpoint（创建代理终端节点）窗口。

5. 对于 Proxy endpoint name（代理终端节点名称），输入您选择的描述性名称。

6. 对于 Target role（目标角色），选择将终端节点设置为读/写还是只读。

   使用读/写端点的连接可以执行任何类型的操作，如数据定义语言（DDL）语句、数据操作语言（DML）语句和查询。这些终端节点始终连接到 Aurora 集群的主实例。仅在应用程序中使用单个终端节点时，可以将读/写终端节点用于常规数据库操作。您还可以将读/写终端节点用于管理操作、联机事务处理 (OLTP) 应用程序和提取-转换-加载 (ETL) 作业。

   使用只读终端节点的连接只能执行查询。当 Aurora 集群中有多个读取器实例时，RDS 代理可以为终端节点的每个连接使用不同的读取器实例。这样，查询密集型应用程序就可以利用 Aurora 的集群功能。您可以通过添加更多读取器数据库实例来向集群添加更多查询容量。这些只读连接不会在集群的主实例上增加任何开销。这样，您的报告和分析查询就不会减慢 OLTP 应用程序的写入操作。

7. 对于虚拟私有云（VPC），选择默认值，以便从您通常用于访问代理或其关联数据库的同一 EC2 实例或其它资源访问端点。要为此代理设置跨 VPC 访问，请选择原定设置值以外的 VPC。有关跨 VPC 访问的更多信息，请参阅 访问 VPC 中的 Aurora 数据库。

8. 对于子网，预设情况下，RDS Proxy 会填充与关联代理相同的子网。要限制对端点的访问，以便只有 VPC 的地址范围的一部分能够与其连接，请删除一个或多个子网。

9. 对于 VPC security group (VPC 安全组)，您可以选择现有安全组或创建新安全组。默认情况下，RDS Proxy 会填充与关联代理相同的一个或多个安全组。如果代理的入站和出站规则适用于此端点，则保留默认选项。

   如果选择创建新的安全组，请在此页面上为该安全组指定一个名称。然后，从 EC2 控制台编辑安全组设置。

10. 选择 Create proxy endpoint（创建代理终端节点）。

AWS CLI

要创建代理终端节点，请使用 AWS CLIcreate-db-proxy-endpoint 命令。

包括以下必需参数：

- `--db-proxy-name value`
- `--db-proxy-endpoint-name value`
- `--vpc-subnet-ids list_of_ids`。用空格分隔子网 ID。您无需指定 VPC 本身的 ID。

您还可以包括以下可选参数：

- `--target-role { READ_WRITE | READ_ONLY }`此参数默认为 READ_WRITE。READ_ONLY 值仅对包含一个或多个读取器数据库实例的 Aurora 预调配集群有影响。当代理与仅包含写入器数据库实例的 Aurora 集群相关联时，您不能指定 READ_ONLY。有关将只读端点与 Aurora 集群结合使用的详细信息，请参阅将读取器终端节点与 Aurora 集群结合使用 。
- `--vpc-security-group-ids value`。用空格分隔安全组 ID。如果省略此参数，则 RDS Proxy 使用 VPC 的默认安全组。RDS Proxy 根据您为 `--vpc-subnet-ids` 参数指定的子网 ID 确定 VPC。

Example

下面的示例用于创建一个名为 `my-endpoint` 的代理终端节点。

对于 Linux、macOS 或 Unix：

```
aws rds create-db-proxy-endpoint \
  --db-proxy-name my-proxy \
  --db-proxy-endpoint-name my-endpoint \
  --vpc-subnet-ids subnet_id subnet_id subnet_id ... \
  --target-role READ_ONLY \
  --vpc-security-group-ids security_group_id ]
```

对于 Windows：

```
aws rds create-db-proxy-endpoint ^
  --db-proxy-name my-proxy ^
  --db-proxy-endpoint-name my-endpoint ^
  --vpc-subnet-ids subnet_id_1 subnet_id_2 subnet_id_3 ... ^
  --target-role READ_ONLY ^
  --vpc-security-group-ids security_group_id
```

RDS API

要创建代理端点，请使用 RDS API CreateDBProxyEndpoint 操作。

查看代理终端节点

要查看现有代理端点，请按照以下说明进行操作：

控制台

查看代理终端节点的详细信息

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Proxies (代理)。
3. 在列表中，选择要查看其终端节点的代理。单击代理名称以查看其详细信息页面。
4. 在 Proxy endpoints（代理终端节点）部分中，选择要查看的终端节点。单击其名称以查看详细信息页面。
5. 检查您对其值感兴趣的参数。您可以检查以下属性：
   - 终端节点是读/写还是只读。
   - 您在数据库连接字符串中使用的终端节点地址。
   - 与终端节点关联的 VPC、子网和安全组。

AWS CLI

要查看一个或多个代理端点，请使用 AWS CLI describe-db-proxy-endpoints 命令。

您可以包括以下参数：

- `--db-proxy-endpoint-name`
- `--db-proxy-name`

以下示例描述了 `my-endpoint` 代理终端节点：

Example

对于 Linux、macOS 或 Unix：

```
aws rds describe-db-proxy-endpoints \
  --db-proxy-endpoint-name my-endpoint
```

对于 Windows：

```
aws rds describe-db-proxy-endpoints ^
  --db-proxy-endpoint-name my-endpoint
```

RDS API

要描述一个或多个代理终端节点，请使用 RDS API DescribeDBProxyEndpoints 操作。

修改代理终端节点

要修改代理端点，请按照以下说明进行操作：

控制台

修改一个或多个代理终端节点

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Proxies (代理)。
3. 在列表中，选择要修改其终端节点的代理。单击代理名称以查看其详细信息页面。
4. 在 Proxy endpoints（代理终端节点）部分中，选择要修改的终端节点。您可以在列表中选择它，也可以单击其名称以查看详细信息页面。
5. 在代理详细信息页面上的 Proxy endpoints（代理终端节点）部分中，选择 Edit（编辑）。或者，在代理端点详细信息页面上，对操作，选择编辑。
6. 更改要修改的参数的值。
7. 选择 Save changes（保存更改）。

AWS CLI

要修改代理端点，请使用 AWS CLI modify-db-proxy-endpoint 命令和以下必需参数：

- `--db-proxy-endpoint-name`

使用以下一个或多个参数指定对终端节点属性的更改：

- `--new-db-proxy-endpoint-name`
- `--vpc-security-group-ids`。用空格分隔安全组 ID。

以下示例用于将 my-endpoint 代理终端节点重命名为 new-endpoint-name。

Example

对于 Linux、macOS 或 Unix：

```
aws rds modify-db-proxy-endpoint \
  --db-proxy-endpoint-name my-endpoint \
  --new-db-proxy-endpoint-name new-endpoint-name
```

对于 Windows：

```
aws rds modify-db-proxy-endpoint ^
  --db-proxy-endpoint-name my-endpoint ^
  --new-db-proxy-endpoint-name new-endpoint-name
```

RDS API

要修改代理终端节点，请使用 RDS API ModifyDBProxyEndpoint 操作。

删除代理终端节点

要删除代理的端点，请按照以下说明操作：

> **Note**
>
> 您无法删除 RDS 代理自动为每个代理创建的默认代理端点。
> 删除代理时，RDS Proxy 会自动删除所有关联的终端节点。

控制台

使用 AWS Management Console 删除代理终端节点

1. 在导航窗格中，选择 Proxies (代理)。
2. 在列表中，选择要删除其终端节点的代理。单击代理名称以查看其详细信息页面。
3. 在 Proxy endpoints（代理终端节点）部分中，选择要删除的终端节点。您可以在列表中选择一个或多个终端节点，或单击单个终端节点的名称以查看详细信息页面。
4. 在代理详细信息页面上的 Proxy endpoints（代理终端节点）部分中，选择 Delete（删除）。或者，在代理端点详细信息页面上，对于操作，选择编辑。

AWS CLI

要删除代理终端节点，请运行 delete-db-proxy-endpoint 命令并包含以下必需参数：

- `--db-proxy-endpoint-name`

以下命令用于删除名为 `my-endpoint` 的代理终端节点。

对于 Linux、macOS 或 Unix：

```
aws rds delete-db-proxy-endpoint \
  --db-proxy-endpoint-name my-endpoint
```

对于 Windows：

```
aws rds delete-db-proxy-endpoint ^
  --db-proxy-endpoint-name my-endpoint
```

RDS API

要使用 RDS API 删除代理终端节点，请运行 DeleteDBProxyEndpoint 操作。对于 `DBProxyEndpointName` 参数，指定代理终端节点的名称。

使用 Amazon CloudWatch 监控 RDS Proxy 指标

您可以使用 Amazon CloudWatch 监控 RDS Proxy。CloudWatch 可从代理收集和处理原始数据，并将数据处理为便于读取的近乎实时的指标。要在 CloudWatch 控制台中查找这些指标，请依次选择 Metrics (指标)、RDS 和 Per-Proxy Metrics (每个代理指标)。有关更多信息，请参阅 Amazon CloudWatch 用户指南 中的使用 Amazon CloudWatch 指标。

> **Note**
>
> RDS 为与代理关联的每个底层 Amazon EC2 实例发布这些指标。一个代理可能由多个 EC2 实例提供服务。使用 CloudWatch 统计数据可汇总所有关联实例的代理值。
> 其中一些指标可能在代理第一次成功连接之后才可见。

在 RDS Proxy 日志中，每个条目都以关联的代理终端节点的名称作为前缀。此名称可以是您为用户定义的端点指定的名称，或者是执行读/写请求的代理的默认端点的特殊名称 `default`。

所有 RDS Proxy 指标都在 proxy 组中。

每个代理终端节点都有自己的 CloudWatch 指标。您可以单独监控每个代理终端节点的使用情况。有关代理终端节点的更多信息，请参阅 使用 Amazon RDS Proxy 终端节点。

您可以使用以下维度集之一聚合每个指标的值。例如，通过使用 ProxyName 维度集，您可以分析特定代理的所有流量。通过使用其他维度集，您可以用不同的方式拆分指标。您可以根据每个代理的不同终端节点或目标数据库，或者每个数据库的读/写和只读流量来拆分指标。

- 维度集 1：ProxyName
- 维度集 2：ProxyName、EndpointName
- 维度集 3：ProxyName、TargetGroup、Target
- 维度集 4：ProxyName、TargetGroup、TargetRole

指标	描述	有效期	CloudWatch 维度集
AvailabilityPercentage	目标组在维度指示的角色中可用的时间百分比。每分钟报告一次此指标。此指标最有用的统计数据是 Sum。	1 minute	Dimension set 4
ClientConnections	当前客户端连接数。每分钟报告一次此指标。此指标最有用的统计数据是 Sum。	1 minute	Dimension set 1, Dimension set 2
ClientConnectionsClosed	已关闭的客户端连接数。此指标最有用的统计数据是 Sum。	1 分钟及更久	Dimension set 1, Dimension set 2
ClientConnectionsInSetup	已打开但尚未完成设置的当前客户端连接数量。每分钟报告一次此指标。该指标最	1 minute	Dimension set 1, Dimension set 2

指标	描述	有效期	CloudWatch 维度集
	有用的统计数据是 Sum。		
ClientConnectionsNoTLS	当前没有传输层安全性 (TLS) 的客户端连接的数量。每分钟报告一次此指标。此指标最有用的统计数据是 Sum。	1 minute	Dimension set 1, Dimension set 2
ClientConnectionsReceived	收到的客户端连接请求数。此指标最有用的统计数据是 Sum。	1 分钟及更久	Dimension set 1, Dimension set 2
ClientConnectionsSetupFailedAuth	由于身份验证或 TLS 配置错误而失败的客户端连接尝试次数。此指标最有用的统计数据是 Sum。	1 分钟及更久	Dimension set 1, Dimension set 2
ClientConnectionsSetupSucceeded	使用任何具有或不具有 TLS 的身份验证机制成功建立的客户端连接数。此指标最有用的统计数据是 Sum。	1 分钟及更久	Dimension set 1, Dimension set 2
ClientConnectionsTLS	当前具有 TLS 的客户端连接的数量。每分钟报告一次此指标。此指标最有用的统计数据是 Sum。	1 minute	Dimension set 1, Dimension set 2
DatabaseConnectionRequests	创建数据库连接的请求数。此指标最有用的统计数据是 Sum。	1 分钟及更久	Dimension set 1, Dimension set 3, Dimension set 4

指标	描述	有效期	CloudWatch 维度集
DatabaseConnectionRequestsWithTLS	创建具有 TLS 的数据库连接的请求数。此指标最有用的统计数据是 Sum。	1 分钟及更久	Dimension set 1, Dimension set 3, Dimension set 4
DatabaseConnections	当前数据库连接数。每分钟报告一次此指标。此指标最有用的统计数据是 Sum。	1 minute	Dimension set 1, Dimension set 3, Dimension set 4
DatabaseConnectionsBorrowLatency	被监视的代理建立数据库连接所花费的时间（以微秒为单位）。此指标最有用的统计数据是 Sum。	1 分钟及更久	Dimension set 1, Dimension set 2
DatabaseConnectionsCurrentlyBorrowed	当前处于借用状态的数据库连接数。每分钟报告一次此指标。此指标最有用的统计数据是 Sum。	1 minute	Dimension set 1, Dimension set 3, Dimension set 4
DatabaseConnectionsCurrentlyInTransaction	事务中的当前数据库连接数。每分钟报告一次此指标。此指标最有用的统计数据是 Sum。	1 minute	Dimension set 1, Dimension set 3, Dimension set 4
DatabaseConnectionsCurrentlySessionPinned	由于客户端请求中更改会话状态的操作，当前固定的当前数据库连接数。每分钟报告一次此指标。此指标最有用的统计数据是 Sum。	1 minute	Dimension set 1, Dimension set 3, Dimension set 4

指标	描述	有效期	CloudWatch 维度集
DatabaseConnectionsSetupFailed	失败的数据库连接请求数。此指标最有用的统计数据是 Sum。	1 分钟及更久	Dimension set 1, Dimension set 3, Dimension set 4
DatabaseConnectionsSetupSucceeded	使用或不使用 TLS 成功建立的数据库连接数。此指标最有用的统计数据是 Sum。	1 分钟及更久	Dimension set 1, Dimension set 3, Dimension set 4
DatabaseConnectionsWithTLS	具有 TLS 的当前数据库连接数。每分钟报告一次此指标。此指标最有用的统计数据是 Sum。	1 minute	Dimension set 1, Dimension set 3, Dimension set 4
MaxDatabaseConnectionsAllowed	允许的最大数据库连接数。每分钟报告一次此指标。此指标最有用的统计数据是 Sum。	1 minute	Dimension set 1, Dimension set 3, Dimension set 4
QueryDatabaseResponseLatency	数据库响应查询所花费的时间（以微秒为单位）。此指标最有用的统计数据是 Average。	1 分钟及更久	Dimension set 1, Dimension set 2, Dimension set 3, Dimension set 4
QueryRequests	收到的查询数。包含多个语句的查询被计为一个查询。此指标最有用的统计数据是 Sum。	1 分钟及更久	Dimension set 1, Dimension set 2

指标	描述	有效期	CloudWatch 维度集
QueryRequestsNoTLS	从非 TLS 连接收到的查询数。包含多个语句的查询被计为一个查询。此指标最有用的统计数据是 Sum。	1 分钟及更久	Dimension set 1, Dimension set 2
QueryRequestsTLS	从 TLS 连接收到的查询数。包含多个语句的查询被计为一个查询。此指标最有用的统计数据是 Sum。	1 分钟及更久	Dimension set 1, Dimension set 2
QueryResponseLatency	从获取查询请求到代理响应该请求的时间（以微秒为单位）。此指标最有用的统计数据是 Average。	1 分钟及更久	Dimension set 1, Dimension set 2

您可以在AWS Management Console中的 CloudWatch 下找到 RDS Proxy 活动的日志。每个代理在 Log groups (日志组) 页面中都有一个条目。

Important

这些日志供相关人员进行故障排除，而不是用于编程访问。日志的格式和内容可能会发生变化。

特别是，较旧的日志不包含指示每个请求的终端节点的任何前缀。在较新的日志中，每个条目都以关联的代理终端节点的名称作为前缀。此名称可以是您为用户定义的终端节点指定的名称，也可以是使用代理的默认终端节点的请求的特殊名称 default。

使用 RDS 代理事件

事件表示环境中发生的更改，例如 AWS 环境或软件即服务（SaaS）合作伙伴中的服务或应用程序。也可以是您自己的一个自定义应用程序或服务。例如，Amazon Aurora 在创建或修改 RDS Proxy 时生

成事件。Amazon Aurora 将事件近乎实时地传输到 Amazon EventBridge。接下来，您可以查找可以订阅的 RDS Proxy 事件列表以及 RDS Proxy 事件示例。

有关使用事件的详细信息，请参阅以下内容：

- 有关如何使用 AWS Management Console、AWS CLI 或 RDS API 查看事件的说明，请参阅 查看 Amazon RDS 事件。
- 若要了解如何配置 Amazon Aurora 向 EventBridge 发送事件，请参阅 创建对 Amazon Aurora 事件触发的规则。

RDS Proxy 事件

下表显示了 RDS Proxy 为源类型时的事件类别和事件列表。

类别	RDS 事件 ID	消息	注意
配置更改	RDS-EVENT-0204	RDS 修改了数据库代理 *name*。	无
配置更改	RDS-EVENT-0207	RDS 修改了数据库代理 *name* 的端点。	无
配置更改	RDS-EVENT-0213	RDS 检测到已添加数据库实例并自动将其添加到数据库代理 *name* 的目标组。	无
配置更改	RDS-EVENT-0214	RDS 检测到已删除数据库实例 *name* 并自动将其从数据库代理 *name* 的目标组 *name* 中删除。	无
配置更改	RDS-EVENT-0215	RDS 检测到已删除数据库集群 *name* 并自动将其从数据库代理 *name* 的目标组 *name* 中删除。	无
创建	RDS-EVENT-0203	RDS 创建了数据库代理 *name*。	无

类别	RDS 事件 ID	消息	注意
创建	RDS-EVENT-0206	RDS 为数据库代理 *name* 创建了端点 *name*。	无
删除	RDS-EVENT-0205	RDS 删除了数据库代理 *name*。	无
删除	RDS-EVENT-0208	RDS 删除了数据库代理 *name* 的端点 *name*。	无
失败	RDS-EVENT-0243	RDS 无法为代理 *name* 预调配容量，因为您的子网 *name* 中没有足够的 IP 地址可用。要解决此问题，请确保您的子网具有最少的未使用 IP 地址数（如 RDS 代理文档中所建议）。	要确定您的实例类的建议数量，请参阅计划 IP 地址容量。
失败	RDS-EVENT-0275	RDS 限制了一些与数据库代理##的连接。从客户端到代理的并发连接请求数已超出了限制。	无

以下是采用 JSON 格式的 RDS 代理事件示例。该事件显示 RDS 已修改名为 `my-rds-proxy` 的 RDS Proxy 的名为 `my-endpoint` 的端点。事件 ID 为 RDS-EVENT-0207。

```
{
  "version": "0",
  "id": "68f6e973-1a0c-d37b-f2f2-94a7f62ffd4e",
  "detail-type": "RDS DB Proxy Event",
  "source": "aws.rds",
  "account": "123456789012",
  "time": "2018-09-27T22:36:43Z",
  "region": "us-east-1",
  "resources": [
     "arn:aws:rds:us-east-1:123456789012:db-proxy:my-rds-proxy"
  ],
  "detail": {
```

```
    "EventCategories": [
      "configuration change"
    ],
    "SourceType": "DB_PROXY",
    "SourceArn": "arn:aws:rds:us-east-1:123456789012:db-proxy:my-rds-proxy",
    "Date": "2018-09-27T22:36:43.292Z",
    "Message": "RDS modified endpoint my-endpoint of DB Proxy my-rds-proxy.",
    "SourceIdentifier": "my-endpoint",
    "EventID": "RDS-EVENT-0207"
  }
}
```

RDS Proxy 命令行示例

要了解连接命令和 SQL 语句的组合如何与 RDS Proxy 交互，请查看以下示例。

示例

- Preserving Connections to a MySQL Database Across a Failover
- Adjusting the max_connections Setting for an Aurora DB Cluster

Example 在故障转移时保留与 MySQL 数据库的连接

此 MySQL 示例演示在失效转移期间打开的连接如何继续工作。例如，当您重启数据库或数据库由于某个问题而变得不可用时。本示例使用一个名为 `the-proxy` 的代理，以及一个包含数据库实例 `instance-8898` 和 `instance-9814` 的 Aurora 数据库集群。从 Linux 命令行运行 `failover-db-cluster` 命令时，代理连接到的写入器实例将更改为不同的数据库实例。您可以看到，与代理关联的数据库实例在连接保持打开状态时发生更改。

```
$ mysql -h the-proxy.proxy-demo.us-east-1.rds.amazonaws.com -u admin_user -p
Enter password:
...

mysql> select @@aurora_server_id;
+--------------------+
| @@aurora_server_id |
+--------------------+
| instance-9814      |
+--------------------+
1 row in set (0.01 sec)
```

```
mysql>
[1]+  Stopped                 mysql -h the-proxy.proxy-demo.us-east-1.rds.amazonaws.com
 -u admin_user -p
$ # Initially, instance-9814 is the writer.
$ aws rds failover-db-cluster --db-cluster-identifier cluster-56-2019-11-14-1399
JSON output
$ # After a short time, the console shows that the failover operation is complete.
$ # Now instance-8898 is the writer.
$ fg
mysql -h the-proxy.proxy-demo.us.us-east-1.rds.amazonaws.com -u admin_user -p

mysql> select @@aurora_server_id;
+--------------------+
| @@aurora_server_id |
+--------------------+
| instance-8898      |
+--------------------+
1 row in set (0.01 sec)

mysql>
[1]+  Stopped                 mysql -h the-proxy.proxy-demo.us-east-1.rds.amazonaws.com
 -u admin_user -p
$ aws rds failover-db-cluster --db-cluster-identifier cluster-56-2019-11-14-1399
JSON output
$ # After a short time, the console shows that the failover operation is complete.
$ # Now instance-9814 is the writer again.
$ fg
mysql -h the-proxy.proxy-demo.us-east-1.rds.amazonaws.com -u admin_user -p

mysql> select @@aurora_server_id;
+--------------------+
| @@aurora_server_id |
+--------------------+
| instance-9814      |
+--------------------+
1 row in set (0.01 sec)
+---------------+---------------+
| Variable_name | Value         |
+---------------+---------------+
| hostname      | ip-10-1-3-178 |
+---------------+---------------+
1 row in set (0.02 sec)
```

Example 调整 Aurora 数据库集群的 max_connections 设置

本示例演示如何调整 Aurora MySQL 数据库集群的 max_connections 设置。为此，请根据与 MySQL 5.7 兼容的集群的原定设置参数设置，创建自己的数据库集群参数组。您可以为 max_connections 设置指定一个值，从而覆盖设置默认值的公式。将数据库集群参数组与数据库集群关联。

```
export REGION=us-east-1
export CLUSTER_PARAM_GROUP=rds-proxy-mysql-57-max-connections-demo
export CLUSTER_NAME=rds-proxy-mysql-57

aws rds create-db-parameter-group --region $REGION \
  --db-parameter-group-family aurora-mysql5.7 \
  --db-parameter-group-name $CLUSTER_PARAM_GROUP \
  --description "Aurora MySQL 5.7 cluster parameter group for RDS Proxy demo."

aws rds modify-db-cluster --region $REGION \
  --db-cluster-identifier $CLUSTER_NAME \
  --db-cluster-parameter-group-name $CLUSTER_PARAM_GROUP

echo "New cluster param group is assigned to cluster:"
aws rds describe-db-clusters --region $REGION \
  --db-cluster-identifier $CLUSTER_NAME \
  --query '*[*].{DBClusterParameterGroup:DBClusterParameterGroup}'

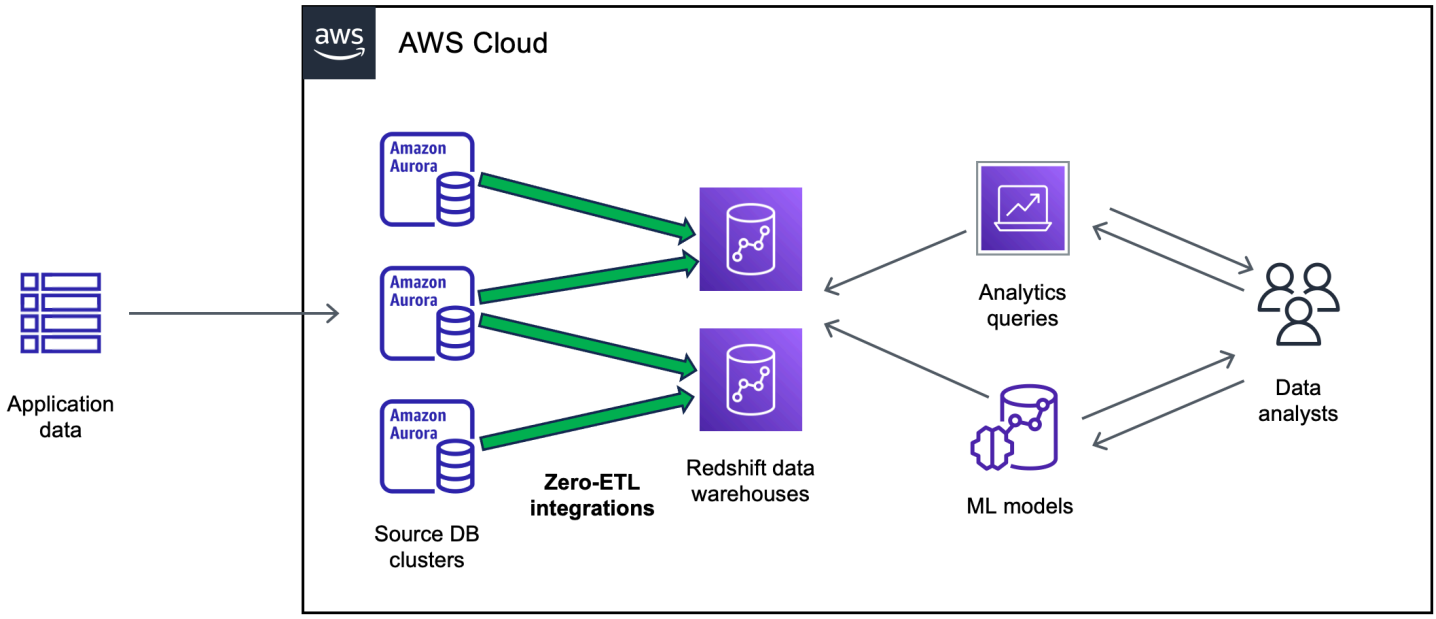
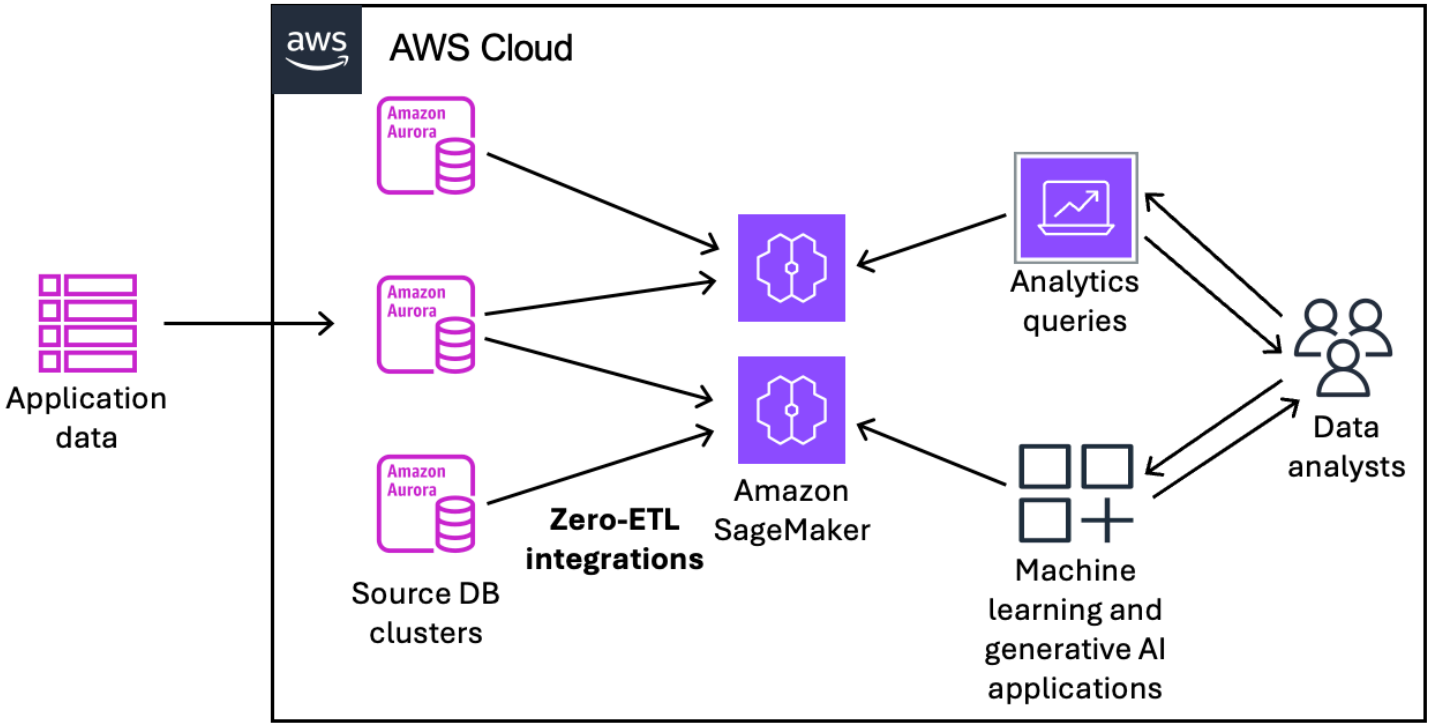
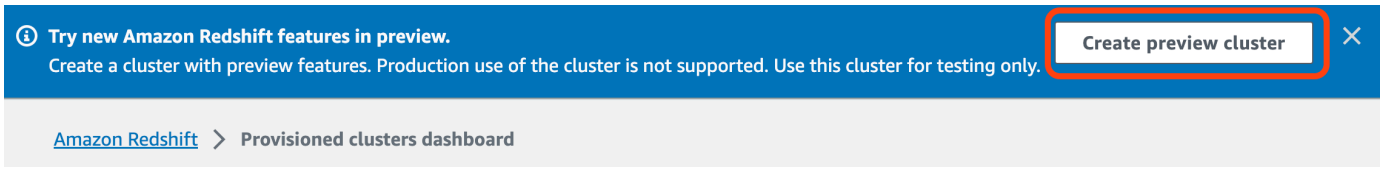
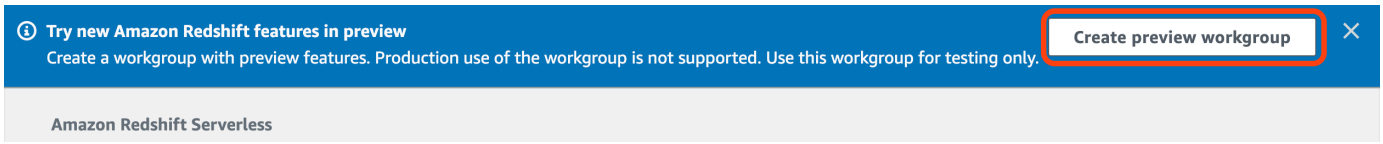
echo "Current value for max_connections:"
aws rds describe-db-cluster-parameters --region $REGION \
  --db-cluster-parameter-group-name $CLUSTER_PARAM_GROUP \
  --query '*[*].{ParameterName:ParameterName,ParameterValue:ParameterValue}' \
  --output text | grep "^max_connections"

echo -n "Enter number for max_connections setting: "
read answer

aws rds modify-db-cluster-parameter-group --region $REGION --db-cluster-parameter-
group-name $CLUSTER_PARAM_GROUP \
  --parameters "ParameterName=max_connections,ParameterValue=$
$answer,ApplyMethod=immediate"

echo "Updated value for max_connections:"
aws rds describe-db-cluster-parameters --region $REGION \
  --db-cluster-parameter-group-name $CLUSTER_PARAM_GROUP \
  --query '*[*].{ParameterName:ParameterName,ParameterValue:ParameterValue}' \
```

```
  --output text | grep "^max_connections"
```

RDS 代理故障排除

下面，您可以找到一些常见 RDS 代理问题的故障排除思路，以及有关 RDS Proxy 的 CloudWatch Logs 的信息。

在 RDS Proxy 日志中，每个条目都以关联的代理终端节点的名称作为前缀。此名称可以是您为用户定义的端点指定的名称。或者，对于执行读/写请求的代理的默认端点，它可以是特殊名称 `default`。有关代理终端节点的更多信息，请参阅 使用 Amazon RDS Proxy 终端节点。

主题

- 验证代理的连接
- 常见问题和解决方案
- 排查 RDS for MySQL 的 RDS 代理问题
- 排查 RDS for PostgreSQL 的 RDS 代理问题

验证代理的连接

您可以使用以下命令验证连接中的所有组件（例如代理、数据库和计算实例）是否均可相互进行通信。

使用 describe-db-proxies 命令检查代理本身。此外，使用 describe-db-proxy-target-groups 命令检查关联的目标组。检查目标的详细信息是否与您打算与代理关联的 Aurora 集群匹配。使用如下命令。

```
aws rds describe-db-proxies --db-proxy-name $DB_PROXY_NAME
aws rds describe-db-proxy-target-groups --db-proxy-name $DB_PROXY_NAME
```

要确认代理可以连接到底层数据库，请使用 describe-db-proxy-targets 命令检查目标组中指定的目标。使用如下命令。

```
aws rds describe-db-proxy-targets --db-proxy-name $DB_PROXY_NAME
```

describe-db-proxy-targets 命令的输出包括一个 `TargetHealth` 字段。您可以检查 `State` 内部的字段 `Reason`、`Description` 和 `TargetHealth`，以检查代理是否可以与底层数据库实例进行通信。

- `State` 值 `AVAILABLE` 表示代理可以连接到数据库实例。

- State 值 UNAVAILABLE 表示存在临时或永久的连接问题。在这种情况下，请检查 Reason 和 Description 字段。例如，如果 Reason 的值为 PENDING_PROXY_CAPACITY，请在代理完成其扩展操作后，重试连接。如果 Reason 的值为 UNREACHABLE、CONNECTION_FAILED 或 AUTH_FAILURE，请使用 Description 字段中的说明来帮助您诊断问题。
- 在更改为 State 或 REGISTERING 之前，AVAILABLE 字段的值可能在短时间内为 UNAVAILABLE。

如果以下 Netcat 命令 (nc) 成功，您可以从登录的 EC2 实例或其他系统访问代理终端节点。如果您与代理和关联数据库不在同一 VPC 中，则该命令将报告失败。您可能未在同一 VPC 中就可以直接登录到数据库。但是，除非在同一 VPC 中，否则无法登录到代理。

```
nc -zx MySQL_proxy_endpoint 3306

nc -zx PostgreSQL_proxy_endpoint 5432
```

您可以使用以下命令来确保 EC2 实例具有所需属性。特别是，EC2 实例的 VPC 必须与代理连接到的 RDS 数据库实例Aurora 集群的 VPC 相同。

```
aws ec2 describe-instances --instance-ids your_ec2_instance_id
```

检查用于代理的 Secrets Manager 密钥。

```
aws secretsmanager list-secrets
aws secretsmanager get-secret-value --secret-id your_secret_id
```

确保 get-secret-value 显示的 SecretString 字段被编码为包含 username 和 password 字段的 JSON 字符串。以下示例显示了 SecretString 字段的格式。

```
{
  "ARN": "some_arn",
  "Name": "some_name",
  "VersionId": "some_version_id",
  "SecretString": '{"username":"some_username","password":"some_password"}',
  "VersionStages": [ "some_stage" ],
  "CreatedDate": some_timestamp
}
```

常见问题和解决方案

本节介绍使用 RDS 代理时的一些常见问题和潜在的解决方案。

运行 `aws rds describe-db-proxy-targets` CLI 命令后，如果 `TargetHealth` 描述显示 `Proxy does not have any registered credentials`，请验证以下内容：

- 已经为用户注册了访问代理的凭证。
- 用于访问代理使用的 Secrets Manager 密钥的 IAM 角色是有效的。

创建或连接数据库代理时，您可能会遇到以下 RDS 事件。

类别	RDS 事件 ID	描述
失败	RDS-EVENT-0243	RDS 无法为代理预调配容量，因为您的子网中没有足够的 IP 地址可用。要解决此问题，请确保您的子网具有最少的未使用 IP 地址数。要确定您的实例类的建议数量，请参阅计划 IP 地址容量。
失败	RDS-EVENT-0275	RDS 限制了一些与数据库代理##的连接。从客户端到代理的并发连接请求数已超出了限制。

创建新代理或连接到代理时，您可能会遇到以下问题。

错误	原因或解决方法
`403: The security token included in the request is invalid`	选择现有 IAM 角色，而不是选择创建新角色。

排查 RDS for MySQL 的 RDS 代理问题

连接到 MySQL 代理时，您可能会遇到以下问题。

错误	原因或解决方法
`ERROR 1040 (HY000): Connections rate limit exceeded (`*`limit_value`*` )`	从客户端到代理的连接请求速率超过了限制。
`ERROR 1040 (HY000): IAM authentication rate limit exceeded`	从客户端到代理的具有 IAM 身份验证的并发请求数超过了限制。
`ERROR 1040 (HY000): Number simultaneous connections exceeded (`*`limit_value`*` )`	从客户端到代理的并发连接请求数超出了限制。
`ERROR 1045 (28000): Access denied for user '`*`DB_USER`*`'@'%' (usi password: YES)`	代理使用的 Secrets Manager 密钥与现有数据库用户的用户名和密码不匹配。更新 Secrets Manager 密钥中的凭证，或者确保数据库用户存在并且具有与密钥中的密码相同的密码。
`ERROR 1105 (HY000): Unknown error`	出现未知错误。
`ERROR 1231 (42000):`	为 `character_set_client` 参数设置的值无效。例如，值 ucs2 无效，因为它会导致 MySQL 服务器崩溃。

错误	原因或解决方法
`Variable ''character_set_client'' can't be set to the value of value`	
`ERROR 3159 (HY000): This RDS Proxy requires TLS connections.`	您在代理中启用了需要传输层安全性设置，但您的连接将参数 `ssl-mode=DISABLED` 包含在 MySQL 客户端中。请执行以下任一操作： • 禁用代理的需要传输层安全性设置。 • 在 MySQL 客户端中，使用最小设置 `ssl-mode=REQUIRED` 连接到数据库。
`ERROR 2026 (HY000): SSL connection error: Internal Server Error`	与代理的 TLS 握手失败。一些可能的原因包括： • SSL 是必需的，但服务器不支持它。 • 发生内部服务器错误。 • 发生握手错误。
`ERROR 9501 (HY000): Timed-out waiting to acquire database connection`	等待获取数据库连接时代理超时。一些可能的原因包括： • 代理无法建立数据库连接，因为已达到最大连接数 • 代理无法建立数据库连接，因为数据库不可用。

排查 RDS for PostgreSQL 的 RDS 代理问题

连接到 PostgreSQL 代理时，您可能会遇到以下问题。

错误	原因	解决方案
`ERROR 28000: IAM authentication is`	用户在 PostgreSQL 客户端中，尝试使用具有设置	用户需要在 PostgreSQL 客户端中使用最小设置 `sslmode=require` 连接到数据库。有关

错误	原因	解决方案
`allowed only with SSL connections.`	`sslmode=disable` 的 IAM 身份验证连接到数据库。	更多信息，请参阅 PostgreSQL SSL 支持文档。
`ERROR 28000: This RDS proxy has no credentials for the role` *`role_name`* `. Check the credentials for this role and try again.`	这个角色没有 Secrets Manager 密钥。	为此角色添加 Secrets Manager 密钥。有关更多信息，请参阅 设置 RDS 代理的 AWS Identity and Access Management（IAM）策略。
`ERROR 28000: RDS supports only IAM, MD5, or SCRAM authentication.`	用于连接到代理的数据库客户端正在使用代理当前不支持的身份验证机制。	如果您不使用 IAM 身份验证，请使用 MD5 或 SCRAM 密码身份验证。
`ERROR 28000: A user name is missing from the connection startup packet. Provide a user name for this connection.`	用于连接到代理的数据库客户端在尝试建立连接时未发送用户名。	请确保在使用您选择的 PostgreSQL 客户端设置与代理的连接时定义用户名。
`ERROR 28000: IAM is allowed only with SSL connections.`	客户端尝试使用 IAM 身份验证进行连接，但未启用 SSL。	在 PostgreSQL 客户端中启用 SSL。
`ERROR 28000: This RDS Proxy requires TLS connections.`	用户启用了需要传输层安全性选项，但尝试在 PostgreSQL 客户端中使用 `sslmode=disable` 进行连接。	要修复此错误，请执行以下操作之一： • 禁用代理的需要传输层安全性选项。 • 在 PostgreSQL 客户端中，使用最小设置 `sslmode=allow` 连接到数据库。

错误	原因	解决方案
`ERROR 28P01: IAM authentication failed for user` *`user_name`* `. Check the IAM token for this user and try again.`	该错误可能是由于以下原因引起的： • 客户端提供了错误的 IAM 用户名。 • 客户端向用户提供了错误的 IAM 授权令牌。 • 客户端在使用没有所需权限的 IAM 策略。 • 客户端向用户提供了过期的 IAM 授权令牌。	要修复此错误，请执行以下操作： 1. 确认提供的 IAM 用户存在。 2. 确认 IAM 授权令牌属于提供的 IAM 用户。 3. 确认 IAM 策略具有足够的 RDS 权限。 4. 检查所用 IAM 授权令牌的有效性。
`ERROR 28P01: The password that was provided for the role` *`role_name`* `is wrong.`	此角色的密码与 Secrets Manager 密钥不匹配。	检查 Secrets Manager 中此角色的密钥，以查看密码是否与 PostgreSQL 客户端中使用的密码相同。
`ERROR 28P01: The IAM authentication failed for the role` *`role_name`* `. Check the IAM token for this role and try again.`	用于 IAM 身份验证的 IAM 令牌存在问题。	生成新的身份验证令牌并在新连接中使用它。
`ERROR 0A000: Feature not supported: RDS Proxy supports only version 3.0 of the PostgreSQL messaging protocol.`	用于连接到代理的 PostgreSQL 客户端使用早于 3.0 的协议。	使用支持 3.0 消息传递协议的较新 PostgreSQL 客户端。如果您使用的是 PostgreSQL `psql` CLI，请使用大于或等于 7.4 的版本。

错误	原因	解决方案
`ERROR 0A000: Feature not supported: RDS Proxy currently doesn't support streaming replication mode.`	用于连接到代理的 PostgreSQL 客户端正在尝试使用流复制模式，该模式目前不受 RDS 代理支持。	在用于连接的 PostgreSQL 客户端中关闭流复制模式。
`ERROR 0A000: Feature not supported: RDS Proxy currently doesn't support the option` *`option_name`* `.`	通过启动消息，用于连接到代理的 PostgreSQL 客户端正在请求 RDS 代理当前不支持的选项。	在用于连接的 PostgreSQL 客户端中，关闭上述消息中显示为不支持的选项。
`ERROR 53300: The IAM authentication failed because of too many competing requests.`	从客户端到代理的具有 IAM 身份验证的并发请求数超过了限制。	降低使用 PostgreSQL 客户端中的 IAM 身份验证建立连接的速率。
`ERROR 53300: The maximum number of client connections to the proxy exceeded` *`number_value`* `.`	从客户端到代理的并发连接请求数超出了限制。	减少从 PostgreSQL 客户端到此 RDS 代理的活动连接数。
`ERROR 53300: Rate of connection to proxy exceeded` *`number_value`* `.`	从客户端到代理的连接请求速率超过了限制。	降低从 PostgreSQL 客户端建立连接的速率。
`ERROR XX000: Unknown error.`	出现未知错误。	请联系 AWS Support，以便我们调查此问题。

错误	原因	解决方案
`ERROR 08000: Timed-out waiting to acquire database connection.`	代理在 `ConnectionBorrowTimeout` 设置指定的持续时间内等待获取数据库连接时超时。一些可能的原因包括： • 代理无法建立数据库连接，因为已达到最大连接数。 • 代理无法建立数据库连接，因为数据库不可用，或者建立到数据库的连接所花费的时间超过了所配置的 `ConnectionBorrowTimeout` 。	可能的解决方案包括以下各项： • 避免固定代理连接。请参阅避免固定 RDS 代理。 • 检查 `ConnectionBorrowTimeout` 和 `MaxConnectionsPercent` 设置。请参阅RDS 代理连接注意事项。 • 查看目标可用性。请参阅使用 Amazon CloudWatch 监控 RDS Proxy 指标 中的 `AvailabilityPercentage` 。
`ERROR XX000: Request returned an error: database_error .`	从代理建立的数据库连接返回错误。	解决方案取决于特定的数据库错误。示例为：`Request returned an error: database "your-database-name" does not exist`。这意味着数据库服务器上不存在指定的数据库名称。或者，这意味着服务器上不存在用作数据库名称（如果未指定数据库名称）的用户名。

对已删除的 **postgres** 数据库进行故障排除

如果您错误地从实例中删除了 `postgres` 数据库，则需要还原数据库以还原与实例的连接。在数据库实例中运行以下命令：

```
CREATE DATABASE postgres;
```

```
GRANT CONNECT ON DATABASE postgres TO rdsproxyadmin;
```

将 RDS Proxy 与 AWS CloudFormation 一起使用

您可以将 RDS Proxy 与 AWS CloudFormation 一起使用。这有助于您创建相关资源组。此类组可能包含可连接到新创建的 Aurora 数据库集群的代理。AWS CloudFormation 中的 RDS Proxy 支持涉及两种新的注册表类型：DBProxy 和 DBProxyTargetGroup。

以下列表显示了 RDS Proxy 的 AWS CloudFormation 模板示例。

```
Resources:
 DBProxy:
   Type: AWS::RDS::DBProxy
   Properties:
     DBProxyName: CanaryProxy
     EngineFamily: MYSQL
     RoleArn:
      Fn::ImportValue: SecretReaderRoleArn
     Auth:
       - {AuthScheme: SECRETS, SecretArn: !ImportValue ProxySecret, IAMAuth: DISABLED}
     VpcSubnetIds:
       Fn::Split: [",", "Fn::ImportValue": SubnetIds]

 ProxyTargetGroup:
   Type: AWS::RDS::DBProxyTargetGroup
   Properties:
     DBProxyName: CanaryProxy
     TargetGroupName: default
     DBInstanceIdentifiers:
       - Fn::ImportValue: DBInstanceName
   DependsOn: DBProxy
```

有关此示例中的资源的更多信息，请参阅 DBProxy 和 DBProxyTargetGroup。

有关您可以使用 AWS CloudFormation 创建的资源的更多信息，请参阅 RDS 资源类型参考。

在 Aurora Global Database 中使用 RDS 代理

Aurora Global Database 是跨多个 AWS 区域的单一数据库，允许在发生任何区域级的中断时提供低延迟的全局读取和灾难恢复能力。它为您的部署提供了内置的容错功能，因为数据库实例不依赖于单

个 AWS 区域，而是依赖于多个区域和不同的可用区。有关更多信息，请参阅 使用 Amazon Aurora Global Database。

可以将 RDS 代理用于 Aurora Global Database 中的任何数据库集群。在开始一起使用这些特征之前，请确保您熟悉以下信息。

> **Important**
>
> 如果数据库集群是开启写入转发的全局数据库的一部分，则按照分配给写入转发的限额来减小代理的 `MaxConnectionsPercent` 值。写入转发限额在数据库集群参数 `aurora_fwd_writer_max_connections_pct` 中设置。有关写入转发的信息，请参阅 在 Amazon Aurora Global Database 中使用写入转发。

对全局数据库使用 RDS 代理的限制

当 Aurora 数据库集群开启写入转发时，RDS 代理不支持 `aurora_replica_read_consistency` 变量的 `SESSION` 值。设置此值可能导致意外行为。

RDS 代理端点如何与全局数据库配合使用

当您了解 RDS 代理端点如何与全局数据库配合使用时，您可以通过这两种特征更好地管理使用 Aurora 数据库的应用程序。

对于以全局数据库的主集群作为注册目标的代理，代理端点的工作方式与任何 Aurora 数据库集群的工作方式相同。代理的读/写端点会将所有请求发送到集群的写入器实例。代理的只读端点将所有请求发送到读取器实例。如果在连接打开时读取器变得不可用，RDS 代理会将连接上的后续查询重定向到另一个读取器实例。对于以辅助集群作为已注册目标的代理，发送到代理的只读端点的请求也会发送到读取器实例。由于集群没有写入器实例，因此发送到读/写端点的请求会失败并显示错误“`The target group doesn't have any associated read/write instances`”。

全球数据库切换和失效转移操作均涉及主数据库集群和其中一个辅助数据库集群之间的角色切换。当选定的辅助集群成为新的主集群时，其一个读取器实例将提升为写入器。此数据库实例现在是全局集群的新写入器实例。请确保将应用程序的写入操作重定向到与新的主集群关联的代理的相应读/写端点。此代理端点可能是默认端点或自定义读/写端点。

RDS 代理通过读/写端点对所有请求进行排队，并在新的主集群可用后立即将它们发送到此集群的写入器实例。无论切换或失效转移操作是否完成，它都会这样做。在切换或失效转移期间，旧主集群的代理

的默认端点仍接受写入操作。但是，一旦该集群成为辅助集群，所有写入操作都会失败。要了解如何以及何时执行特定的全球切换或失效转移任务，请参阅以下主题：

- 全球数据库切换 – 对 Amazon Aurora Global Database 执行切换
- 全球数据库失效转移 – 从计划外停机中恢复 Amazon Aurora Global Database

Aurora 零 ETL 集成

Aurora 与 Amazon Redshift 和 Amazon SageMaker 的零 ETL 集成支持使用 Aurora 中的数据进行近乎实时的分析和机器学习（ML）。这是一个完全托管式解决方案，用于在将事务数据写入 Aurora 数据库集群后，使这些数据在分析目标中可用。提取、转换、加载（ETL）是将来自多个来源的数据合并到一个大型中央数据仓库的过程。

借助零 ETL 集成，Aurora 数据库集群中的数据可在 Amazon Redshift 或 Amazon SageMaker 智能湖仓中以近乎实时的方式使用。一旦这些数据进入目标数据仓库或数据湖中，您就可以使用内置功能为分析、机器学习和人工智能工作负载提供支持，例如机器学习、实体化视图、数据共享、对多个数据存储和数据湖的联合访问，以及与 Amazon SageMaker AI、QuickSight 及其它 AWS 服务的集成。

要创建零 ETL 集成，您需要将 Aurora 数据库集群指定为源，并将支持的数据仓库或智能湖仓指定为目标。该集成会将数据从源数据库复制到目标数据仓库或智能湖仓中。

下图展示了这项与 Amazon Redshift 进行零 ETL 集成的功能：



下图展示了这项与 Amazon SageMaker 智能湖仓进行零 ETL 集成的功能：



该集成还会监控数据管道的运行状况，并在可能的情况下从问题中恢复。您可以创建多个 Aurora 数据库集群与单个目标数据仓库或智能湖仓的集成，从而使您能够获得跨多个应用程序的全面洞察。

有关零 ETL 集成的定价的更多信息，请参阅 Amazon Aurora 定价和 Amazon Redshift 定价。

主题

- 优势
- 重要概念
- 限制
- 限额
- 支持的区域
- 开始使用 Aurora 与 Amazon Redshift 的零 ETL 集成
- 创建 Aurora 与 Amazon Redshift 的零 ETL 集成
- 创建 Aurora 与 Amazon SageMaker 智能湖仓的零 ETL 集成
- Aurora 零 ETL 集成的数据筛选
- 向源 Aurora 数据库集群中添加数据并在 Amazon Redshift 中对其进行查询
- 查看和监控 Aurora 零 ETL 集成
- 修改 Aurora 零 ETL 集成

- 删除 Aurora 零 ETL 集成
- Aurora 与 Amazon Redshift 的零 ETL 集成问题排查

优势

Aurora 零 ETL 集成具有以下好处：

- 帮助您从多个数据来源获得整体见解。
- 无需构建和维护执行提取、转换、加载（ETL）操作的复杂数据管道。零 ETL 集成通过为您预调配和管理管道来消除构建和管理管道所带来的挑战。
- 减少运营负担和成本，让您专注于改进应用程序。
- 让您利用目标目的地的分析和机器学习功能，从事务数据和其它数据中获得见解，从而有效地应对关键的、时间敏感的事件。

重要概念

在开始使用零 ETL 集成时，请考虑以下概念：

集成

完全托管式数据管道，可自动将事务数据和架构从 Aurora 数据库集群复制到数据仓库或目录。

源数据库集群

从中复制数据的 Aurora 数据库集群。可以指定一个使用预置数据库实例或 Aurora Serverless v2 数据库实例作为源的数据库集群。

目标

要将数据复制到的数据仓库或智能湖仓。有两种类型的数据仓库：预调配集群数据仓库和无服务器数据仓库。预调配集群数据仓库是一个由称作节点的各种计算资源构成的集合，这些节点已整理到名为集群的组中。无服务器数据仓库由存储计算资源的工作组和存放数据库对象和用户的命名空间组成。这两个数据仓库都运行分析引擎并包含一个或多个数据库。

目标智能湖仓由目录、数据库、表和视图组成。有关智能湖仓架构的更多信息，请参阅《Amazon SageMaker Unified Studio User Guide》中的 Amazon SageMaker Lakehouse components。

多个源数据库集群可以写入同一个目标。

有关更多信息，请参阅《Amazon Redshift 开发人员指南》中的数据仓库系统架构。

限制

以下限制适用于 Aurora 零 ETL 集成。

主题

- 一般限制
- Aurora MySQL 限制
- Aurora PostgreSQL 限制
- Amazon Redshift 限制
- Amazon SageMaker 智能湖仓限制

一般限制

- 源数据库集群必须与目标位于同一区域。
- 如果集群已有集成，则无法重命名数据库集群或其任何实例。
- 不能在相同的源数据库和目标数据库之间创建多个集成。
- 您无法删除已有集成的数据库集群。您必须先删除所有关联的集成。
- 如果您停止源数据库集群，则在恢复集群之前，可能不会将最后几个事务复制到目标。
- 如果您的集群是蓝绿部署的源，则在切换期间，蓝色和绿色环境无法具有现有的零 ETL 集成。您必须先删除集成，接着切换，然后重新创建集成。
- 一个数据库集群必须至少包含一个数据库实例才能成为集成的来源。
- 您无法为本身是跨账户克隆的源数据库集群创建集成，例如使用 AWS Resource Access Manager（AWS RAM）共享的集群。
- 如果源集群是 Aurora 全局数据库中的主数据库集群，并且它故障转移到其中的一个辅助集群，则集成将变为非活动状态。您必须删除并重新创建集成。
- 如果源数据库正在积极创建另一个集成，则无法为其创建集成。
- 最初创建集成或重新同步表时，从源到目标的数据做种可能需要 20-25 分钟或更长时间，具体取决于源数据库的大小。这种延迟可能导致副本滞后延长。
- 某些数据类型不支持。有关更多信息，请参阅 the section called “数据类型差异”。
- 系统表、临时表和视图不会复制到目标仓库。
- `ALTER TABLE` 分区操作会导致表重新同步，以便将数据从 Aurora 重新加载到分析目标。该表在重新同步时将不可用于查询。有关更多信息，请参阅 the section called “我的一个或多个 Amazon Redshift 表需要重新同步”。

Aurora MySQL 限制

- 源数据库集群必须运行受支持的 Aurora MySQL 版本。有关受支持的版本的列表，请参阅 the section called “零 ETL 集成 ”。
- 零 ETL 集成依赖于 MySQL 二进制日志（binlog）来捕获持续数据更改。请勿使用基于二进制日志的数据筛选，因为这可能会导致源数据库和目标数据库之间的数据不一致。
- 零 ETL 集成仅适用于配置为使用 InnoDB 存储引擎的数据库。
- 不支持带有预定义表更新的外键引用。具体而言，`CASCADE`、`SET NULL` 和 `SET DEFAULT` 操作不支持 `ON DELETE` 和 `ON UPDATE` 规则。尝试使用对另一个表的此类引用创建或更新表会使该表进入失败状态。
- 在源数据库集群上执行的 XA 事务会导致集成进入 `Syncing` 状态。

Aurora PostgreSQL 限制

- 源数据库集群必须运行受支持的 Aurora PostgreSQL 版本。有关受支持的版本的列表，请参阅 the section called “零 ETL 集成 ”。
- 如果选择 Aurora PostgreSQL 源数据库集群，则必须至少指定一种数据筛选条件模式。该模式必须至少包含一个数据库 (*`database-name`*`.*.*`)，才能复制到目标仓库。有关更多信息，请参阅 the section called “零 ETL 集成的数据筛选”。
- 在源 Aurora PostgreSQL 数据库集群中创建的所有数据库都必须使用 UTF-8 编码。
- 如果您在源数据库集群上执行 declarative partitioning 事务，则所有受影响的表都将进入失败状态，并且不再可供访问。
- 不支持两阶段事务。
- 如果从作为集成源的数据库集群中删除所有数据库实例，然后重新添加数据库实例，则源集群和目标集群之间的复制会中断。
- 源数据库集群无法使用 Aurora Limitless Database。

Amazon Redshift 限制

有关与零 ETL 集成相关的 Amazon Redshift 限制的列表，请参阅《Amazon Redshift 管理指南》中的将零 ETL 集成与 Amazon Redshift 结合使用时的注意事项。

Amazon SageMaker 智能湖仓限制

以下是 Amazon SageMaker 智能湖仓零 ETL 集成的限制。

- 目录名称的长度限制为 19 个字符。

限额

您的账户有以下与 Aurora 零 ETL 集成相关的限额。除非另行指定，否则每个限额将基于区域。

名称	默认值	描述
集成	100	AWS 账户内的集成总数。
每个目标的集成数	50	向单个目标数据仓库或智能湖仓发送数据的集成数量。
每个源集群的集成数量	5	从单个源数据库集群发送数据的集成数量。

此外，目标仓库对每个数据库实例或集群节点中支持的表数量设置了某些限制。有关 Amazon Redshift 配额和限制的更多信息，请参阅《Amazon Redshift 管理指南》中的 Amazon Redshift 中的配额和限制。

支持的区域

Aurora 零 ETL 集成在 AWS 区域的一个子集中提供。有关受支持的 区域的列表，请参阅the section called “零 ETL 集成 ”。

开始使用 Aurora 与 Amazon Redshift 的零 ETL 集成

在创建与 Amazon Redshift 的零 ETL 集成之前，请使用所需的参数和权限配置您的 Aurora 数据库集群和 Amazon Redshift 数据仓库。在安装过程中，您将完成以下步骤：

1. 创建自定义数据库集群参数组。
2. 创建源数据库集群。
3. 创建目标 Amazon Redshift 数据仓库。

完成这些步骤后，请继续执行the section called “创建与 Amazon Redshift 的零 ETL 集成”。

您可以使用 AWS SDK 为您自动完成设置过程。有关更多信息，请参阅 the section called “使用 AWS SDK 设置集成（仅限 Aurora MySQL）”。

步骤 1：创建自定义数据库集群参数组

Aurora 与 Amazon Redshift 的零 ETL 集成需要为控制复制的数据库集群参数提供特定值。具体而言，Aurora MySQL 需要增强型二进制日志（aurora_enhanced_binlog），而 Aurora PostgreSQL 需要增强型逻辑复制（aurora.enhanced_logical_replication）。

要配置二进制日志记录或逻辑复制，必须先创建自定义数据库集群参数组，然后将其与源数据库集群关联。

根据您的源数据库引擎，使用以下设置创建自定义数据库集群参数组。有关创建参数组的说明，请参阅the section called “数据库集群参数组”。

Aurora MySQL（aurora-mysql8.0 系列）：

- aurora_enhanced_binlog=1
- binlog_backup=0
- binlog_format=ROW
- binlog_replication_globaldb=0
- binlog_row_image=full
- binlog_row_metadata=full

此外，请确保 binlog_transaction_compression 参数未设置为 ON，也未将 binlog_row_value_options 参数设置为 PARTIAL_JSON。

有关 Aurora MySQL 增强型二进制日志的更多信息，请参阅the section called “设置增强型二进制日志”。

Aurora PostgreSQL（aurora-postgresql15 系列）：

> **Note**
>
> 对于 Aurora PostgreSQL 数据库集群，您必须在美国东部（俄亥俄州）（us-east-2）AWS 区域的 Amazon RDS 数据库预览环境中创建自定义参数组。

- rds.logical_replication=1

- aurora.enhanced_logical_replication=1
- aurora.logical_replication_backup=0
- aurora.logical_replication_globaldb=0

启用增强型逻辑复制（aurora.enhanced_logical_replication）会自动将 REPLICA IDENTITY 参数设置为 FULL，这意味着所有列值都写入提前写入日志（WAL）。这将增加源数据库集群的 IOPS。

步骤 2：选择或创建源数据库集群

创建自定义数据库集群参数组后，选择或创建一个 Aurora MySQL 或 Aurora PostgreSQL 数据库集群。该集群将成为向 Amazon Redshift 复制数据的源。

该集群必须运行 Aurora MySQL 版本 3.05（与 MySQL 8.0.32 兼容）或更高版本，或者 Aurora PostgreSQL（兼容 PostgreSQL 15.4 和零 ETL 支持）。有关创建数据库集群的说明，请参阅。

> **Note**
>
> 您必须在美国东部（俄亥俄州）（us-east-2）AWS 区域的 Amazon RDS 数据库预览环境中创建 Aurora PostgreSQL 数据库集群。

在其它配置下，将默认的数据库集群参数组更改为您在上一步中创建的自定义参数组。

> **Note**
>
> 对于 Aurora MySQL，您在已创建集群之后 将参数组与数据库集群关联，则必须重启集群中的主数据库实例以应用更改，然后才能创建零 ETL 集成。有关说明，请参阅 the section called “重启 Aurora 数据库集群或实例”。
>
> 在 Aurora PostgreSQL 与 Amazon Redshift 的零 ETL 集成预览版期间，您必须在创建集群时将集群与自定义数据库集群参数组相关联。在源数据库集群已经创建后，您不能执行此操作，否则您需要删除并重新创建集群。

步骤 3：创建目标 Amazon Redshift 数据仓库

创建源数据库集群后，必须在 Amazon Redshift 中创建和配置目标数据仓库。数据仓库必须满足以下要求：

- 在预览版中创建（仅适用于 Aurora PostgreSQL 源）。对于 Aurora MySQL 源，您必须创建生产集群和工作组。
  - 要在预览版中创建预调配集群，请从预调配集群控制面板上的横幅中选择创建预览版集群。有关更多信息，请参阅创建预览版集群。



    创建集群时，将预览版跟踪设置为 `preview_2023`。
  - 要在预览版中创建 Redshift Serverless 工作组，请从无服务器控制面板的横幅中选择创建预览版工作组。有关更多信息，请参阅创建预览版工作组。



- 使用 RA3 节点类型（`ra3.xlplus`、`ra3.4xlarge` 或 `ra3.16xlarge`）或 Redshift Serverless。
- 已加密（如果使用预置集群）。有关更多信息，请参阅 Amazon Redshift 数据库加密。

有关创建数据仓库的说明，请参阅预调配集群的创建集群或 Redshift Serverless 的创建带命名空间的工作组。

在数据仓库上启用区分大小写

要使集成成功，必须为数据仓库启用区分大小写参数（`enable_case_sensitive_identifier`）。默认情况下，所有预调配集群和 Redshift Serverless 工作组均禁用区分大小写。

要启用区分大小写，请根据您的数据仓库类型执行以下步骤：

- 预调配集群 – 要在预调配集群上启用区分大小写，请创建一个启用 `enable_case_sensitive_identifier` 参数的自定义参数组。然后，将该参数组与集群关联。有关说明，请参阅使用控制台管理参数组或使用 AWS CLI 配置参数值。

> **Note**
>
> 将自定义参数组与集群关联后，请记得重启集群。

- 无服务器工作组 - 要在 Redshift Serverless 工作组上启用区分大小写，必须使用 AWS CLI。Amazon Redshift 控制台目前不支持修改 Redshift Serverless 参数值。发送以下 update-workgroup 请求：

```
aws redshift-serverless update-workgroup \
  --workgroup-name target-workgroup \
  --config-parameters
 parameterKey=enable_case_sensitive_identifier,parameterValue=true
```

  修改参数值后，无需重启工作组。

为数据仓库配置授权

创建数据仓库后，必须将源 Aurora 数据库集群配置为授权的集成源。有关说明，请参阅为您的 Amazon Redshift 数据仓库配置授权。

使用 AWS SDK 设置集成（仅限 Aurora MySQL）

您可以运行以下 Python 脚本来自动设置所需的资源，而不必手动设置每个资源。代码示例使用 适用于 Python (Boto3) 的 AWS SDK 创建源 Aurora MySQL 数据库集群和目标 Amazon Redshift 数据仓库，其中每个都具有所需的参数值。然后，它会等待集群可用后，再在它们之间创建零 ETL 集成。您可以根据需要设置的资源注释掉不同的函数。

要安装所需依赖项，请运行以下命令：

```
pip install boto3
pip install time
```

在脚本中，可以选择修改源组、目标组和参数组的名称。最后一个函数在设置资源后创建一个名为 `my-integration` 的集成。

Python 代码示例

```
import boto3
import time

# Build the client using the default credential configuration.
# You can use the CLI and run 'aws configure' to set access key, secret
# key, and default Region.
```

```
rds = boto3.client('rds')
redshift = boto3.client('redshift')
sts = boto3.client('sts')

source_cluster_name = 'my-source-cluster' # A name for the source cluster
source_param_group_name = 'my-source-param-group' # A name for the source parameter
 group
target_cluster_name = 'my-target-cluster' # A name for the target cluster
target_param_group_name = 'my-target-param-group' # A name for the target parameter
 group

def create_source_cluster(*args):
    """Creates a source Aurora MySQL DB cluster"""

    response = rds.create_db_cluster_parameter_group(
        DBClusterParameterGroupName=source_param_group_name,
        DBParameterGroupFamily='aurora-mysql8.0',
        Description='For Aurora MySQL zero-ETL integrations'
    )
    print('Created source parameter group: ' + response['DBClusterParameterGroup']
['DBClusterParameterGroupName'])

    response = rds.modify_db_cluster_parameter_group(
        DBClusterParameterGroupName=source_param_group_name,
        Parameters=[
            {
                'ParameterName': 'aurora_enhanced_binlog',
                'ParameterValue': '1',
                'ApplyMethod': 'pending-reboot'
            },
            {
                'ParameterName': 'binlog_backup',
                'ParameterValue': '0',
                'ApplyMethod': 'pending-reboot'
            },
            {
                'ParameterName': 'binlog_format',
                'ParameterValue': 'ROW',
                'ApplyMethod': 'pending-reboot'
            },
            {
                'ParameterName': 'binlog_replication_globaldb',
                'ParameterValue': '0',
                'ApplyMethod': 'pending-reboot'
```

```
            },
            {
                'ParameterName': 'binlog_row_image',
                'ParameterValue': 'full',
                'ApplyMethod': 'pending-reboot'
            },
            {
                'ParameterName': 'binlog_row_metadata',
                'ParameterValue': 'full',
                'ApplyMethod': 'pending-reboot'
            }
        ]
    )
    print('Modified source parameter group: ' +
 response['DBClusterParameterGroupName'])

    response = rds.create_db_cluster(
        DBClusterIdentifier=source_cluster_name,
        DBClusterParameterGroupName=source_param_group_name,
        Engine='aurora-mysql',
        EngineVersion='8.0.mysql_aurora.3.05.2',
        DatabaseName='myauroradb',
        MasterUsername='username',
        MasterUserPassword='Password01**'
    )
    print('Creating source cluster: ' + response['DBCluster']['DBClusterIdentifier'])
    source_arn = (response['DBCluster']['DBClusterArn'])
    create_target_cluster(target_cluster_name, source_arn, target_param_group_name)

    response = rds.create_db_instance(
        DBInstanceClass='db.r6g.2xlarge',
        DBClusterIdentifier=source_cluster_name,
        DBInstanceIdentifier=source_cluster_name + '-instance',
        Engine='aurora-mysql'
    )
    return(response)

def create_target_cluster(target_cluster_name, source_arn, target_param_group_name):
    """Creates a target Redshift cluster"""

    response = redshift.create_cluster_parameter_group(
        ParameterGroupName=target_param_group_name,
        ParameterGroupFamily='redshift-1.0',
        Description='For Aurora MySQL zero-ETL integrations'
```

```
    )
    print('Created target parameter group: ' + response['ClusterParameterGroup']
['ParameterGroupName'])

    response = redshift.modify_cluster_parameter_group(
        ParameterGroupName=target_param_group_name,
        Parameters=[
            {
                'ParameterName': 'enable_case_sensitive_identifier',
                'ParameterValue': 'true'
            }
        ]
    )
    print('Modified target parameter group: ' + response['ParameterGroupName'])

    response = redshift.create_cluster(
        ClusterIdentifier=target_cluster_name,
        NodeType='ra3.4xlarge',
        NumberOfNodes=2,
        Encrypted=True,
        MasterUsername='username',
        MasterUserPassword='Password01**',
        ClusterParameterGroupName=target_param_group_name
    )
    print('Creating target cluster: ' + response['Cluster']['ClusterIdentifier'])

    # Retrieve the target cluster ARN
    response = redshift.describe_clusters(
        ClusterIdentifier=target_cluster_name
    )
    target_arn = response['Clusters'][0]['ClusterNamespaceArn']

    # Retrieve the current user's account ID
    response = sts.get_caller_identity()
    account_id = response['Account']

    # Create a resource policy specifying cluster ARN and account ID
    response = redshift.put_resource_policy(
        ResourceArn=target_arn,
        Policy='''
        {
            \"Version\":\"2012-10-17\",
            \"Statement\":[
                {\"Effect\":\"Allow\",
```

```
                \"Principal\":{
                    \"Service\":\"redshift.amazonaws.com\"
                },
                \"Action\":[\"redshift:AuthorizeInboundIntegration\"],
                \"Condition\":{
                    \"StringEquals\":{
                        \"aws:SourceArn\":\"%s\"}
                    }
                },
                {\"Effect\":\"Allow\",
                \"Principal\":{
                    \"AWS\":\"arn:aws:iam::%s:root\"},
                \"Action\":\"redshift:CreateInboundIntegration\"}
            ]
        }
        ''' % (source_arn, account_id)
    )
    return(response)

def wait_for_cluster_availability(*args):
    """Waits for both clusters to be available"""

    print('Waiting for clusters to be available...')

    response = rds.describe_db_clusters(
        DBClusterIdentifier=source_cluster_name
    )
    source_status = response['DBClusters'][0]['Status']
    source_arn = response['DBClusters'][0]['DBClusterArn']

    response = rds.describe_db_instances(
        DBInstanceIdentifier=source_cluster_name + '-instance'
    )
    source_instance_status = response['DBInstances'][0]['DBInstanceStatus']

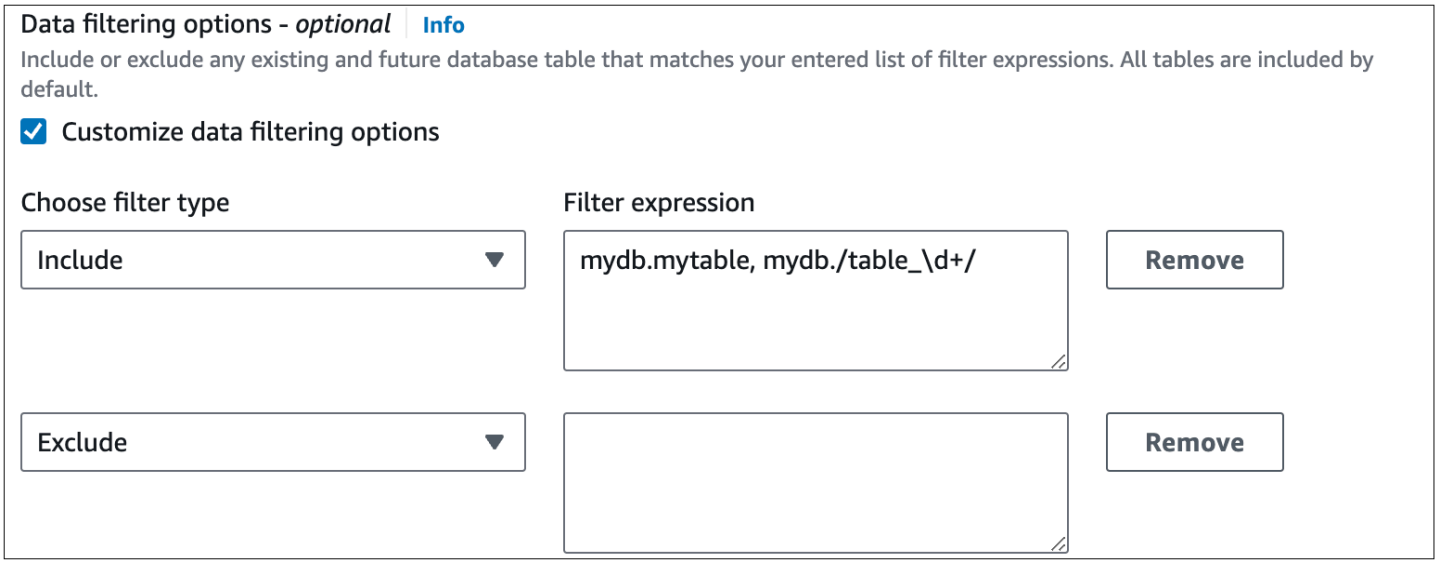
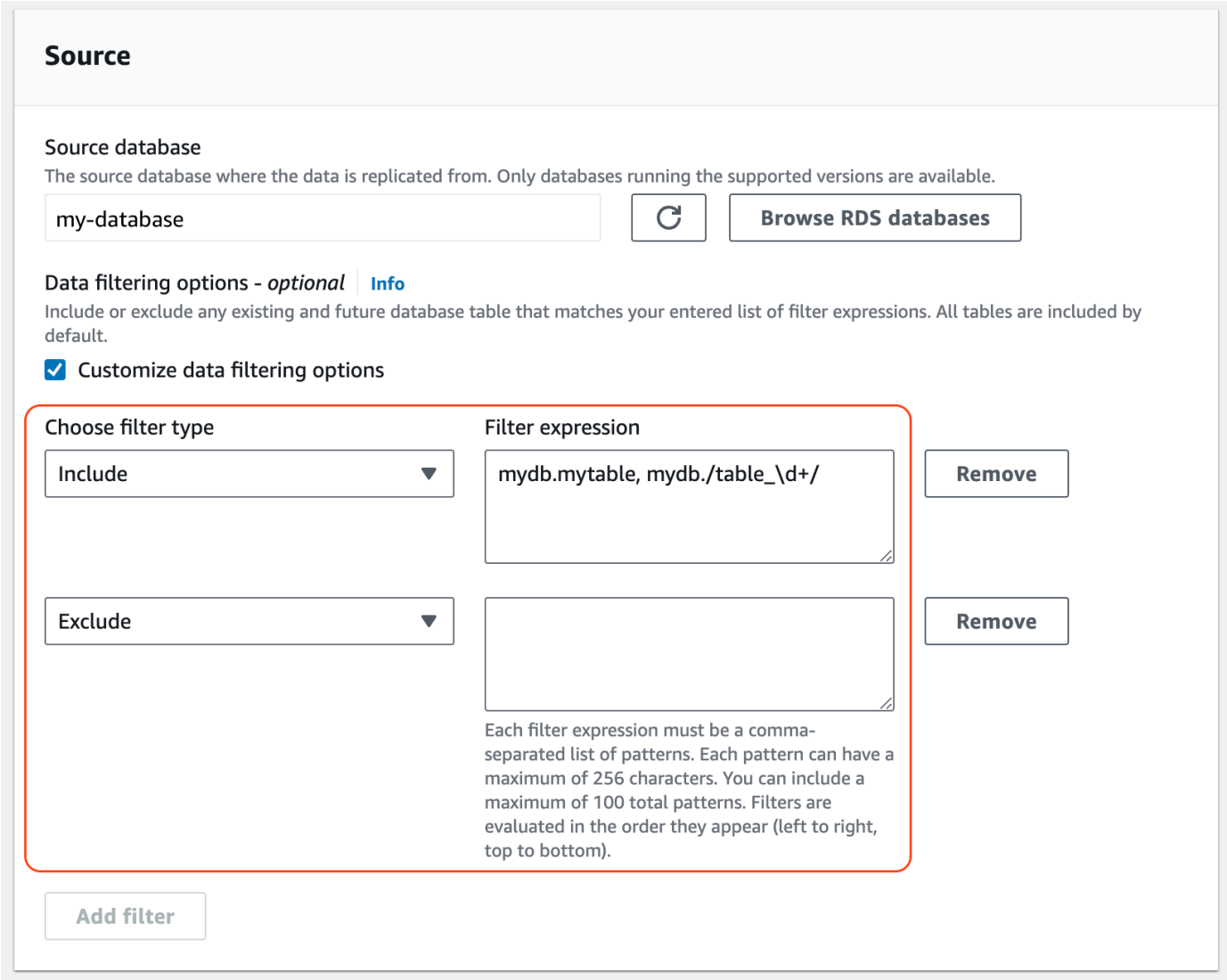
    response = redshift.describe_clusters(
        ClusterIdentifier=target_cluster_name
    )
    target_status = response['Clusters'][0]['ClusterStatus']
    target_arn = response['Clusters'][0]['ClusterNamespaceArn']

    # Every 60 seconds, check whether the clusters are available.
    if source_status != 'available' or target_status != 'available' or
 source_instance_status != 'available':
```

```
        time.sleep(60)
        response = wait_for_cluster_availability(
            source_cluster_name, target_cluster_name)
    else:
        print('Clusters available. Ready to create zero-ETL integration.')
        create_integration(source_arn, target_arn)
        return

def create_integration(source_arn, target_arn):
    """Creates a zero-ETL integration using the source and target clusters"""

    response = rds.create_integration(
        SourceArn=source_arn,
        TargetArn=target_arn,
        IntegrationName='my-integration'
    )
    print('Creating integration: ' + response['IntegrationName'])

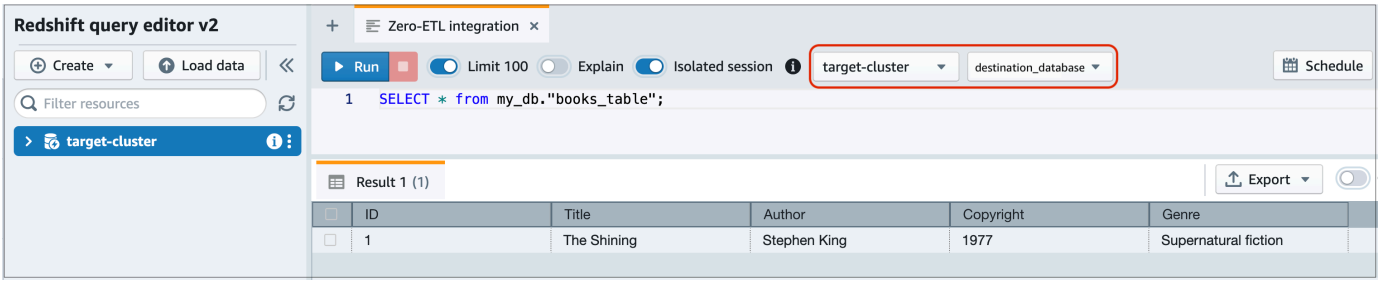
def main():
    """main function"""
    create_source_cluster(source_cluster_name, source_param_group_name)
    wait_for_cluster_availability(source_cluster_name, target_cluster_name)

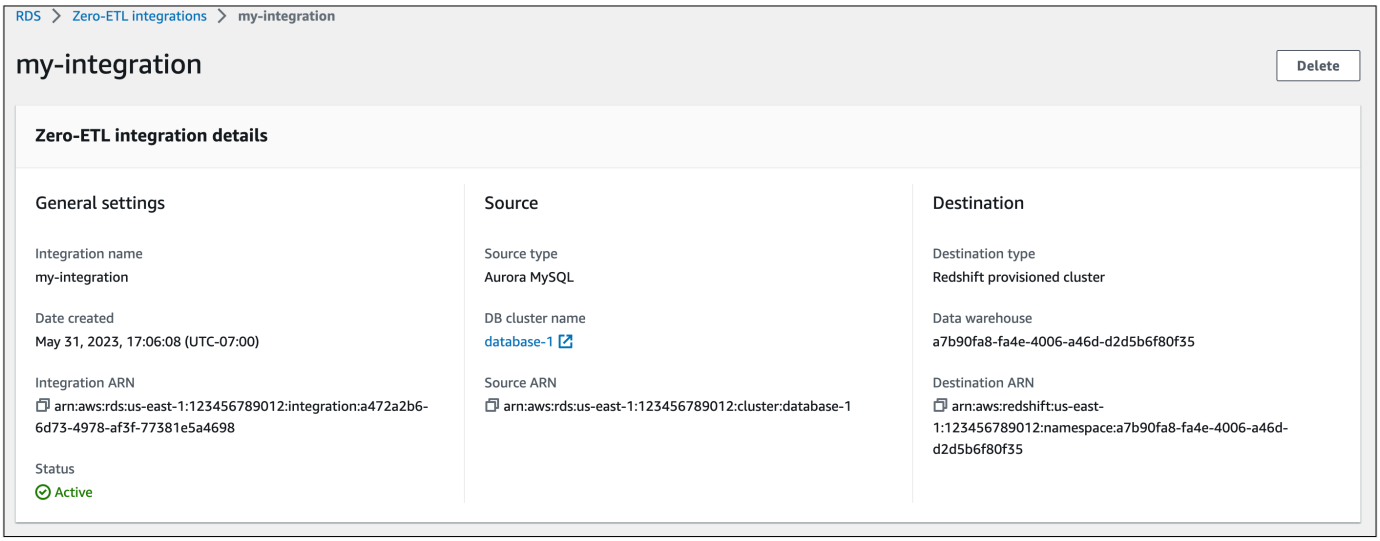
if __name__ == "__main__":
    main()
```

后续步骤

借助源 Aurora 数据库集群和 Amazon Redshift 目标数据仓库，您可以创建零 ETL 集成并复制数据。有关说明，请参阅 the section called “创建与 Amazon Redshift 的零 ETL 集成”。

创建 Aurora 与 Amazon Redshift 的零 ETL 集成

在创建 Aurora 零 ETL 集成时，需要指定源 Aurora 数据库集群和目标 Amazon Redshift 数据仓库。您还可以自定义加密设置和添加标签。Aurora 在源数据库集群与其目标之间创建集成。集成激活后，您插入到源数据库集群中的任何数据都将复制到配置的 Amazon Redshift 目标中。

先决条件

在创建零 ETL 集成之前，必须创建源数据库集群和目标 Amazon Redshift 数据仓库。您还必须通过将数据库集群添加为授权集成源来允许复制到数据仓库。

有关完成其中每个步骤的说明，请参阅the section called “开始使用零 ETL 集成”。

所需的权限

创建零 ETL 集成需要具有某些 IAM 权限。至少您需要具有执行以下操作的权限：

- 为源 Aurora 数据库集群创建零 ETL 集成。
- 查看和删除所有零 ETL 集成。
- 在目标数据仓库中创建入站集成。

以下示例策略演示了创建和管理集成所需的最低权限。如果您的用户或角色具有更广泛的权限（例如 AdministratorAccess 托管式策略），则可能不需要这些确切的权限。

> **Note**
>
> Redshift Amazon 资源名称（ARN）采用以下格式。请注意，在无服务器命名空间 UUID 之前使用了正斜杠（/）而不是冒号（:）。
>
> - 预调配集群 – arn:aws:**redshift**:{region}:{account-id}:namespace:*namespace-uuid*
> - 无服务器 – arn:aws:**redshift-serverless**:{region}:{account-id}:namespace/*namespace-uuid*

Redshift 目标的示例策略

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": [
            "rds:CreateIntegration"
        ],
        "Resource": [
            "arn:aws:rds:{region}:{account-id}:cluster:source-db",
            "arn:aws:rds:{region}:{account-id}:integration:*"
```

```
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "rds:DescribeIntegrations"
            ],
            "Resource": ["*"]
        },
        {
            "Effect": "Allow",
            "Action": [
                "rds:DeleteIntegration",
                "rds:ModifyIntegration"
            ],
            "Resource": [
                "arn:aws:rds:{region}:{account-id}:integration:*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "redshift:CreateInboundIntegration"
            ],
            "Resource": [
                "arn:aws:redshift:{region}:{account-id}:namespace:namespace-uuid"
            ]
        }]
}
```

在不同的账户中选择目标数据仓库

如果您计划指定位于另一个 AWS 账户中的目标 Amazon Redshift 数据仓库，则必须创建一个角色，以允许当前账户中的用户访问目标账户中的资源。有关更多信息，请参阅在您拥有的其他 AWS 账户中向 IAM 用户提供访问权限。

该角色必须具有以下权限，这些权限允许用户查看目标账户中可用的 Amazon Redshift 预调配集群和 Redshift Serverless 命名空间。

必需的权限和信任策略

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "redshift:DescribeClusters",
            "redshift-serverless:ListNamespaces"
         ],
         "Resource":[
            "*"
         ]
      }
   ]
}
```

该角色必须具有以下信任策略，该策略指定目标账户 ID。

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Principal":{
            "AWS": "arn:aws:iam::{external-account-id}:root"
         },
         "Action":"sts:AssumeRole"
      }
   ]
}
```

有关创建角色的说明，请参阅使用自定义信任策略创建角色。

创建零 ETL 集成

您可以使用 AWS Management Console、AWS CLI 或 RDS API 创建零 ETL 集成。

> **Important**
>
> 零 ETL 集成不支持刷新或重新同步操作。如果在创建集成后遇到与集成相关的问题，必须删除此集成并创建一个新集成。

RDS 控制台

创建零 ETL 集成

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在左侧导航窗格中，选择零 ETL 集成。
3. 选择创建零 ETL 集成。
4. 在集成标识符中，输入集成的名称。该名称可包含最多 63 个字母数字字符，并且可以包含连字符。

   > **Important**
   >
   > 目录名称的长度限制为 19 个字符。如果要将集成标识符用作目录名称，请确保它符合此要求。

5. 选择下一步。
6. 对于源，选择数据将源自其中的 Aurora 数据库集群。

   > **Note**
   >
   > 如果数据库集群参数配置不正确，RDS 会通知您。如果您收到此消息，可以选择为我修复，也可以手动配置它们。有关手动修复它们的说明，请参阅the section called “步骤 1：创建自定义数据库集群参数组”。
   > 修改数据库集群参数需要重启。在创建集成之前，必须完成重启，并且必须成功地将新的参数值应用于集群。

7. （可选）选择自定义数据筛选选项并向您的集成添加数据筛选条件。您可以使用数据筛选条件来定义复制到目标数据仓库的范围。有关更多信息，请参阅 the section called “零 ETL 集成的数据筛选”。
8. 成功配置源数据库集群后，选择下一步。
9. 对于目标，执行以下操作：

   1. （可选）要为 Amazon Redshift 目标使用不同的 AWS 账户，请选择指定其他账户。然后，输入有权显示您数据仓库的 IAM 角色的 ARN。有关创建 IAM 角色的说明，请参阅the section called “在不同的账户中选择目标数据仓库”。
   2. 对于 Amazon Redshift 数据仓库，选择从源数据库集群中复制的数据的目标。您可以选择预调配的 Amazon Redshift 集群或 Redshift Serverless 命名空间作为目标。

   > **Note**
   >
   > 如果指定数据仓库的资源策略或区分大小写设置配置不正确，RDS 会通知您。如果您收到此消息，可以选择为我修复，也可以手动配置它们。有关手动修复这些问题的说明，请参阅《Amazon Redshift 管理指南》中的为您的数据仓库开启区分大小写和为您的数据仓库配置授权。
   > 修改预调配 Redshift 集群的区分大小写需要重启。在创建集成之前，必须完成重启，并且必须成功地将新的参数值应用于集群。
   > 如果您选择的源和目标位于不同的 AWS 账户，则 Amazon RDS 无法为您修复这些设置。您必须导航到另一个账户，然后在 Amazon Redshift 中手动修复这些问题。

10. 正确配置目标数据仓库后，选择下一步。
11. （可选）对于标签，向集成添加一个或多个标签。有关更多信息，请参阅 the section called “为 Aurora 和 RDS 资源添加标签”。
12. 对于加密，请指定您希望如何加密集成。默认情况下，RDS 会加密所有与 AWS 拥有的密钥 的集成。要改为选择客户自主管理型密钥，请启用自定义加密设置并选择用于加密的 KMS 密钥。有关更多信息，请参阅 the section called “加密 Amazon Aurora 资源”。

    （可选）添加加密上下文。有关更多信息，请参阅 AWS Key Management Service 开发人员指南中的加密内容。

    > **Note**
    >
    > 除了您添加的任何加密上下文对外，Amazon RDS 还会添加以下加密上下文对：

- `aws:redshift:integration:arn` - `IntegrationArn`
- `aws:servicename:id` - `Redshift`

这会将您可以添加的加密上下文对总数从 8 减少到 6，并增加授予约束条件的总字符限制。有关更多信息，请参阅《AWS Key Management Service 开发人员指南》中的 Using grant constraints。

13. 选择下一步。
14. 查看您的集成设置并选择创建零 ETL 集成。

    如果创建失败，请参阅the section called “我无法创建零 ETL 集成”以了解故障排除步骤。

集成在创建时状态为 `Creating`，而目标 Amazon Redshift 数据仓库的状态为 `Modifying`。在此期间，您无法查询数据仓库或对其进行任何配置更改。

成功创建集成后，集成和目标 Amazon Redshift 数据仓库的状态都更改为 `Active`。

AWS CLI

要使用 AWS CLI 创建零 ETL 集成，请使用带有以下选项的 create-integration 命令：

> **Note**
>
> 请记住，目录名称限制为 19 个字符。如果要将集成名称用作目录名称，请相应地选择集成名称。

- `--integration-name` – 指定集成的名称。
- `--source-arn` – 指定将作为集成源的 Aurora 数据库集群的 ARN。
- `--target-arn` – 指定将作为集成目标的 Amazon Redshift 数据仓库的 ARN。

Example

对于 Linux、macOS 或 Unix：

```
aws rds create-integration \
    --integration-name my-integration \
```

```
    --source-arn arn:aws:rds:{region}:{account-id}:my-db \
    --target-arn arn:aws:redshift:{region}:{account-id}:namespace:namespace-uuid
```

对于 Windows：

```
aws rds create-integration ^
    --integration-name my-integration ^
    --source-arn arn:aws:rds:{region}:{account-id}:my-db ^
    --target-arn arn:aws:redshift:{region}:{account-id}:namespace:namespace-uuid
```

RDS API

要使用 Amazon RDS API 创建零 ETL 集成，请结合以下参数使用 CreateIntegration 操作：

> **Note**
>
> 目录名称限制为 19 个字符。如果要将 IntegrationName 参数用作目录名称，请确保它符合此要求。

- `IntegrationName` – 指定集成的名称。
- `SourceArn` – 指定将作为集成源的 Aurora 数据库集群的 ARN。
- `TargetArn` – 指定将作为集成目标的 Amazon Redshift 数据仓库的 ARN。

使用客户自主管理型密钥加密集成

如果您指定自定义 KMS 密钥而不是创建集成时的 AWS 拥有的密钥，则密钥策略必须为 Amazon Redshift 服务主体提供对 `CreateGrant` 操作的访问权限。此外，它必须允许当前用户执行 `DescribeKey` 和 `CreateGrant` 操作。

以下示例策略演示了如何提供密钥策略中所需的权限。它包括用于进一步缩小权限范围的上下文键。

示例密钥策略

JSON

```
{
    "Version": "2012-10-17",
    "Id": "Key policy",
```

```
    "Statement": [
        {
            "Sid": "Enables IAM user permissions",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::{account-ID}:root"
            },
            "Action": "kms:*",
            "Resource": "*"
        },
        {
            "Sid": "Allows the Redshift service principal to add a grant to a KMS
key",
            "Effect": "Allow",
            "Principal": {
                "Service": "redshift.amazonaws.com"
            },
            "Action": "kms:CreateGrant",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "kms:EncryptionContext:{context-key}":"{context-value}"
                },
                "ForAllValues:StringEquals": {
                    "kms:GrantOperations": [
                        "Decrypt",
                        "GenerateDataKey",
                        "CreateGrant"
                    ]
                }
            }
        },
        {
            "Sid": "Allows the current user or role to add a grant to a KMS key",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::{account-ID}:role/{role-name}"
            },
            "Action": "kms:CreateGrant",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "kms:EncryptionContext:{context-key}":"{context-value}",
                    "kms:ViaService": "rds.us-east-1.amazonaws.com"
```

```
                },
                "ForAllValues:StringEquals": {
                    "kms:GrantOperations": [
                        "Decrypt",
                        "GenerateDataKey",
                        "CreateGrant"
                    ]
                }
            }
        },
        {
            "Sid": "Allows the current uer or role to retrieve information about
 a KMS key",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::{account-ID}:role/{role-name}"
            },
            "Action": "kms:DescribeKey",
            "Resource": "*"
        }
    ]
}
```

有关更多信息，请参阅《AWS Key Management Service 开发人员指南》中的创建密钥策略。

后续步骤

成功创建零 ETL 集成后，您必须在目标 Amazon Redshift 集群或工作组中创建目标数据库。然后，您可以开始向源 Aurora 数据库集群中添加数据，并在 Amazon Redshift 中对其进行查询。有关说明，请参阅在 Amazon Redshift 中创建目标数据库。

创建 Aurora 与 Amazon SageMaker 智能湖仓的零 ETL 集成

在创建 Aurora 与 Amazon SageMaker 智能湖仓的零 ETL 集成时，您应指定源 Aurora 数据库集群和目标 AWS Glue 托管目录。您还可以自定义加密设置和添加标签。Aurora 在源数据库集群与其目标之间创建集成。在集成处于活动状态后，您插入到源数据库集群中的任何数据都将复制到配置的目标中。

先决条件

在创建与 Amazon SageMaker 智能湖仓的零 ETL 集成之前，必须创建源数据库集群和目标 AWS Glue 托管目录。您还必须通过将数据库集群添加为授权的集成源来支持复制到目录。

有关完成其中每个步骤的说明，请参阅the section called “开始使用零 ETL 集成”。

所需的权限

创建与 Amazon SageMaker 智能湖仓的零 ETL 集成需要具有某些 IAM 权限。至少您需要具有执行以下操作的权限：

- 为源 Aurora 数据库集群创建零 ETL 集成。
- 查看和删除所有零 ETL 集成。
- 创建到目标 AWS Glue 托管目录的入站集成。
- 访问 AWS Glue 托管目录使用的 Amazon S3 存储桶。
- 如果配置了自定义加密，则使用 AWS KMS 密钥进行加密。
- 将资源注册到 Lake Formation 中。
- 将资源策略放在 AWS Glue 托管目录上以对入站集成进行授权。

以下示例策略演示了创建和管理与 Amazon SageMaker 智能湖仓的集成所需的最低权限。如果您的用户或角色具有更广泛的权限（例如 `AdministratorAccess` 托管式策略），则可能不需要这些确切的权限。

此外，您必须在目标 AWS Glue 托管目录上配置资源策略以便对入站集成进行授权。使用以下 AWS CLI 命令来应用资源策略。

在目标目录上对入站集成进行授权的示例 AWS CLI 命令

```
aws glue put-resource-policy \
      --policy-in-json  '{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Principal": {
            "Service": "glue.amazonaws.com"
        },
        "Action": [
            "glue:AuthorizeInboundIntegration"
        ],
        "Resource": ["arn:aws:glue:region:account_id:catalog/catalog_name"],
        "Condition": {
            "StringEquals": {
                "aws:SourceArn": "arn:aws:rds:region:account_id:db:source_name"
```

```
            }
        }
    },
    {
        "Effect": "Allow",
        "Principal": {
            "AWS": "account_id"
        },
        "Action": ["glue:CreateInboundIntegration"],
        "Resource": ["arn:aws:glue:region:account_id:catalog/catalog_name"]
    }
    ]
}' \
      --region region
```

> **Note**
>
> Glue 目录 Amazon 资源名称（ARN）的格式如下：
>
> - Glue 目录：arn:aws:**glue**:{region}:{account-id}:catalog/*catalog-name*

选择不同账户中的目标 AWS Glue 托管目录

如果您计划指定位于另一个 AWS 账户中的目标 AWS Glue 托管目录，则必须创建一个角色，以支持当前账户中的用户访问目标账户中的资源。有关更多信息，请参阅[在您拥有的其他 AWS 账户中向 IAM 用户提供访问权限]。

该角色必须具有以下权限，这些权限支持用户查看目标账户中可用的 AWS Glue 目录。

必需的权限和信任策略

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "glue:GetCatalog"
         ],
         "Resource":[
            "*"
```

```
            ]
        }
    ]
}
```

该角色必须具有以下信任策略，该策略指定目标账户 ID。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Principal":{
            "AWS": "arn:aws:iam::{external-account-id}:root"
         },
         "Action":"sts:AssumeRole"
      }
   ]
}
```

有关创建角色的说明，请参阅使用自定义信任策略创建角色。

创建与 Amazon SageMaker 智能湖仓的零 ETL 集成

您可以使用 AWS Management Console、AWS CLI 或 RDS API 创建与 Amazon SageMaker 智能湖仓的零 ETL 集成。

> **Important**
>
> 与 Amazon SageMaker 智能湖仓的零 ETL 集成不支持刷新或重新同步操作。如果在创建集成后遇到与集成相关的问题，必须删除此集成并创建一个新集成。

RDS 控制台

创建与 Amazon SageMaker 智能湖仓的零 ETL 集成

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在左侧导航窗格中，选择零 ETL 集成。
3. 选择创建零 ETL 集成。

4. 在集成标识符中，输入集成的名称。该名称可包含最多 63 个字母数字字符，并且可以包含连字符。
5. 选择下一步。
6. 对于源，选择数据将源自其中的 Aurora 数据库集群。

> **Note**
>
> 如果数据库集群参数配置不正确，RDS 会通知您。如果您收到此消息，可以选择为我修复，也可以手动配置它们。有关手动修复它们的说明，请参阅the section called “步骤 1：创建自定义数据库集群参数组”。
> 修改数据库集群参数需要重启。在创建集成之前，必须完成重启，并且必须成功地将新的参数值应用于集群。

7. （可选）选择自定义数据筛选选项并向您的集成添加数据筛选条件。您可以使用数据筛选条件来定义复制到目标 Amazon SageMaker 智能湖仓的范围。有关更多信息，请参阅 the section called “零 ETL 集成的数据筛选”。
8. 成功配置源数据库集群后，选择下一步。
9. 对于目标，执行以下操作：

    1. （可选）要为 Amazon SageMaker 智能湖仓目标使用不同的 AWS 账户，请选择指定不同的账户。然后，输入有权显示 AWS Glue 目录的 IAM 角色的 ARN。有关创建 IAM 角色的说明，请参阅the section called “选择不同账户中的目标 AWS Glue 托管目录”。
    2. 对于 AWS Glue 目录，选择从源数据库集群中复制的数据的目标。可以选择使用现有 AWS Glue 托管目录作为目标。
    3. 目标 IAM 角色需要对目标目录的描述权限，并且必须具有以下权限：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "glue:GetCatalog",
            "Resource": [
                "arn:aws:glue:region:account-id:catalog/*",
                "arn:aws:glue:region:account-id:catalog"
            ]
        }
```

```
    ]
}
```

目标 IAM 角色必须拥有以下信任关系：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "glue.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

4. 您必须使用在???中创建的 Lake Formation 管理员角色，来为目标 IAM 角色授予对目标 AWS Glue 托管目录的描述权限。

> **Note**
>
> 如果指定 AWS Glue 托管目录的资源策略或配置设置未得以正确配置，RDS 会通知您。
> 如果您收到此消息，可以选择为我修复，也可以手动配置它们。
> 如果您选择的源和目标位于不同的 AWS 账户，则 Amazon RDS 无法为您修复这些设置。
> 您必须导航到另一个账户，然后在 SageMaker Unified Studio 中手动修复这些设置。

10. 在正确配置了目标 AWS Glue 托管目录后，选择下一步。
11. （可选）对于标签，向集成添加一个或多个标签。有关更多信息，请参阅 the section called “为 Aurora 和 RDS 资源添加标签”。
12. 对于加密，请指定您希望如何加密集成。默认情况下，RDS 会加密所有与 AWS 拥有的密钥 的集成。要改为选择客户自主管理型密钥，请启用自定义加密设置并选择用于加密的 KMS 密钥。有关更多信息，请参阅 the section called “加密 Amazon Aurora 资源”。

    （可选）添加加密上下文。有关更多信息，请参阅 AWS Key Management Service 开发人员指南中的加密内容。

> **Note**
>
> 除了您添加的任何加密上下文对外，Amazon RDS 还会添加以下加密上下文对：
>
> - `aws:glue:integration:arn` - `IntegrationArn`
> - `aws:servicename:id` - `glue`
>
> 这会将您可以添加的加密上下文对总数从 8 减少到 6，并增加授予约束条件的总字符限制。有关更多信息，请参阅《AWS Key Management Service 开发人员指南》中的 Using grant constraints。

13. 选择下一步。
14. 查看您的集成设置并选择创建零 ETL 集成。

    如果创建失败，请参阅the section called “零 ETL 集成问题排查”以了解故障排除步骤。

集成在创建时状态为 `Creating`，而目标 Amazon SageMaker 智能湖仓的状态为 `Modifying`。在此期间，您无法查询目录或对其进行任何配置更改。

成功创建集成后，集成和目标 Amazon SageMaker 智能湖仓的状态都更改为 `Active`。

AWS CLI

要使用 AWS CLI 来准备要进行零 ETL 集成的目标 AWS Glue 托管目录，必须首先使用带有以下选项的 create-integration-resource-property 命令：

- `--resource-arn`：指定将成为集成目标的 AWS Glue 托管目录的 ARN。
- `--target-processing-properties`：指定 IAM 角色的 ARN 以访问目标 AWS Glue 托管目录

```
aws glue create-integration-resource-property --region us-east-1
 --resource-arn arn:aws:glue:region:account_id:catalog/catalog_name \
 --target-processing-properties '{"RoleArn" : "arn:aws:iam::account_id:role/
TargetIamRole"}'
```

要使用 AWS CLI 创建与 Amazon SageMaker 智能湖仓的零 ETL 集成，请使用带有以下选项的 create-integration 命令：

- `--integration-name` – 指定集成的名称。
- `--source-arn` – 指定将作为集成源的 Aurora 数据库集群的 ARN。
- `--target-arn`：指定将成为集成目标的 AWS Glue 托管目录的 ARN。

Example

对于 Linux、macOS 或 Unix：

```
aws rds create-integration \
    --integration-name my-sagemaker-integration \
    --source-arn arn:aws:rds:{region}:{account-id}:my-db \
    --target-arn arn:aws:glue:{region}:{account-id}:catalog/catalog-name
```

对于 Windows：

```
aws rds create-integration ^
    --integration-name my-sagemaker-integration ^
    --source-arn arn:aws:rds:{region}:{account-id}:my-db ^
    --target-arn arn:aws:glue:{region}:{account-id}:catalog/catalog-name
```

RDS API

要使用 Amazon RDS API 创建与 Amazon SageMaker 的零 ETL 集成，请结合使用 `CreateIntegration` 操作和以下参数：

> **Note**
>
> 目录名称限制为 19 个字符。如果要将 IntegrationName 参数用作目录名称，请确保它符合此要求。

- `IntegrationName` – 指定集成的名称。
- `SourceArn` – 指定将作为集成源的 Aurora 数据库集群的 ARN。
- `TargetArn`：指定将成为集成目标的 AWS Glue 托管目录的 ARN。

使用客户自主管理型密钥加密集成

如果您在创建与 Amazon SageMaker 的集成时指定自定义 KMS 密钥而不是 AWS 拥有的密钥，则密钥策略必须为 SageMaker Unified Studio 服务主体提供对 `CreateGrant` 操作的访问权限。此外，它必须允许当前用户执行 `DescribeKey` 和 `CreateGrant` 操作。

以下示例策略演示了如何提供密钥策略中所需的权限。它包括用于进一步缩小权限范围的上下文键。

示例密钥策略

```
{
    "Version": "2012-10-17",
    "Id": "Key policy",
    "Statement": [
        {
            "Sid": "Enables IAM user permissions",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::{account-ID}:root"
            },
            "Action": "kms:*",
            "Resource": "*"
        },
        {
            "Sid": "Allows the Glue service principal to add a grant to an AWS KMS
 key",
            "Effect": "Allow",
            "Principal": {
                "Service": "glue.amazonaws.com"
            },
            "Action": "kms:CreateGrant",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "kms:EncryptionContext:{context-key}":"{context-value}"
                },
                "ForAllValues:StringEquals": {
                    "kms:GrantOperations": [
                        "Decrypt",
                        "GenerateDataKey",
                        "CreateGrant"
                    ]
                }
```

```
            }
        },
        {
            "Sid": "Allows the current user or role to add a grant to a KMS key",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::{account-ID}:role/{role-name}"
            },
            "Action": "kms:CreateGrant",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "kms:EncryptionContext:{context-key}":"{context-value}",
                    "kms:ViaService": "rds.us-east-1.amazonaws.com"
                },
                "ForAllValues:StringEquals": {
                    "kms:GrantOperations": [
                        "Decrypt",
                        "GenerateDataKey",
                        "CreateGrant"
                    ]
                }
            }
        },
        {
            "Sid": "Allows the current uer or role to retrieve information about a KMS
 key",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::{account-ID}:role/{role-name}"
            },
            "Action": "kms:DescribeKey",
            "Resource": "*"
        }
    ]
}
```

有关更多信息，请参阅《AWS Key Management Service 开发人员指南》中的创建密钥策略。

后续步骤

在成功创建与 Amazon SageMaker 的零 ETL 集成后，您可以开始向源 Aurora 数据库集群添加数据，并在 Amazon SageMaker 智能湖仓中查询这些数据。数据将自动复制，并可供分析和机器学习工作负载使用。

Aurora 零 ETL 集成的数据筛选

Aurora 零 ETL 集成支持数据筛选，这使您可以控制哪些数据从源 Aurora 数据库集群复制到目标数据仓库。您可以应用一个或多个筛选条件来有选择地包含或排除特定的表，而不是复制整个数据库。这可以通过确保只传输相关数据，来协助您优化存储和查询性能。目前，筛选仅限于数据库级和表级。不支持列级和行级筛选。

当您要执行以下操作时，数据筛选可能很有用：

- 联接来自两个或更多不同源集群的某些表，而您不需要来自任一集群的完整数据。
- 仅使用表的子集而不是整个数据库实例集来执行分析，从而节省成本。
- 从某些表中筛选掉敏感信息，例如电话号码、地址或信用卡详细信息。

您可以使用 AWS Management Console、AWS Command Line Interface（AWS CLI）或 Amazon RDS API 向零 ETL 集成添加数据筛选条件。

如果集成将预置集群作为其目标，则该集群必须使用补丁 180 或更高版本，才能使用数据筛选。

主题

- 数据筛选条件的格式
- 筛选条件逻辑
- 筛选条件优先级
- Aurora MySQL 示例
- Aurora PostgreSQL 示例
- 向集成添加数据筛选条件
- 从集成中移除数据筛选条件

数据筛选条件的格式

您可以为单个集成定义多个筛选条件。每个筛选条件要么包含、要么排除任何与筛选表达式中的模式之一匹配的现有和将来的数据库表。Aurora 零 ETL 集成使用 Maxwell 筛选条件语法进行数据筛选。

每个筛选条件都包含以下元素：

元素	描述
筛选条件类型	`Include` 筛选条件类型包括与筛选表达式中的模式之一匹配的所有表。`Exclude` 筛选条件类型排除与模式之一匹配的所有表。
筛选表达式	逗号分隔的模式列表。表达式必须使用 Maxwell 筛选条件语法。
模式	一种筛选条件模式，格式为 *`database`*`.`*`table`*（对于 Aurora MySQL）或 *`database`*`.`*`schema`*`.`*`table`*（对于 Aurora PostgreSQL）。您可以指定文字名称，也可以定义正则表达式。 **Note** 对于 Aurora MySQL，数据库和表名称均支持正则表达式。对于 Aurora PostgreSQL，只有架构和表名称支持正则表达式，数据库名称不支持正则表达式。 您不能包含列级别筛选条件或黑名单。 单个集成最多可以有总共 99 个模式。在控制台中，可以在单个筛选表达式中输入模式，也可以将它们分散在多个表达式中。单个模式的长度不能超过 256 个字符。

> **Important**
>
> 如果选择 Aurora PostgreSQL 源数据库集群，则必须至少指定一种数据筛选条件模式。该模式必须至少包含一个数据库 (*database-name*.*.*)，才能复制到目标数据仓库。

下图显示了控制台中 Aurora MySQL 数据筛选条件的结构：



> **Important**
>
> 请勿在筛选条件模式中包含个人身份信息、机密信息或敏感信息。

AWS CLI 中的数据筛选条件

使用 AWS CLI 添加数据筛选条件时，语法与使用控制台略有不同。必须为每种模式分别指定筛选条件类型（`Include` 或 `Exclude`），因此不能将多个模式分组到一个筛选条件类型下。

例如，在控制台中，可以在单个 `Include` 语句下对以下逗号分隔的模式进行分组：

Aurora MySQL

```
mydb.mytable, mydb./table_\d+/
```

Aurora PostgreSQL

```
mydb.myschema.mytable, mydb.myschema./table_\d+/
```

但是，使用 AWS CLI 时，相同的数据筛选条件必须采用以下格式：

Aurora MySQL

```
'include: mydb.mytable, include: mydb./table_\d+/'
```

Aurora PostgreSQL

```
'include: mydb.myschema.mytable, include: mydb.myschema./table_\d+/'
```

筛选条件逻辑

如果您在集成中未指定任何数据筛选条件，Aurora 会采用默认筛选条件 `include:*.*`，这会将所有表复制到目标数据仓库。但是，如果您至少添加一个筛选条件，则默认逻辑会切换为 `exclude:*.*`，默认情况下这会排除所有表。这使您可以显式定义要在复制中包含哪些数据库和表。

例如，如果您定义以下筛选条件：

```
'include: db.table1, include: db.table2'
```

Aurora 按以下方式评估筛选条件：

```
'exclude:*.*, include: db.table1, include: db.table2'
```

因此，Aurora 仅将来自名为 db 的数据库的 `table1` 和 `table2` 复制到目标数据仓库。

筛选条件优先级

Aurora 按您指定的顺序评估数据筛选条件。在 AWS Management Console中，它将从左到右、从上到下处理筛选表达式。第二个筛选条件或第一个筛选条件之后的单个模式可以覆盖它。

例如，如果第一个筛选条件是 `Include books.stephenking`，则它只包括 books 数据库中的 `stephenking` 表。但是，如果您添加第二个筛选条件 `Exclude books.*`，则它会覆盖第一个筛选条件。这可以防止将 books 索引中的任何表复制到目标数据仓库。

当您指定至少一个筛选条件时，默认情况下，逻辑以采用 `exclude:*.*` 开始，这会自动将所有表排除 在复制范围之外。作为最佳实践，请按照从最广泛到最具体的顺序定义筛选条件。首先使用一个或多个 `Include` 语句来指定要复制的数据，然后添加 `Exclude` 筛选条件来有选择性地移除某些表。

同样的原则也适用于使用 AWS CLI 定义的筛选条件。Aurora 按您指定筛选条件模式的顺序评估这些模式，因此一个模式可能会覆盖您指定的在它之前的模式。

Aurora MySQL 示例

以下示例演示了如何将数据筛选用于 Aurora MySQL 零 ETL 集成：

- 包括所有数据库和所有表：

```
'include: *.*'
```

- 包括 books 数据库中的所有表：

```
'include: books.*'
```

- 排除任何名为 mystery 的表：

```
'include: *.*, exclude: *.mystery'
```

- 包括 books 数据库中的两个特定表：

```
'include: books.stephen_king, include: books.carolyn_keene'
```

- 包括 books 数据库中的所有表，但那些包含子字符串 mystery 的表除外：

```
'include: books.*, exclude: books./.*mystery.*/'
```

- 包括 books 数据库中的所有表，但那些以 mystery 开头的表除外：

```
'include: books.*, exclude: books./mystery.*/'
```

- 包括 books 数据库中的所有表，但那些以 mystery 结束的表除外：

```
'include: books.*, exclude: books./.*mystery/'
```

- 包括 books 数据库中的所有以 table_ 开头的表，但名为 table_stephen_king 的表除外。例如，将复制 table_movies 或 table_books，但不复制 table_stephen_king。

```
'include: books./table_.*/, exclude: books.table_stephen_king'
```

Aurora PostgreSQL 示例

以下示例演示了如何将数据筛选用于 Aurora PostgreSQL 零 ETL 集成：

- 包括 books 数据库中的所有表：

```
'include: books.*.*'
```

- 排除 books 数据库中名为 mystery 的所有表：

```
'include: books.*.*, exclude: books.*.mystery'
```

- 在 mystery 架构的 books 数据库中包括一个表，在 finance 架构的 employee 数据库中包括一个表：

```
'include: books.mystery.stephen_king, include: employee.finance.benefits'
```

- 包括 books 数据库和 science_fiction 架构中的所有表，但那些包含子字符串 king 的表除外：

```
'include: books.science_fiction.*, exclude: books.*./.*king.*/
```

- 包括 books 数据库中的所有表，但那些架构名称以 sci 开头的表除外：

```
'include: books.*.*, exclude: books./sci.*/.*'
```

- 包括 books 数据库中的所有表，但那些在 mystery 架构中以 king 结束的表除外：

```
'include: books.*.*, exclude: books.mystery./.*king/'
```

- 包括 books 数据库中的所有以 table_ 开头的表，但名为 table_stephen_king 的表除外。例如，复制 fiction 架构中的 table_movies 和 mystery 架构中的 table_books，但不复制任一架构中的 table_stephen_king：

```
'include: books.*./table_.*/, exclude: books.*.table_stephen_king'
```

向集成添加数据筛选条件

您可以使用 AWS Management Console、AWS CLI 或 Amazon RDS API 配置数据筛选。

> **Important**
>
> 如果您在创建集成后添加筛选条件，Aurora 会将其视为似乎始终存在。它会移除目标数据仓库中不符合新筛选条件的任何数据，并重新同步所有受影响的表。

RDS 控制台

将数据筛选条件添加到零 ETL 集成中

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择零 ETL 集成。选择要向其添加数据筛选条件的集成，然后选择修改。
3. 在源下，添加一个或多个 `Include` 和 `Exclude` 语句。

   下图显示了 MySQL 集成的数据筛选条件的示例：



4. 如果您对更改感到满意，请选择继续和保存更改。

AWS CLI

要使用 AWS CLI 向零 ETL 集成添加数据筛选条件，请调用 modify-integration 命令。除了集成标识符外，还要使用逗号分隔的 `Include` 和 `Exclude` Maxwell 筛选条件列表来指定 `--data-filter` 参数。

Example

以下示例向 `my-integration` 添加筛选条件模式。

对于 Linux、macOS 或 Unix：

```
aws rds modify-integration \
```

```
    --integration-identifier my-integration \
    --data-filter 'include: foodb.*, exclude: foodb.tbl, exclude: foodb./table_\d+/'
```

对于 Windows：

```
aws rds modify-integration ^
    --integration-identifier my-integration ^
    --data-filter 'include: foodb.*, exclude: foodb.tbl, exclude: foodb./table_\d+/'
```

RDS API

要使用 RDS API 修改零 ETL 集成，请调用 ModifyIntegration 操作。指定集成标识符，并提供逗号分隔的筛选条件模式列表。

从集成中移除数据筛选条件

当您从集成中移除数据筛选条件时，Aurora 会重新评估剩余的筛选条件，就好像移除的筛选条件从未存在过一样。然后，它将所有以前排除但现在符合条件的数据复制到目标数据仓库。这会触发所有受影响的表重新同步。

向源 Aurora 数据库集群中添加数据并在 Amazon Redshift 中对其进行查询

要创建将数据从 Amazon Aurora 复制到 Amazon Redshift 的零 ETL 集成，您必须在 Amazon Redshift 中创建目标数据库。

首先，连接到您的 Amazon Redshift 集群或工作组，并创建一个引用您的集成标识符的数据库。然后，您可以向 Aurora 数据库集群中添加数据，并在 Amazon Redshift 中查看其复制的内容。

主题

- 在 Amazon Redshift 中创建目标数据库
- 向源数据库集群中添加数据
- 在 Amazon Redshift 中查询您的 Aurora 数据
- Aurora 和 Amazon Redshift 数据库之间的数据类型差异

在 Amazon Redshift 中创建目标数据库

在开始将数据复制到 Amazon Redshift 中之前，创建集成后，您必须在目标数据仓库中创建一个目标数据库。此目标数据库必须包含对集成标识符的引用。您可以使用 Amazon Redshift 控制台或查询编辑器 v2 来创建数据库。

有关创建目标数据库的说明，请参阅在 Amazon Redshift 中创建目标数据库。

向源数据库集群中添加数据

在配置集成后，您可以将一些数据添加到您希望复制到 Amazon Redshift 数据仓库的 Aurora 数据库集群中。

> **Note**
>
> Amazon Aurora 和 Amazon Redshift 中的数据类型存在差异。有关数据类型映射的表，请参阅the section called “数据类型差异”。

首先，使用您选择的 MySQL 或 PostgreSQL 客户端连接到源数据库集群。有关说明，请参阅 the section called “连接到数据库集群”。

然后，创建一个表并插入一行示例数据。

> **Important**
>
> 确保该表有主键。否则，它无法复制到目标数据仓库。

pg_dump 和 pg_restore PostgreSQL 实用程序最初创建没有主键的表，然后添加主键。如果您使用其中一个实用程序，我们建议您先创建一个架构，然后在单独的命令中加载数据。

MySQL

以下示例使用 MySQL Workbench 实用程序。

```
CREATE DATABASE my_db;

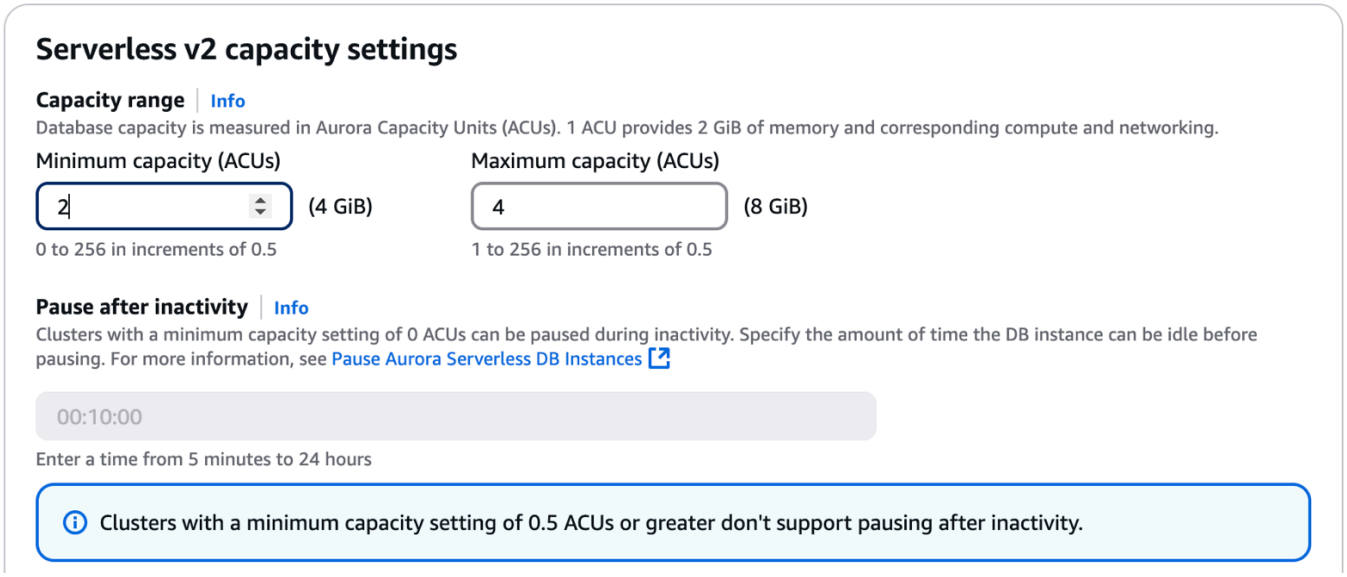
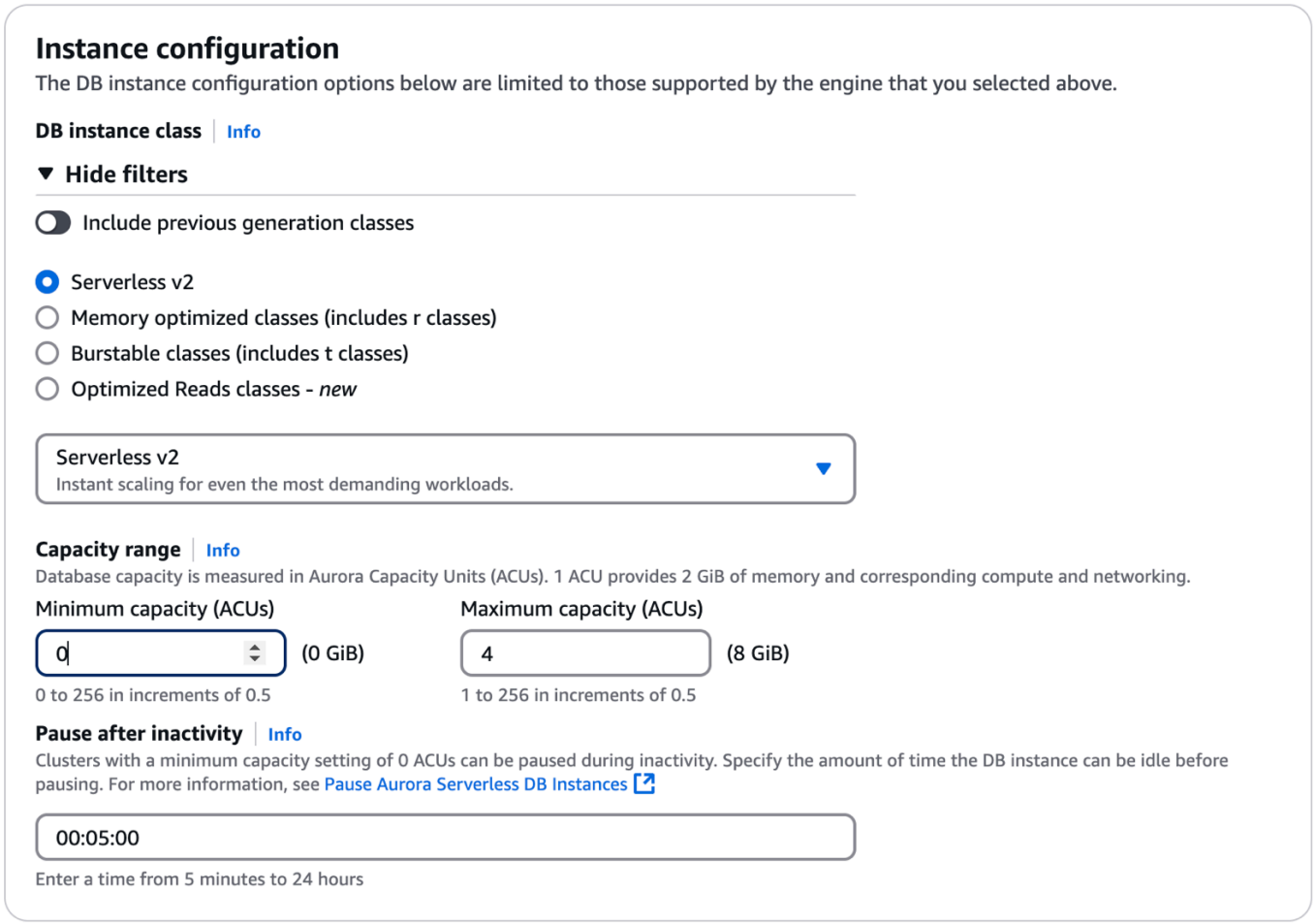
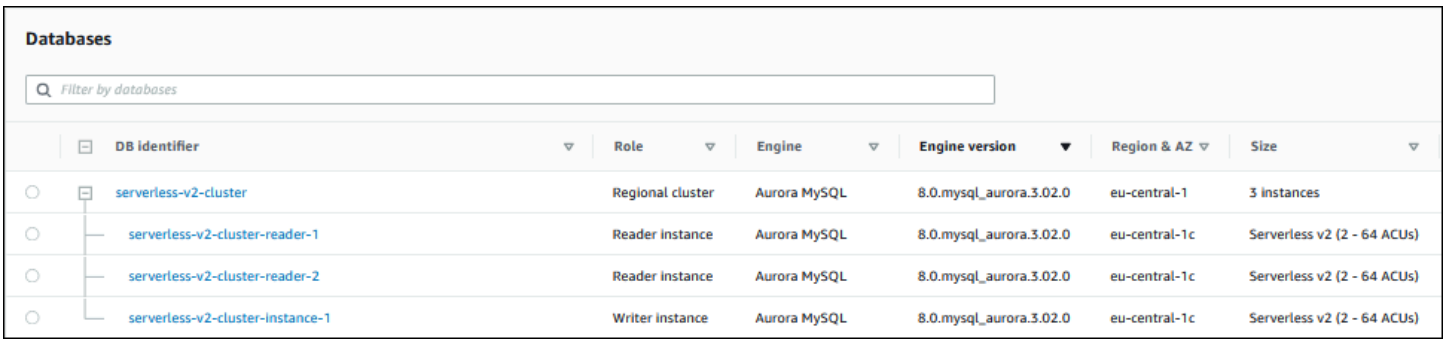
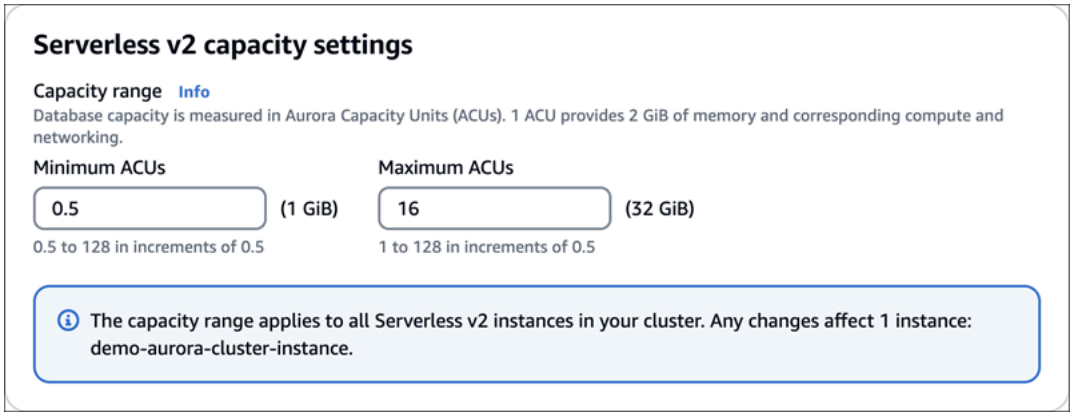
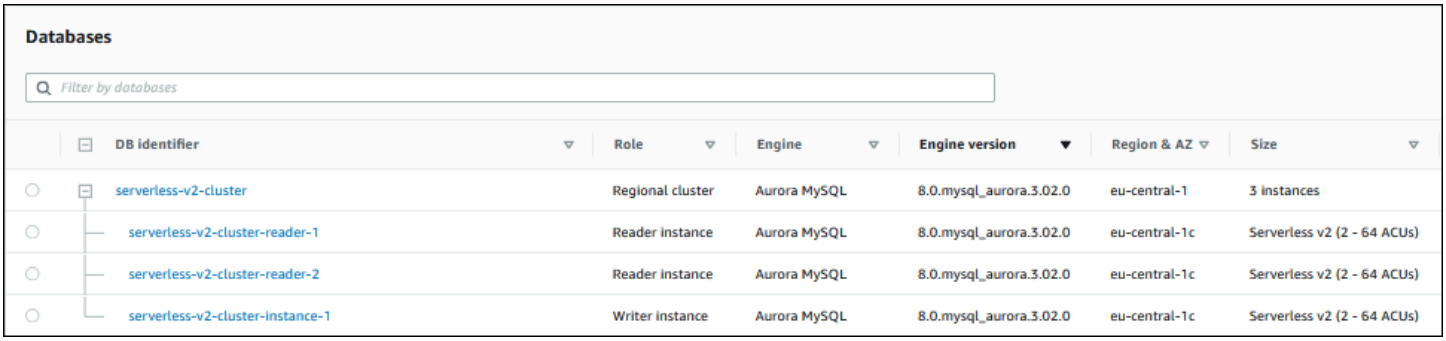
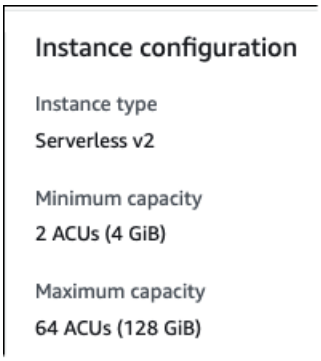
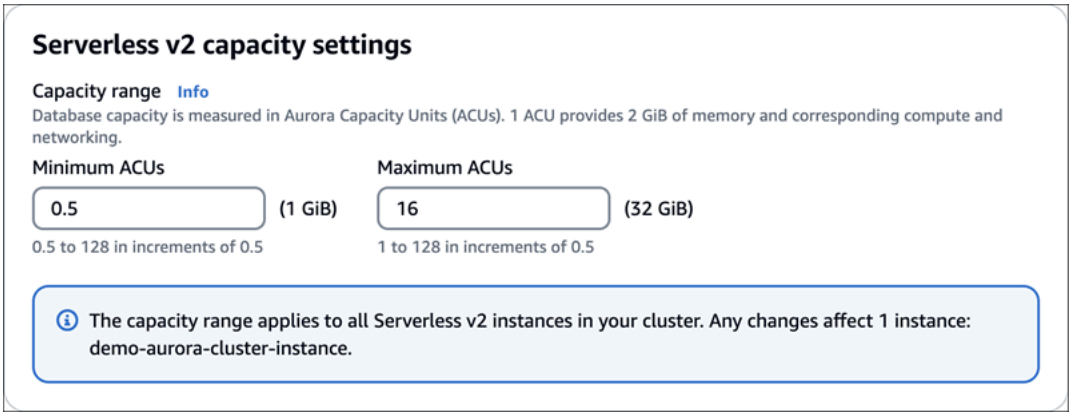
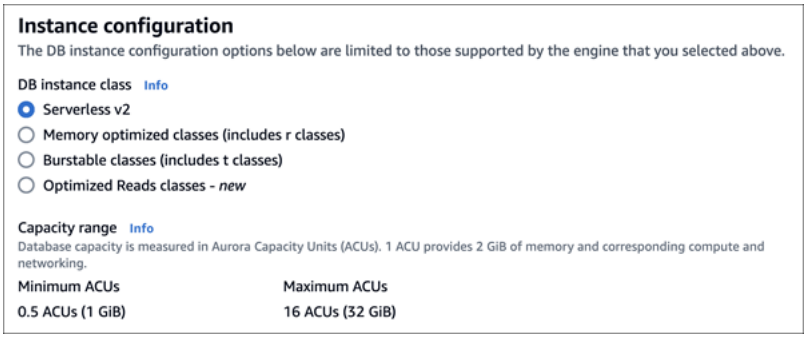
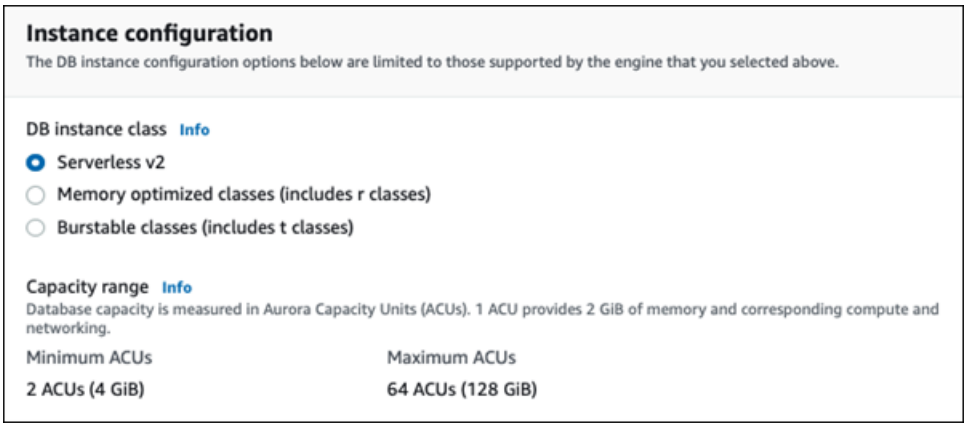
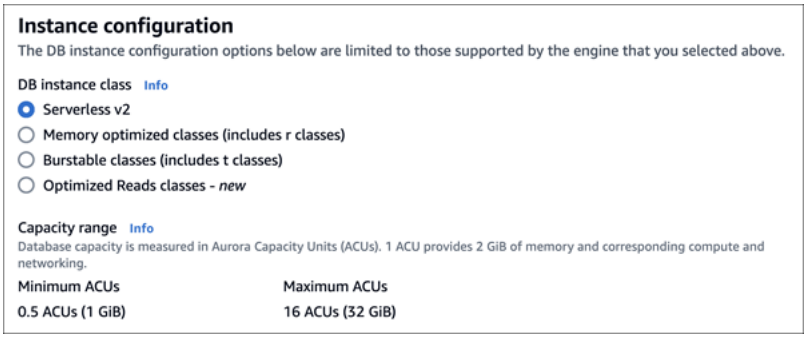
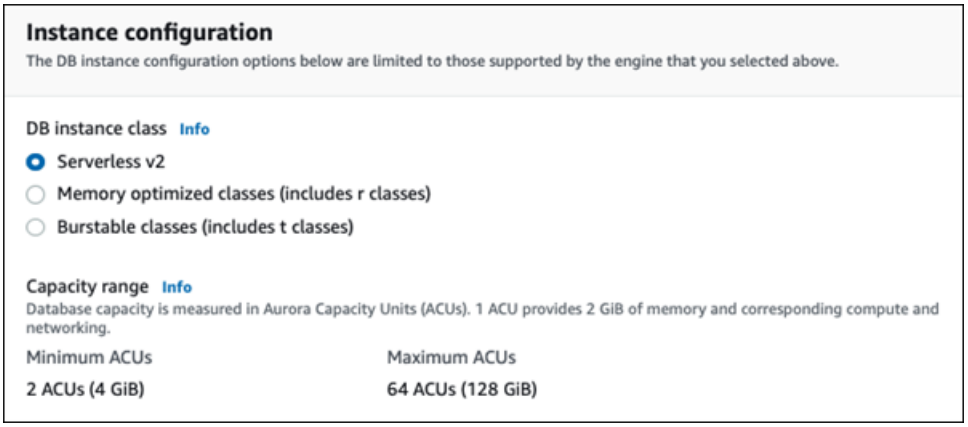
USE my_db;
```

```
CREATE TABLE books_table (ID int NOT NULL, Title VARCHAR(50) NOT NULL, Author
 VARCHAR(50) NOT NULL,
Copyright INT NOT NULL, Genre VARCHAR(50) NOT NULL, PRIMARY KEY (ID));

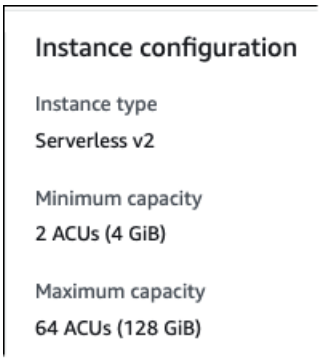
INSERT INTO books_table VALUES (1, 'The Shining', 'Stephen King', 1977, 'Supernatural
 fiction');
```

PostgreSQL

以下示例使用 psql PostgreSQL 交互式终端。连接到集群时，请包括您在创建集成时指定的命名数据库。

```
psql -h mycluster.cluster-123456789012.us-east-2.rds.amazonaws.com -p 5432 -U username
 -d named_db;

named_db=> CREATE TABLE books_table (ID int NOT NULL, Title VARCHAR(50) NOT NULL,
 Author VARCHAR(50) NOT NULL,
Copyright INT NOT NULL, Genre VARCHAR(50) NOT NULL, PRIMARY KEY (ID));

named_db=> INSERT INTO books_table VALUES (1, "The Shining", "Stephen King", 1977,
 "Supernatural fiction");
```

在 Amazon Redshift 中查询您的 Aurora 数据

将数据添加到 Aurora 数据库集群后，它会复制到 Amazon Redshift 中并准备好以供查询。

查询复制的数据

1. 导航到 Amazon Redshift 控制台，然后从左侧导航窗格中选择查询编辑器 v2。
2. 连接到您的集群或工作组，然后从下拉菜单中选择您通过集成创建的目标数据库（本示例中为 destination_database）。有关创建目标数据库的说明，请参阅在 Amazon Redshift 中创建目标数据库。
3. 使用 SELECT 语句来查询您的数据。在本例中，您可以运行以下命令，从您在源 Aurora 数据库集群中创建的表中选择所有数据：

   ```
   SELECT * from my_db."books_table";
   ```



- *my_db* 是 Aurora 数据库模式名称。仅 MySQL 数据库需要此选项。
- *books_table* 是 Aurora 表名称。

也可以使用命令行客户端查询数据。例如：

```
destination_database=# select * from my_db."books_table";

 ID |       Title |        Author |   Copyright |                 Genre |  txn_seq |
 txn_id
----+-------------+---------------+-------------+-----------------------+----------
+--------+
  1 | The Shining |  Stephen King |        1977 |   Supernatural fiction |        2 |
 12192
```

> **Note**
>
> 为了区分大小写，请对架构、表和列名使用双引号（""）。有关更多信息，请参阅 enable_case_sensitive_identifier。

Aurora 和 Amazon Redshift 数据库之间的数据类型差异

下表显示 Aurora MySQL 或 Aurora PostgreSQL 数据类型与相应 Amazon Redshift 数据类型的映射。Amazon Aurora 目前仅支持将这些数据类型用于零 ETL 集成。

如果您的源数据库集群中的表包含不受支持的数据类型，则该表将不同步并且 Amazon Redshift 目标无法使用该表。从源到目标的流式传输仍在继续，但数据类型不受支持的表不可用。要修复该表并使其在 Amazon Redshift 中可用，您必须手动恢复重大更改，然后通过运行 `ALTER DATABASE...INTEGRATION REFRESH` 来刷新集成。

主题

- Aurora MySQL
- Aurora PostgreSQL

Aurora MySQL

Aurora MySQL 数据类型	Amazon Redshift 数据类型	描述	限制
INT	INTEGER	有符号的四字节整数	
SMALLINT	SMALLINT	有符号的二字节整数	
TINYINT	SMALLINT	有符号的二字节整数	
MEDIUMINT	INTEGER	有符号的四字节整数	
BIGINT	BIGINT	有符号的八字节整数	
INT UNSIGNED	BIGINT	有符号的八字节整数	
TINYINT UNSIGNED	SMALLINT	有符号的二字节整数	
MEDIUMINT UNSIGNED	INTEGER	有符号的四字节整数	
BIGINT UNSIGNED	DECIMAL(20,0)	可选精度的精确数字	
DECIMAL(p,s) = NUMERIC(p,s)	DECIMAL (p,s)	可选精度的精确数字	不支持精度大于 38 和比例大于 37

Aurora MySQL 数据类型	Amazon Redshift 数据类型	描述	限制
DECIMAL(p,s) UNSIGNED = NUMERIC(p,s) UNSIGNED	DECIMAL (p,s)	可选精度的精确数字	不支持精度大于 38 和比例大于 37
FLOAT4/REAL	REAL	单精度浮点数	
FLOAT4/REAL UNSIGNED	REAL	单精度浮点数	
DOUBLE/REAL/FLOAT8	DOUBLE PRECISION	双精度浮点数	
DOUBLE/REAL/FLOAT8 UNSIGNED	DOUBLE PRECISION	双精度浮点数	
BIT(n)	VARBYTE(8)	长度可变的二进制值	
BINARY(n)	VARBYTE(n)	长度可变的二进制值	
VARBINARY (n)	VARBYTE(n)	长度可变的二进制值	
CHAR(n)	VARCHAR (n)	长度可变的字符串值	
VARCHAR (n)	VARCHAR (n)	长度可变的字符串值	
TEXT	VARCHAR(65535)	长度可变、最多 65535 字节的字符串值	
TINYTEXT	VARCHAR(255)	长度可变、最多 255 字节的字符串值	

Aurora MySQL 数据类型	Amazon Redshift 数据类型	描述	限制
MEDIUMTEXT	VARCHAR(65535)	长度可变、最多 65535 字节的字符串值	
LONGTEXT	VARCHAR(65535)	长度可变、最多 65535 字节的字符串值	
ENUM	VARCHAR(1020)	长度可变、最多 1020 字节的字符串值	
SET	VARCHAR(1020)	长度可变、最多 1020 字节的字符串值	
DATE	DATE	日历日期（年、月、日）	
DATETIME	TIMESTAMP	日期和时间（没有时区）	
TIMESTAMP(p)	TIMESTAMP	日期和时间（没有时区）	
TIME	VARCHAR(18)	长度可变、最多 18 字节的字符串值	
YEAR	VARCHAR(4)	长度可变、最多 4 字节的字符串值	
JSON	SUPER	作为值的半结构化数据或文档	

Aurora PostgreSQL

Aurora PostgreSQL 的零 ETL 集成不支持自定义数据类型或由扩展创建的数据类型。

> **Important**
>
> Aurora PostgreSQL 与 Amazon Redshift 功能的零 ETL 集成已发布预览版。文档和特征都可能会更改。您仅可在测试环境中使用此功能，而不要在生产环境中使用。有关预览条款和条件，请参阅 AWS 服务条款中的测试版和预览。

Aurora PostgreSQL 数据类型	Amazon Redshift 数据类型	描述	限制
bigint	BIGINT	有符号的八字节整数	
bigserial	BIGINT	有符号的八字节整数	
bit(n)	VARBYTE(n)	长度可变的二进制值	
bit varying(n)	VARBYTE(n)	长度可变的二进制值	
bit	VARBYTE(1024000)	长度可变、最多 1,024,000 字节的字符串值	
布尔值	BOOLEAN	逻辑布尔值（true/false)	
bytea	VARBYTE(1024000)	长度可变、最多 1024000 字节的字符串值	
character(n)	CHAR(n)	固定长度字符串	

Aurora PostgreSQL 数据类型	Amazon Redshift 数据类型	描述	限制
character varying(n)	VARCHAR(65535)	长度可变的字符串值	
date	DATE	日历日期（年、月、日）	• 不支持大于 9999-12-31 的值 • 不支持 B.C. 值
double precision	DOUBLE PRECISION	双精度浮点数	不支持亚正常值
整数	INTEGER	有符号的四字节整数	
money	DECIMAL(20,3)	货币金额	
numeric(p,s)	DECIMAL (p,s)	长度可变的字符串值	• 不支持 NaN 值 • 不支持精度大于 38 和比例大于 37 • 不支持负比例
real	REAL	单精度浮点数	
smallint	SMALLINT	有符号的二字节整数	
smallserial	SMALLINT	有符号的二字节整数	
Serial	INTEGER	有符号的四字节整数	
文本	VARCHAR(65535)	长度可变、最多 65535 字节的字符串值	

Aurora PostgreSQL 数据类型	Amazon Redshift 数据类型	描述	限制
time [(p)] [没有时区]	VARCHAR(19)	长度可变、最多 19 字节的字符串值	不支持 `Infinity` 和 `-Infinity` 值
有时区的 time [(p)]	VARCHAR(22)	长度可变、最多 22 字节的字符串值	• 不支持 `Infinity` 和 `-Infinity` 值
timestamp [(p)] [没有时区]	TIMESTAMP	日期和时间（没有时区）	• 不支持 `Infinity` 和 `-Infinity` 值 • 不支持大于 `9999-12-31` 的值 • 不支持 B.C. 值
有时区的 timestamp [(p)]	TIMESTAMPTZ	日期和时间（有时区）	• 不支持 `Infinity` 和 `-Infinity` 值 • 不支持大于 `9999-12-31` 的值 • 不支持 B.C. 值

查看和监控 Aurora 零 ETL 集成

您可以查看 Amazon Aurora 零 ETL 集成的详细信息，以查看其配置信息和当前状态。您还可以通过在 Amazon Redshift 中查询特定的系统视图来监控集成的状态。此外，Amazon Redshift 向 Amazon CloudWatch 发布某些与集成相关的指标，您可以在 Amazon Redshift 控制台中查看这些指标。

主题

- 查看集成
- 对于 Amazon Redshift 使用系统表监控集成
- 监控与适用于 Amazon Redshift 的 Amazon EventBridge 的集成

查看集成

您可以使用 AWS Management Console、AWS CLI 或 RDS API 来查看 Aurora 零 ETL 集成。

控制台

查看零 ETL 集成的详细信息

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 从左侧导航窗格中，选择零 ETL 集成。
3. 选择一个集成以查看其详细信息，例如其源数据库集群和目标数据仓库。



集成可以具有以下状态：

- `Creating` – 正在创建集成。
- `Active` – 集成正在将事务数据发送到目标数据仓库。
- `Syncing` – 集成遇到了可恢复的错误，正在重新设置数据种子。受影响的表在完成重新同步之前不可用于查询。

- `Needs attention` – 集成遇到了需要手动干预才能解决的事件或错误。要修复问题，请按照集成详细信息页面上错误消息中的说明修复问题。
- `Failed` – 集成遇到了无法恢复的事件或错误，无法修复。您必须删除并重新创建集成。
- `Deleting` – 正在删除集成。

AWS CLI

要使用 AWS CLI 查看当前账户中的所有 零 ETL 集成，请使用 describe-integrations 命令并指定 `--integration-identifier` 选项。

Example

对于 Linux、macOS 或 Unix：

```
aws rds describe-integrations \
    --integration-identifier ee605691-6c47-48e8-8622-83f99b1af374
```

对于 Windows：

```
aws rds describe-integrations ^
    --integration-identifier ee605691-6c47-48e8-8622-83f99b1af374
```

RDS API

要使用 Amazon RDS API 查看零 ETL 集成，请结合 `IntegrationIdentifier` 参数使用 `DescribeIntegrations` 操作。

对于 Amazon Redshift 使用系统表监控集成

Amazon Redshift 具有包含系统运行方式相关信息的系统表和视图。您可以像查询任何其他数据库表那样查询这些系统表和视图。有关 Amazon Redshift 中的系统表和视图的更多信息，请参阅《Amazon Redshift 数据库开发人员指南》中的系统表和视图参考。

您可以查询以下系统视图和表，以获取有关 Aurora 零 ETL 集成的信息：

- SVV_INTEGRATION – 为您的集成提供配置详细信息。
- SVV_INTEGRATION_TABLE_STATE – 描述集成中每个表的状态。
- SYS_INTEGRATION_TABLE_STATE_CHANGE – 显示集成的表状态更改日志。
- SYS_INTEGRATION_ACTIVITY – 提供有关已完成的集成运行的信息。

所有集成相关的 Amazon CloudWatch 指标均源自 Amazon Redshift。有关更多信息，请参阅《Amazon Redshift 管理指南》中的零 ETL 集成的指标。目前，Amazon Aurora 不向 Amazon CloudWatch 发布任何与集成相关的指标。

监控与适用于 Amazon Redshift 的 Amazon EventBridge 的集成

Amazon Redshift 会将集成相关的事件发送到 Amazon EventBridge。有关事件及其相应事件 ID 的列表，请参阅《Amazon Redshift 管理指南》中的与 Amazon EventBridge 的零 ETL 集成事件通知。

修改 Aurora 零 ETL 集成

您只能在受支持的数据仓库中修改零 ETL 集成的名称、描述和数据筛选选项。您无法修改用于加密集成的 AWS KMS 密钥，也无法修改源数据库或目标数据库。

如果您将数据筛选条件添加到现有集成中，Aurora 会重新评估筛选条件，就好像此筛选条件一直存在一样。它会移除当前位于目标数据仓库中，但与新筛选标准不匹配的所有数据。如果您从集成中移除数据筛选条件，它会将以前不符合筛选标准（但现在符合）的所有数据复制到目标数据仓库中。有关更多信息，请参阅 the section called “零 ETL 集成的数据筛选”。

您可以使用 AWS Management Console、AWS CLI 或 Amazon RDS API 修改零 ETL 集成。

RDS 控制台

修改零 ETL 集成

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择零 ETL 集成，然后选择要修改的集成。
3. 选择修改，然后对任何可用设置进行修改。
4. 当所有更改都达到您的要求时，选择修改。

AWS CLI

要使用 AWS CLI 修改零 ETL 集成，请调用 modify-integration 命令。与 `--integration-identifier` 一起，指定以下任意选项：

- `--integration-name` – 为集成指定新名称。
- `--description` – 为集成指定新描述。

- `--data-filter` – 为集成指定数据筛选选项。有关更多信息，请参阅 the section called “零 ETL 集成的数据筛选”。

Example

以下请求修改现有集成。

对于 Linux、macOS 或 Unix：

```
aws rds modify-integration \
    --integration-identifier ee605691-6c47-48e8-8622-83f99b1af374 \
    --integration-name my-renamed-integration
```

对于 Windows：

```
aws rds modify-integration ^
    --integration-identifier ee605691-6c47-48e8-8622-83f99b1af374 ^
    --integration-name my-renamed-integration
```

RDS API

要使用 RDS API 修改零 ETL 集成，请调用 ModifyIntegration 操作。指定集成标识符，以及您要修改的参数。

删除 Aurora 零 ETL 集成

当您删除零 ETL 集成时，Amazon Aurora 会将其从源Aurora 数据库集群中删除。事务数据不会从 Amazon Aurora 或分析目标中删除，但 Aurora 不会向 Amazon Redshift 或 Amazon SageMaker 发送新数据。

只有当集成的状态为 `Active`、`Failed`、`Syncing` 或 `Needs attention` 时，您才能将其删除。

您可以使用 AWS Management Console、AWS CLI 或 RDS API 删除零 ETL 集成。

控制台

删除零 ETL 集成

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。

2. 从左侧导航窗格中，选择零 ETL 集成。
3. 选择要删除的零 ETL 集成。
4. 依次选择操作和删除，然后确认删除。

AWS CLI

要删除零 ETL 集成，请使用 delete-integration 命令并指定 `--integration-identifier` 选项。

Example

对于 Linux、macOS 或 Unix：

```
aws rds delete-integration \
    --integration-identifier ee605691-6c47-48e8-8622-83f99b1af374
```

对于 Windows：

```
aws rds delete-integration ^
    --integration-identifier ee605691-6c47-48e8-8622-83f99b1af374
```

RDS API

要使用 Amazon RDS API 删除零 ETL 集成，请结合 `IntegrationIdentifier` 参数使用 `DeleteIntegration` 操作。

Aurora 与 Amazon Redshift 的零 ETL 集成问题排查

您可以通过在 Amazon Redshift 中查询 SVV_INTEGRATION 系统表来检查零 ETL 集成的状态。如果 `state` 列的值为 `ErrorState`，则表示有问题。有关更多信息，请参阅 the section called “使用系统表进行监控”。

使用以下信息来排查 Aurora 与 Amazon Redshift 的零 ETL 集成的常见问题。

主题

- 我无法创建零 ETL 集成
- 我的集成卡在 Syncing 状态
- 我的表未复制到 Amazon Redshift
- 我的一个或多个 Amazon Redshift 表需要重新同步

我无法创建零 ETL 集成

如果您无法创建零 ETL 集成，请确保您的源数据库集群的以下内容正确无误：

- 您的源数据库集群正在运行 Aurora MySQL 版本 3.05（与 MySQL 8.0.32 兼容）或更高版本，或者 Aurora PostgreSQL（兼容 PostgreSQL 15.4 和零 ETL 支持）。要验证引擎版本，请选择数据库集群的配置选项卡，并检查引擎版本。
- 您已正确配置了数据库集群参数。如果所需参数设置不正确或与集群不关联，则创建将失败。请参阅 the section called “步骤 1：创建自定义数据库集群参数组”。

此外，请确保您的数据仓库在以下方面正确无误：

- 已启用区分大小写。请参阅为您的数据仓库开启区分大小写。
- 您添加了正确的授权主体和集成源。请参阅为您的 Amazon Redshift 数据仓库配置授权。
- 数据仓库已加密（如果它是预调配集群）。请参阅 Amazon Redshift 数据库加密。

我的集成卡在 `Syncing` 状态

如果您更改其中一个必需的数据库集群参数的值，则集成状态可能会始终显示为 `Syncing`。

要修复此问题，请检查与源数据库集群关联的参数组中的参数值，并确保它们与所需值相匹配。有关更多信息，请参阅 the section called “步骤 1：创建自定义数据库集群参数组”。

如果您修改任何参数，请务必重启数据库集群来应用更改。

我的表未复制到 Amazon Redshift

如果您没有看到一个或多个表反映在 Amazon Redshift 中，可以运行以下命令来重新同步表：

```
ALTER DATABASE dbname INTEGRATION REFRESH TABLES table1, table2;
```

有关更多信息，请参阅《Amazon Redshift SQL 参考》中的 ALTER DATABASE。

您的数据可能未复制，因为一个或多个源表没有主键。Amazon Redshift 中的监控控制面板将这些表的状态显示为 `Failed`，而总体零 ETL 集成的状态更改为 `Needs attention`。要解决此问题，您可以在表中确定一个可以成为主键的现有键，也可以添加合成主键。有关详细解决方案，请参阅 以下资源：

- Handle tables without primary keys while creating Amazon Aurora MySQL or Amazon RDS for MySQL zero-ETL integrations with Amazon Redshift
- Handle tables without primary keys while creating Amazon Aurora PostgreSQL zero-ETL integrations with Amazon Redshift

我的一个或多个 Amazon Redshift 表需要重新同步

在源数据库集群上运行某些命令可能需要重新同步您的表。在这些情况下，SVV_INTEGRATION_TABLE_STATE 系统视图显示 `table_state` 为 `ResyncRequired`，这意味着集成必须将该特定表的数据从 MySQL 完全重新加载到 Amazon Redshift。

当表开始重新同步时，它进入 `Syncing` 状态。您无需执行任何手动操作即可重新同步表。在表数据重新同步时，您无法在 Amazon Redshift 中访问这些数据。

以下是一些可以使表进入 `ResyncRequired` 状态的示例操作，以及可供考虑的可能替代方案。

操作	示例	或者
在特定位置添加一列	`ALTER TABLE table_name ADD COLUMN column_name INTEGER NOT NULL first;`	Amazon Redshift 不支持使用 `first` 或 `after` 关键字将列添加到特定位置。如果目标表中列的顺序不重要，请使用更简单的命令将该列添加到表的末尾： `ALTER TABLE table_name ADD COLUMN column_na`

操作	示例	或者
		`me column_type ;`
使用默认 CURRENT_TIMESTAMP 添加时间戳列	`ALTER TABLE table_name ADD COLUMN column_name TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP;`	现有表行的 CURRENT_TIMESTAMP 值由 Aurora MySQL 计算，如果不进行完整表数据重新同步，就无法在 Amazon Redshift 中进行模拟。 如果可能，请将默认值切换为文本常量（例如 2023-01-01 00:00:15），以避免表可用性出现延迟。
在单个命令中执行多个列操作	`ALTER TABLE table_name ADD COLUMN column_1, RENAME COLUMN column_2 TO column_3;`	考虑将命令拆分为两个单独的操作（ADD 和 RENAME），这不需要重新同步。

使用 Aurora Serverless v2

Aurora Serverless v2 是一种适用于 Amazon Aurora 的按需自动扩展配置。Aurora Serverless v2 帮助自动执行监控工作负载和调整数据库容量的过程。容量会根据应用程序需求自动调整。您只需为数据库集群使用的资源付费。因此，Aurora Serverless v2 可以帮助您保持在预算范围内，避免为不使用的计算资源付费。

这种自动化对于多租户数据库、分布式数据库、开发和测试系统以及其他具有高度可变且不可预测工作负载的环境尤为重要。

主题

- Aurora Serverless v2 使用案例
- Aurora Serverless v2 的优点
- Aurora Serverless v2 的工作原理
- Aurora Serverless v2 的要求和限制
- 创建一个使用 Aurora Serverless v2 的数据库集群
- 管理 Aurora Serverless v2 数据库集群
- Aurora Serverless v2 的性能和扩缩
- 针对 Aurora Serverless v2 缩减到零 ACU 从而自动暂停和恢复
- 迁移到 Aurora Serverless v2

Aurora Serverless v2 使用案例

Aurora Serverless v2 支持很多类型的数据库工作负载。其范围涵盖从开发和测试环境到具有不可预测工作负载的网站和应用程序，再到要求高规模和高可用性的最苛刻的业务关键型应用程序。

Aurora Serverless v2 对于以下使用案例特别有用：

- 可变工作负载 – 您运行的工作负载会突然出现无法预测的活动增加。例如，在开始下雨时显示活动涌现的流量网站。另一个例子是当您提供销售或特别促销活动时流量会增加的电子商务网站。有了 Aurora Serverless v2，您的数据库可自动扩展容量以满足应用程序的峰值负载需求，然后在活动涌现结束时再缩减。有了 Aurora Serverless v2，您不再需要为峰值或平均容量进行预置。您可以指定容量上限来应对最糟糕情况，除非确实需要该容量，否则不使用该容量。

Aurora Serverless v2 中的扩缩粒度帮助您使容量与数据库的需求紧密匹配。对于预置的集群，扩展需要添加一个全新的数据库实例。对于 Aurora Serverless v1 集群，扩展需要将集群的 Aurora 容量单位（ACU）数增加一倍，例如从 16 个增加到 32 个或从 32 个增加到 64 个。相比之下，当只需要多一点容量时，Aurora Serverless v2 可以增加半个 ACU。根据处理工作负载增加所需的额外容量，可以增加 0.5、1、1.5、2 或额外的半个 ACU。当工作负载减少且不再需要该容量时，可以删除 0.5、1、1.5、2 或额外的半个 ACU。

- 多租户应用程序 – 有了 Aurora Serverless v2，您无需为队列中的每个应用程序单独地管理数据库容量。Aurora Serverless v2 会为您管理各个数据库容量。

  您可为每个租户创建一个集群。这样，您就可以视情况使用克隆、快照还原和 Aurora 全局数据库等功能为每个租户增强高可用性和灾难恢复。

  根据一天中的时间、一年中的时间、促销活动等，每个租户可能有特定的忙碌和空闲时段。每个集群都有宽容量范围。这样，活动量较少的集群将产生更低的数据库实例费用。任何集群都可以快速扩展以应对活动较多的时期。

- 新应用程序 – 您正在部署新应用程序，但不确定所需的数据库实例大小。使用 Aurora Serverless v2，您可以设置具有一个或多个数据库实例的集群，并让数据库根据应用程序的容量需求自动扩展。

- 混合用途应用程序 – 假设您有一个在线交易处理（OLTP）应用程序，但是您会周期性地经历查询流量高峰。通过在集群中指定 Aurora Serverless v2 数据库实例的提升层，您可以将集群配置为让读取器数据库实例可以独立于写入器数据库实例进行扩展以处理额外负载。当使用高峰消退时，读取器数据库实例会缩减以匹配写入器数据库实例的容量。

- 容量规划 – 假设您通常通过修改集群中所有数据库实例的数据库实例类来调整数据库容量，或者验证工作负载的最佳数据库容量。借助 Aurora Serverless v2，您可以避免产生此管理开销。您可以通过运行工作负载并检查数据库实例实际扩展的程度来确定适当的最小和最大容量。

  您可以将现有数据库实例从预置修改为 Aurora Serverless v2 或者从 Aurora Serverless v2 修改为预置。在这种情况下，您无需创建新集群或新数据库实例。

  使用 Aurora 全局数据库，辅助集群的容量可能不需要像主集群那样多。您可以在辅助群集中使用 Aurora Serverless v2 数据库实例。这样，如果辅助区域得到提升并接管应用程序的工作负载，集群容量就可以扩展。

- 开发和测试 - 除了运行要求最苛刻的应用程序外，您还可以将 Aurora Serverless v2 用于开发和测试环境。借助 Aurora Serverless v2，您可以创建最低容量较低的数据库实例，而不是使用可突增的 db.t* 数据库实例类。您可以将最大容量设置得足够高，以便这些数据库实例仍然可以运行大量工作

负载，而不会出现内存不足的情况。当数据库未在使用时，所有数据库实例都会缩减以避免产生不必要的费用。

> **Tip**
>
> 为了方便在开发和测试环境中使用 Aurora Serverless v2，在创建新集群时，AWS Management Console 提供 Easy create（轻松创建）快捷方式。如果选择 Dev/Test（开发/测试）选项，Aurora 会创建一个具有 Aurora Serverless v2 数据库实例以及开发和测试系统典型容量范围的集群。

将 Aurora Serverless v2 用于现有的已调配工作负载

假设您已在预置集群上运行 Aurora 应用程序。您可以将一个或多个 Aurora Serverless v2 数据库实例作为读取器数据库实例添加到现有集群，检查应用程序将如何与 Aurora Serverless v2 配合使用。您可以查看读取器数据库实例扩展和缩减的频率。您可以使用 Aurora 故障转移机制将 Aurora Serverless v2 数据库实例提升为写入器，并检查它如何处理读/写工作负载。这样，您可以在最短的停机时间内进行切换，而且无需更改客户端应用程序使用的端点。有关将现有群集转换为 Aurora Serverless v2 的程序的详细信息，请参阅迁移到 Aurora Serverless v2。

Aurora Serverless v2 的优点

Aurora Serverless v2 适用于可变或“高峰”工作负载。对于此类不可预测的工作负载，您可能难以规划何时更改数据库容量。使用熟悉的机制（例如添加数据库实例或更改数据库实例类），您可能也难以足够快地进行容量更改。Aurora Serverless v2 提供了以下优势来帮助处理此类使用案例：

- 比预置更简单的容量管理 – Aurora Serverless v2 减少了规划数据库实例大小和随着工作负载变化调整数据库实例大小的工作量。此外，还可以减少为集群中的所有数据库实例维持一致容量的工作量。
- 在活动较多的时期更快、更轻松地扩展 – Aurora Serverless v2 可根据需要扩展计算和内存容量，而无需中断客户端事务或总体工作负载。借助 Aurora Serverless v2，能够使用读取器数据库实例，除了垂直扩展外，还可以帮助您利用水平扩展。能够使用 Aurora 全局数据库，这意味着您可以在多个 AWS 区域 中分布您的 Aurora Serverless v2 读取工作负载。此功能比预置集群的扩缩机制更便利。它也比 Aurora Serverless v1 中的扩缩功能更快、更精细。
- 在活动量较少的期间经济高效 – Aurora Serverless v2 帮助您避免超额配置数据库实例。Aurora Serverless v2当数据库实例扩展时，以细粒度增量添加资源。您只需为所使用的数据库资源付费。Aurora Serverless v2 资源使用情况按秒计量。这样，当数据库实例缩减时，会立即注册减少的资源使用量。

- 使用预置集群获得更出色的功能 – 借助 Aurora Serverless v2，您可以使用 Aurora Serverless v1 中未提供的许多 Aurora 功能。例如，借助 Aurora Serverless v2，您可以使用读取器数据库实例、全局数据库、AWS Identity and Access Management（IAM）数据库身份验证和 Performance Insights。您还可以使用比 Aurora Serverless v1 更多的配置参数。

  具体而言，借助 Aurora Serverless v2，您可以利用预置集群的以下功能：

  - 读取器数据库实例 – Aurora Serverless v2 可以利用读取器数据库实例进行横向扩展。当集群包含一个或多个读取器数据库实例时，如果写入器数据库实例出现问题，集群可以立即进行故障转移。这是一项 Aurora Serverless v1 没有提供的功能。
  - 多可用区集群 – 您可以在多个可用区（AZ）分发集群的 Aurora Serverless v2 数据库实例。设置多可用区集群有助于确保业务连续性，甚至在极少数情况下会出现影响整个可用区的问题。这是一项 Aurora Serverless v1 没有提供的功能。
  - 全局数据库 – 您可以将 Aurora Serverless v2 与 Aurora 全局数据库结合使用，以便在其他 AWS 区域 中创建集群的额外只读副本，用于灾难恢复目的。
  - RDS Proxy – 您可以使用 Amazon RDS Proxy，允许您的应用程序池化和共享数据库连接，以提高其扩展能力。
- 比 Aurora Serverless v1 更快、更精细、中断更少的扩缩 – Aurora Serverless v2 可以更快地扩展和缩减。扩缩可以按照小至 0.5 ACU 的增量来更改容量，而不一定要将 ACU 的数量翻倍或减半。进行扩缩时通常完全不会暂停处理。扩缩不像 Aurora Serverless v1 那样涉及您必须注意的事件。可以在 SQL 语句运行和事务处于打开状态时进行扩缩，无需等待安静点。

Aurora Serverless v2 的工作原理

以下内容概述了 Aurora Serverless v2 的工作原理。

主题

- Aurora Serverless v2 概览
- Aurora 数据库集群的配置
- Aurora Serverless v2 容量
- Aurora Serverless v2 扩缩
- Aurora Serverless v2 和高可用性
- Aurora Serverless v2 和存储
- Aurora 集群的配置参数

Aurora Serverless v2 概览

Amazon Aurora Serverless v2 适用于要求严苛、变化很大的工作负载。例如，您的数据库使用量在短时间内可能很大，紧接着就是长时间的少量活动或完全没有活动。其中一些例子是定期举办促销活动的零售、游戏或体育网站，以及根据需要生成报告的数据库。其他一些例子包括开发和测试环境，以及使用量可能会迅速增加的新应用程序。对于类似这些情况和许多其他情况，使用预置模型并不总是能提前正确地配置容量。如果您过度预置并且保留不使用的容量，也可能导致更高的成本。

相比之下，Aurora 预置集群适合稳定的工作负载。使用预置集群，您可以选择具有预定义内存量、CPU 处理能力、I/O 带宽等的数据库实例类。如果您的工作负载发生变化，可以手动修改写入器和读取器的实例类。如果您可以在预期的消费模式之前调整容量，并且在更改集群中写入器和读取器的实例类时，出现短暂的中断是可以接受的，则预置模式很适合。

Aurora Serverless v2 专为支持可即时扩展的无服务器数据库集群而从头开始构建。Aurora Serverless v2 旨在提供与预置写入器和读取器相同程度的安全性和隔离度。这些方面在多租户无服务器云环境中至关重要。动态扩缩机制的开销非常小，因此它可以快速响应数据库工作负载的变化。它还足够强大，可以满足处理需求的急剧增长。

通过使用 Aurora Serverless v2，您可以创建 Aurora 数据库集群，而无需将每个写入器和读取器限制在特定的数据库容量。您指定最小和最大容量范围。Aurora 会在该容量范围内扩缩集群中的每个 Aurora Serverless v2 写入器或读取器。通过使用每个写入器或读取器都可以动态扩缩的多可用区集群，您可以获得动态扩缩和高可用性。

Aurora Serverless v2 根据您的最小和最大容量规格自动扩缩数据库资源。扩缩速度很快，因为大多数扩缩事件操作将写入器或读取器保持在同一主机上。在极少数情况下，Aurora Serverless v2 写入器或读取器从一个主机移动到另一个主机，Aurora Serverless v2 自动管理连接。您无需更改数据库客户端应用程序代码或数据库连接字符串。

与预置集群一样，使用 Aurora Serverless v2 时，存储容量和计算容量是分开的。当我们提到 Aurora Serverless v2 容量和扩缩时，增加或减少的总是计算容量。因此，即使 CPU 和内存容量缩减至低水平，您的集群仍会包含许多 TB 的数据。

您可以指定数据库容量，而不是调配和管理数据库服务器。有关 Aurora Serverless v2 容量的详细信息，请参阅 Aurora Serverless v2 容量。每个 Aurora Serverless v2 写入器或读取器的实际容量会随着时间的推移而发生变化，具体取决于您的工作负载。有关该机制的详细信息，请参阅 Aurora Serverless v2 扩缩。

> **Important**
>
> 使用 Aurora Serverless v1 时，您的集群具有单一的计算容量衡量标准，可以在最小容量值和最大容量值之间进行扩缩。使用 Aurora Serverless v2 时，除了写入器之外，您的集群还可以包含读取器。每个 Aurora Serverless v2 写入器和读取器可以在最小容量和最大容量值之间扩缩。因此，Aurora Serverless v2 集群的总容量取决于为数据库集群定义的容量范围以及集群中的写入器和读取器数量。在任何给定时间，您都只需为您的 Aurora 数据库集群中正在主动使用的 Aurora Serverless v2 容量付费。

Aurora 数据库集群的配置

对于每个 Aurora 数据库集群，您可以选择 Aurora Serverless v2 容量、预置容量或两者的任意组合。

您可以设置一个同时包含 Aurora Serverless v2 和预置容量的集群，称为混合配置集群。例如，假设您需要比 Aurora Serverless v2 写入器所提供容量更多的读/写容量。在这种情况下，您可以使用非常大的预置写入器来设置集群。在这种情况下，您仍然可为读取器使用 Aurora Serverless v2。或者假设集群的写入工作负载会变化，但读取工作负载保持稳定。在这种情况下，您可以使用 Aurora Serverless v2 写入器和一个或多个预置读取器设置集群。

您还可以设置由 Aurora Serverless v2 管理所有容量的数据库集群。为此，您可以创建新的集群并从头开始使用 Aurora Serverless v2。或者，您可以将现有集群中的所有预置容量替换为 Aurora Serverless v2。例如，来自旧引擎版本的一些升级路径需要从预置写入器开始，然后用 Aurora Serverless v2 写入器来替换它。对于使用 Aurora Serverless v2 创建新数据库集群或者将现有数据库集群切换到 Aurora Serverless v2 的过程，请参阅创建 Aurora Serverless v2 数据库集群和从预置集群切换到 Aurora Serverless v2。

如果您在数据库集群中根本不使用 Aurora Serverless v2，则数据库集群中的所有写入器和读取器都为预置。这是大多数用户都熟悉的最古老、最常见的数据库集群类型。事实上，在 Aurora Serverless 之前，这种 Aurora 数据库集群没有特殊名称。预置容量是恒定的。费用相对更容易预测。但是，您必须提前预测所需的容量。在某些情况下，您的预测可能不准确，或者容量需求可能会发生变化。在这些情况下，您的数据库集群可能会变得配置不足（比您期望的更慢）或过度配置（比您期望的更昂贵）。

Aurora Serverless v2 容量

Aurora Serverless v2 的计量单位是 Aurora 容量单位（ACU）。Aurora Serverless v2 容量与您用于预置集群的数据库实例类无关。

每个 ACU 是约 2 GiB 的内存、相应的 CPU 和网络的组合。您可以使用此计量单位指定数据库容量范围。`ServerlessDatabaseCapacity` 和 `ACUUtilization` 指标帮助您确定数据库实际使用的容量以及该容量在指定范围内的位置。

在任何时候，每个 Aurora Serverless v2 数据库写入器或读取器有一个容量。该容量用代表 ACU 的浮点数表示。每当写入器或读取器扩缩时，容量会增加或减少。此值每秒测量一次。对于您打算使用 Aurora Serverless v2 的每个数据库集群，定义一个容量范围：每个 Aurora Serverless v2 写入器或读取器可以在之间进行扩缩的最小容量和最大容量值。数据库集群中的每个 Aurora Serverless v2 写入器或读取器的容量范围都相同。每个 Aurora Serverless v2 写入器或读取器有自己的容量，该容量处于该范围内的某个位置。

下表显示 Aurora MySQL 和 Aurora PostgreSQL 支持的 Aurora Serverless v2 容量范围。

容量范围（ACU）	Aurora MySQL 支持的版本	Aurora PostgreSQL 支持的版本
0.5–128	3.02.0 及更高版本	13.6 及更高版本、14.3 及更高版本、15.2 及更高版本、16.1 及更高版本
0.5–256	3.06.0 及更高版本	13.13 及更高版本、14.10 及更高版本、15.5 及更高版本、16.1 及更高版本
0–256	3.08.0 及更高版本	13.15 及更高版本、14.12 及更高版本、15.7 及更高版本、16.3 及更高版本

对于支持自动暂停功能的 Aurora Serverless v2 版本，您可以定义的最小 Aurora Serverless v2 容量为 0 ACU。如果它小于或等于最大容量值，您可以指定一个较高的数值。将最小容量设置为一个很小的数字，可以让负载较轻的数据库集群消耗最少的计算资源。同时，他们随时准备立即接受连接，并在变得忙碌时进行扩展。

我们建议将最小值设置为让每个数据库写入器或读取器可以在缓冲池中保持应用程序工作集的值。这样，在空闲期间，缓冲池的内容不会被丢弃。有关选择最小容量值时的所有注意事项，请参阅选择集群的最小 Aurora Serverless v2 容量设置。有关选择最大容量值时的所有注意事项，请参阅选择集群的最大 Aurora Serverless v2 容量设置。

根据您在多可用区部署中配置读取器的方式，读取器的容量可以与写入器的容量绑定，也可以独立设置。有关如何执行此操作的详细信息，请参阅 Aurora Serverless v2 扩缩。

监控 Aurora Serverless v2 涉及随时间推移测量数据库集群中的写入器和读取器的容量值。如果数据库没有缩减至最小容量，则可以采取行动，例如调整最小值和优化数据库应用程序。如果数据库持续达到最大容量，则可以采用行动，例如增大最大容量。您还可以优化数据库应用程序，并将查询负载分散到更多读取器中。

Aurora Serverless v2 容量的费用按 ACU 小时数来衡量。有关如何计算 Aurora Serverless v2 费用的信息，请参阅 Aurora 定价页面。

假设集群中写入器和读取器的总数为 *n*。在这种情况下，当您未运行任何数据库操作时，集群会消耗大约 *n* x *minimum ACUs*。Aurora 本身可能会运行监控或维护操作，从而产生少量负载。当数据库以全部容量运行时，该集群消耗不超过 *n* x *maximum ACUs*。

有关选择适当的最小和最大 ACU 值的更多详细信息，请参阅选择 Aurora 集群的 Aurora Serverless v2 容量范围。您指定的最小和最大 ACU 值也会影响 Aurora Serverless v2 的某些 Aurora 配置参数的工作方式。有关容量范围和配置参数之间的相互影响的详细信息，请参阅使用 Aurora Serverless v2 的参数组。

Aurora Serverless v2 扩缩

对于每个 Aurora Serverless v2 写入器或读取器，Aurora 会持续跟踪 CPU、内存和网络等资源的使用率。这些测量统称为负载。负载包括应用程序执行的数据库操作。它还包括数据库服务器的后台处理和 Aurora 管理任务。当容量受到这些因素之一的限制时，Aurora Serverless v2 会扩展。当 Aurora Serverless v2 检测到可以通过扩展来解决的性能问题时，它也会扩展。您可以按照 适用于 Aurora Serverless v2 的重要 Amazon CloudWatch 指标 和 使用性能详情监控 Aurora Serverless v2 性能 中的步骤监控资源利用率以及了解它如何影响 Aurora Serverless v2 扩缩。

数据库集群中的写入器和读取器的负载可能会有所不同。写入器处理所有数据定义语言（DDL）语句，例如 `CREATE TABLE`、`ALTER TABLE` 和 `DROP TABLE`。写入器还会处理所有数据操纵语言（DML）语句，例如 `INSERT` 和 `UPDATE`。读取器可以处理只读语句，例如 `SELECT` 查询。

扩缩是为数据库增大或减少 Aurora Serverless v2 容量的操作。使用 Aurora Serverless v2 时，每个写入器和读取器都有自己的当前容量值，以 ACU 衡量。当写入器或读取器的当前容量太低而无法处理负载时，Aurora Serverless v2 会将其扩展到更高的容量。当写入器或读取器的当前容量高于所需容量时，可以将写入器或读取器缩减至更低的容量。

与 Aurora Serverless v1 不同的是，Aurora Serverless v2 可以在每次数据库集群达到阈值时通过将容量加倍来扩展，从而逐步增加容量。当您的工作负载需求开始达到写入器或读取器的当前数据库容量

时，Aurora Serverless v2 会增加该写入器或读取器的 ACU 数量。Aurora Serverless v2 按所需的增量扩展容量，以便让消耗的资源实现最佳性能。扩缩按小至 0.5 ACU 的增量进行。当前容量越大，扩缩增量就越大，因此可以更快地进行扩缩。

由于 Aurora Serverless v2 扩缩非常频繁、精细且无中断，它不会像 Aurora Serverless v1 那样在 AWS Management Console 中导致离散事件。相反，您可以测量 Amazon CloudWatch 指标（例如 `ServerlessDatabaseCapacity` 和 `ACUUtilization`），并随时间推移跟踪它们的最小值、最大值和平均值。了解有关 Aurora 指标的更多信息，请参阅监控 Amazon Aurora 集群中的指标。有关监控 Aurora Serverless v2 的提示，请参阅适用于 Aurora Serverless v2 的重要 Amazon CloudWatch 指标。

您可以选择使读取器与关联写入器同时扩缩，也可以独立于写入器进行扩缩。您可以通过为该读取器指定提升层来完成扩缩。

- 提升层 0 和 1 中的读取器与写入器同时扩缩。这种扩缩行为使得优先级层 0 和 1 中的读取器非常容易获得。这是因为它们的大小总是调整为适当的容量，以便在失效转移情况下接管来自写入器的工作负载。
- 提升层 2-15 中的读取器可以独立于写入器进行扩缩。每个读取器都保持在您为集群指定的最小和最大 ACU 值范围内。当读取器独立于关联的写入器数据库进行扩缩时，它会变为空闲并缩减，同时写入器继续处理大量事务。如果在较低的提升层中没有其他读取器可用，它仍可作为失效转移目标。但是，如果它被提升为写入器，则可能需要扩展以处理写入器的全部工作负载。

有关提升层的详细信息，请参阅为 Aurora Serverless v2 读取器选择提升层。

Aurora Serverless v1 中的扩缩点和相关的超时时间的概念在 Aurora Serverless v2 中不适用。可以在数据库连接打开、SQL 事务处理正在进行、表已锁定以及临时表正在使用的情况下进行 Aurora Serverless v2 扩缩。Aurora Serverless v2 不会等到安静点才开始扩缩。扩缩不会中断任何正在进行的数据库运营。

如果您的工作负载需要的读取容量超过单个写入器和单个读取器可提供的读取容量，则可以向集群添加多个 Aurora Serverless v2 读取器。每个 Aurora Serverless v2 读取器可以在您为数据库集群指定的最小和最大容量值范围内扩缩。您可以使用集群的读取器端点将只读会话定向到读取器并减少写入器上的负载。

Aurora Serverless v2 是否执行扩缩，以及扩缩在启动后的速度也取决于集群的最小和最大 ACU 设置。此外，它还取决于读取器是配置为随写入器一起扩缩还是独立于写入器进行扩缩。有关影响 Aurora Serverless v2 扩缩的因素的详细信息，请参阅 Aurora Serverless v2 的性能和扩缩。

缩减至零

在最近的 Aurora MySQL 和 Aurora PostgreSQL 版本中，Aurora Serverless v2 写入器和读取器可以完全缩减至零 ACU。我们将此功能称为自动暂停和恢复，或者简称自动暂停。您可以通过为最小容量指定零值或非零值来选择是否允许此行为。您也可以选择 Aurora Serverless v2 实例要等待多长时间才能暂停。有关哪些版本具有此功能的信息，请参阅 Aurora Serverless v2 容量。有关如何有效使用它的信息，请参阅针对 Aurora Serverless v2 缩减到零 ACU 从而自动暂停和恢复。

在较早的 Aurora MySQL 和 Aurora PostgreSQL 版本中，空闲 Aurora Serverless v2 写入器和读取器可以缩减至您为集群指定的最小 ACU 值，但不能完全缩减至零 ACU。在这种情况下，当您设置容量范围时，不能选择零 ACU。这种行为与 Aurora Serverless v1 的行为不同，后者会在闲置一段时间后暂停，然后在打开新连接时花一些时间进行恢复。

当您的具有 Aurora Serverless v2 容量的数据库集群在一段时间内不需要时，您也可以像对待预调配数据库集群那样，先停止整个集群，需要时再启动。这种方式最适合开发和测试系统，因为一次可能不需要很多小时，而且恢复集群的速度并不重要。停止/启动集群功能适用于所有 Aurora Serverless v2 版本。有关此功能的更多信息，请参阅停止和启动 Amazon Aurora 数据库集群。

Aurora Serverless v2 和高可用性

为 Aurora 数据库集群建立高可用性的方法是使其成为多可用区数据库集群。多可用区 Aurora 数据库集群在多个可用区（AZ）中具有始终可用的计算容量。即使在出现重大中断的情况下，该配置也能使数据库保持正常运行。如果出现影响写入器乃至整个可用区的问题，Aurora 会执行自动失效转移。使用 Aurora Serverless v2 时，您可以选择随写入器的容量一起扩展和缩减的备用计算容量。这样，第二个可用区的计算容量随时准备好接管当前的工作负载。同时，当数据库空闲时，所有可用区中的计算容量都可以缩减。有关 Aurora 如何使用 AWS 区域 和可用区的详细信息，请参阅 Aurora 数据库实例的高可用性。

除了写入器，Aurora Serverless v2 多可用区功能还使用读取器。支持读取器是 Aurora Serverless v2 的新功能，Aurora Serverless v1 不支持此功能。您可以向 Aurora 数据库集群添加最多 15 个分布在 3 个可用区的 Aurora Serverless v2 读取器。

对于即使出现影响整个集群或整个 AWS 区域的问题时也必须保持可用的业务关键型应用程序，您可以设置 Aurora Global Database。您可以使用辅助集群中的 Aurora Serverless v2 容量，因此它们准备好可以在灾难恢复期间接管。当数据库不忙时，它们也可以缩减。有关 Aurora Global Database 的详细信息，请参阅使用 Amazon Aurora Global Database。

对于失效转移和其他高可用性功能，Aurora Serverless v2 的工作方式与预置实例类似。有关更多信息，请参阅 Amazon Aurora 的高可用性。

假设您想要确保您的 Aurora Serverless v2 集群获得最大可用性。除了写入器之外，您还可以创建读取器。如果您将读取器分配到提升层 0 或 1，那么无论写入器发生什么样的扩缩，读取器也会进行同样的扩缩。这样，容量完全相同的读取器随时准备好在失效转移的情况下接管写入器。

假设您希望在集群继续处理事务的同时为您的业务运行季度报告。如果您向集群添加一个 Aurora Serverless v2 读取器，并将其分配给 2 至 15 的提升层，则您可以直接连接到该读取器以运行报告。根据报告查询占用内存和 CPU 的程度，该读取器可以扩展以适应工作负载。然后，当报告完成后，它可以再次缩减。

Aurora Serverless v2 和存储

每个 Aurora 数据库集群的存储包含所有数据的六个副本，分布在三个可用区。无论数据库集群除了写入器之外是否还包括任何读取器，此内置数据复制都适用。这样，即使出现影响集群计算容量的问题，您的数据也可保持安全。

Aurora Serverless v2 存储具有与 Amazon Aurora 存储中所述相同的可靠性和持久性特征。这是因为无论计算容量是使用 Aurora Serverless v2 还是使用预置实例，Aurora 数据库集群的存储同样适用。

Aurora 集群的配置参数

您可以按照与调整预置数据库集群相同的方式来调整具有 Aurora Serverless v2 容量的集群的所有集群和数据库配置参数。但是，一些与容量相关的参数的处理方式与 Aurora Serverless v2 不同。在混合配置集群中，您为这些容量相关参数指定的参数值仍适用于所有预置的写入器和读取器。

Aurora Serverless v2 写入器和读取器几乎所有参数的工作方式都与预配实例的工作方式相同。例外情况是 Aurora 在扩缩过程中自动调整的一些参数，以及 Aurora 保持为固定值的一些参数，它们取决于最大容量设置。

例如，为缓冲区缓存预留的内存量随着写入器或读取器的扩展而增加，并随着它们的缩减而减少。这样，当数据库不忙时，可以释放内存。相反，Aurora 会根据最大容量设置自动将最大连接数设置为适当的值。这样，在负载下降且 Aurora Serverless v2 缩减时，活动连接不会断开。有关 Aurora Serverless v2 如何处理特定参数的信息，请参阅使用 Aurora Serverless v2 的参数组。

Aurora Serverless v2 的要求和限制

创建要在其中使用 Aurora Serverless v2 数据库实例的集群时，请注意以下要求和限制：

主题

- 区域和版本可用性

- 使用 Aurora Serverless v2 的集群必须已指定容量范围
- 在 Aurora Serverless v2 中不支持某些预置功能
- Aurora Serverless v2 的某些方面与 Aurora Serverless v1 不同

区域和版本可用性

功能可用性和支持因每个 Aurora 数据库引擎的特定版本以及 AWS 区域 而异。有关 Aurora 和 Aurora Serverless v2 的版本和区域可用性的更多信息，请参阅 支持 Aurora Serverless v2 的区域和 Aurora 数据库引擎。

以下示例显示的 AWS CLI 命令用于确认您可为特定 AWS 区域与 Aurora Serverless v2 配合使用的确切数据库引擎值。Aurora Serverless v2 的 `--db-instance-class` 参数始终为 `db.serverless`。`--engine` 参数可以是 `aurora-mysql` 或 `aurora-postgresql`。替换相应的 `--region` 和 `--engine` 值，以便确认您可以使用的 `--engine-version` 值。如果该命令不生成任何输出，则 Aurora Serverless v2 不适用于 AWS 区域和数据库引擎的这一组合。

```
aws rds describe-orderable-db-instance-options --engine aurora-mysql --db-instance-
class db.serverless \
  --region my_region --query 'OrderableDBInstanceOptions[].[EngineVersion]' --output
 text

aws rds describe-orderable-db-instance-options --engine aurora-postgresql --db-
instance-class db.serverless \
  --region my_region --query 'OrderableDBInstanceOptions[].[EngineVersion]' --output
 text
```

使用 Aurora Serverless v2 的集群必须已指定容量范围

Aurora 集群必须有 `ServerlessV2ScalingConfiguration` 属性，然后才可以添加使用 `db.serverless` 数据库实例类的数据库实例。此属性指定容量范围。Aurora Serverless v2 容量范围从最少 0 个 Aurora 容量单位（ACU）至最多 256 个 ACU，增量为 0.5 ACU。允许的最小值取决于 Aurora 版本。每个 ACU 提供相当于约 2 GiB 的 RAM 以及相关的 CPU 和网络。有关 Aurora Serverless v2 如何使用容量范围设置的详细信息，请参阅 Aurora Serverless v2 的工作原理。

有关各种数据库引擎版本支持的容量范围，请参阅 Aurora Serverless v2 容量。

在创建集群和关联的 Aurora Serverless v2 数据库实例时，您可以在 AWS Management Console 中指定最小和最大 ACU 值。您也可以在 AWS CLI 中指定 `--serverless-`

`v2-scaling-configuration` 选项。或者您可以使用 Amazon RDS API 指定 `ServerlessV2ScalingConfiguration` 参数。您可以在创建集群或修改现有集群时指定此属性。有关设置容量范围的步骤，请参阅为集群设置 Aurora Serverless v2 容量范围。有关如何选择最小和最大容量值以及这些设置如何影响某些数据库参数的详细说明，请参阅选择 Aurora 集群的 Aurora Serverless v2 容量范围。

在 Aurora Serverless v2 中不支持某些预置功能

Amazon Aurora Serverless v2 目前未提供 Aurora 预置数据库实例的以下功能：

- 数据库活动流（DAS）。
- Aurora PostgreSQL 的集群缓存管理。`apg_ccm_enabled` 配置参数不适用于 Aurora Serverless v2 数据库实例。

有一些 Aurora 功能适用于 Aurora Serverless v2，但如果您的容量范围低于特定工作负载下的这些功能的内存需求所需的容量，则可能会导致问题。在这种情况下，数据库可能无法像往常一样运行，或者可能会遇到内存不足错误。有关设置适当容量范围的建议，请参阅选择 Aurora 集群的 Aurora Serverless v2 容量范围。如果您的数据库因容量范围配置错误而遇到内存不足错误，如需故障排除信息，请参阅避免内存不足错误。

不支持 Aurora Auto Scaling。这种类型的扩展会根据 CPU 利用率添加新的读取器来处理额外的读取密集型工作负载。但是，基于 CPU 利用率进行扩展对 Aurora Serverless v2 没有意义。作为替代方法，您可以提前创建 Aurora Serverless v2 读取器数据库实例并将其缩减至低容量。与动态添加新数据库实例相比，这是速度更快且中断更少的扩缩集群读取容量的方法。

Aurora Serverless v2 的某些方面与 Aurora Serverless v1 不同

如果您是 Aurora Serverless v1 用户并且这是您首次使用 Aurora Serverless v2，请参阅 Aurora Serverless v2 和 Aurora Serverless v1 要求之间的差异，以了解 Aurora Serverless v1 和 Aurora Serverless v2 之间的要求有何不同。

创建一个使用 Aurora Serverless v2 的数据库集群

要创建可以在其中添加 Aurora Serverless v2 数据库实例的 Aurora 集群，请按照创建 Amazon Aurora 数据库集群中的相同过程进行操作。借助 Aurora Serverless v2，您的集群可以与预置集群互换。您的集群可以有一些数据库实例使用 Aurora Serverless v2，有些数据库实例为已预置。

主题

- Aurora Serverless v2 数据库集群的设置
- 创建 Aurora Serverless v2 数据库集群
- 创建 Aurora Serverless v2 写入器数据库实例

Aurora Serverless v2 数据库集群的设置

确保集群的初始设置满足 Aurora Serverless v2 的要求和限制中列出的要求。指定以下设置，以确保您可以向集群中添加 Aurora Serverless v2 数据库实例：

AWS 区域

在 AWS 区域中创建可使用 Aurora Serverless v2 数据库实例的集群。有关可用区域的详细信息，请参阅 支持 Aurora Serverless v2 的区域和 Aurora 数据库引擎。

数据库引擎版本

选择与 Aurora Serverless v2 兼容的引擎版本。有关的 Aurora Serverless v2 版本要求的信息，请参阅 Aurora Serverless v2 的要求和限制。

数据库实例类

如果您使用 AWS Management Console 创建集群，则可以同时为写入器数据库实例选择数据库实例类。选择 Serverless（无服务器）数据库实例类。选择该数据库实例类时，还要指定写入器数据库实例的容量范围。同样的容量范围适用于添加到该集群的所有其他 Aurora Serverless v2 数据库实例。

如果未看到数据库实例类的无服务器选项，请确保选择 支持 Aurora Serverless v2 的区域和 Aurora 数据库引擎 支持的数据库引擎版本。

使用 AWS CLI 或 Amazon RDS API 时，为数据库实例类指定的参数为 `db.serverless`。

Capacity range（容量范围）

填入适用于集群中所有数据库实例的最小和最大 Aurora 容量单位（ACU）值。当您为数据库实例类选择 Serverless（无服务器）时，Create cluster（创建集群）和 Add reader（添加读取器）控制台页面上都提供了此选项。

有关各种数据库引擎版本支持的容量范围，请参阅 Aurora Serverless v2 容量。

如果您没有看到最小和最大 ACU 字段，请确保为写入器数据库实例选择了无服务器数据库实例类。

如果您最初使用预置数据库实例创建集群，则不指定最小和最大 ACU。在这种情况下，您可以在之后修改集群以添加该设置。您还可以向集群添加 Aurora Serverless v2 读取器数据库实例。您可以在该过程中指定容量范围。

在为集群指定容量范围之前，不能使用 AWS CLI 或 RDS API 向集群添加任何 Aurora Serverless v2 数据库实例。如果您尝试添加 Aurora Serverless v2 数据库实例，会出现错误。在 AWS CLI 或 RDS API 过程，用 `ServerlessV2ScalingConfiguration` 属性表示容量范围。

对于包含多个读取器数据库实例的集群，各个 Aurora Serverless v2 读取器数据库实例的失效转移优先级在该数据库实例如何扩展和缩减方面起着重要作用。在最初创建集群时无法指定优先级。向集群添加第二个或更多读取器数据库实例时，请记住此属性。有关更多信息，请参阅 为 Aurora Serverless v2 读取器选择提升层。

创建 Aurora Serverless v2 数据库集群

您可以使用 AWS Management Console、AWS CLI 或 RDS API 创建 Aurora Serverless v2 数据库集群。

控制台

创建带有 Aurora Serverless v2 写入器的集群

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Databases（数据库）。
3. 选择创建数据库。在显示的页面上，选择以下选项：
   - 对于引擎类型，选择 Aurora（MySQL 兼容）或 Aurora（PostgreSQL 兼容）。
   - 对于版本，请选择 支持 Aurora Serverless v2 的区域和 Aurora 数据库引擎 支持的版本之一。
4. 对于数据库实例类，请选择 Serverless v2。
5. 对于容量范围，您可以接受默认范围。或者，您可以为最小和最大容量单位选择其他值。您可以从最小 0 ACU 到最大 256 ACU 中选择，增量为 0.5 ACU。

   有关 Aurora Serverless v2 容量单位的更多信息，请参阅 Aurora Serverless v2 容量和 Aurora Serverless v2 的性能和扩缩。

   根据您选择的引擎和版本，上限可能为 128 ACU，下限可能为 0.5 ACU，或两者并用。有关 Aurora 引擎和版本的每种组合的限制的详细信息，请参阅 Aurora Serverless v2 容量。



选择 0 ACU 的最小容量可启用 Aurora Serverless v2 自动暂停和恢复功能。在这种情况下，您可以额外选择 Aurora Serverless v2 数据库实例在没有数据库连接的情况下等待多长时间才会自动暂停。有关自动暂停和恢复功能的信息，请参阅[针对 Aurora Serverless v2 缩减到零 ACU 从而自动暂停和恢复]。



6. 选择任何其他数据库集群设置，如Aurora 数据库集群的设置中所述。
7. 选择创建数据库以创建 Aurora 数据库集群，其中，将 Aurora Serverless v2 数据库实例作为写入器实例，也称为主数据库实例。

CLI

要使用 AWS CLI 创建与 Aurora Serverless v2 数据库实例兼容的数据库集群，请按照创建 Amazon Aurora 数据库集群中的 CLI 程序操作。在 `create-db-cluster` 命令中包含以下参数：

- --region *AWS_Region_where_Aurora Serverless v2_instances_are_available*
- --engine-version *serverless_v2_compatible_engine_version*
- --serverless-v2-scaling-configuration MinCapacity=*minimum_capacity*,MaxCapacity=*maximum_capacity*

以下示例显示如何创建 Aurora Serverless v2 数据库集群。

```
aws rds create-db-cluster \
    --db-cluster-identifier my-serverless-v2-cluster \
    --region eu-central-1 \
    --engine aurora-mysql \
    --engine-version 8.0.mysql_aurora.3.04.1 \
    --serverless-v2-scaling-configuration MinCapacity=1,MaxCapacity=4 \
    --master-username myuser \
    --manage-master-user-password
```

> **Note**
>
> 使用 AWS CLI 创建 Aurora Serverless v2 数据库集群时，引擎模式在输出中显示为 `provisioned`，而不是 `serverless`。`serverless` 引擎模式是指 Aurora Serverless v1。

此示例指定了 `--manage-master-user-password` 选项，用于生成管理密码并在 Secrets Manager 中对其进行管理。有关更多信息，请参阅 使用 Amazon Aurora 和 AWS Secrets Manager 管理密码。或者，您可以使用 `--master-password` 选项自行指定和管理密码。

有关的 Aurora Serverless v2 版本要求的信息，请参阅 Aurora Serverless v2 的要求和限制。有关容量范围允许的数字以及这些数字代表的含义的信息，请参阅 Aurora Serverless v2 容量和 Aurora Serverless v2 的性能和扩缩。

要验证现有集群是否已指定了容量设置，请检查 `ServerlessV2ScalingConfiguration` 属性的 `describe-db-clusters` 命令的输出。该属性看上去类似于以下内容。

```
"ServerlessV2ScalingConfiguration": {
    "MinCapacity": 1.5,
    "MaxCapacity": 24.0
}
```

> **Tip**
>
> 如果在创建集群时未指定最小和最大 ACU，则可以在以后使用 `modify-db-cluster` 命令来添加该设置。在此之前，您不能向集群添加任何 Aurora Serverless v2 数据库实例。如果您尝试添加 `db.serverless` 数据库实例，会出现错误。

API

要使用 RDS API 创建与 Aurora Serverless v2 数据库实例兼容的数据库集群，请按照创建 Amazon Aurora 数据库集群中的 API 程序操作。选择以下设置。确保您的 `CreateDBCluster` 操作包括以下参数：

```
EngineVersion serverless_v2_compatible_engine_version
ServerlessV2ScalingConfiguration with MinCapacity=minimum_capacity and
 MaxCapacity=maximum_capacity
```

有关的 Aurora Serverless v2 版本要求的信息，请参阅 Aurora Serverless v2 的要求和限制。有关容量范围允许的数字以及这些数字代表的含义的信息，请参阅 Aurora Serverless v2 容量和 Aurora Serverless v2 的性能和扩缩。

要检查现有集群是否指定了容量设置，请检查 `ServerlessV2ScalingConfiguration` 属性的 `DescribeDBClusters` 操作。该属性看上去类似于以下内容。

```
"ServerlessV2ScalingConfiguration": {
    "MinCapacity": 1.5,
    "MaxCapacity": 24.0
}
```

> **Tip**
>
> 如果在创建集群时没有指定最小和最大 ACU，则可以在以后使用 `ModifyDBCluster` 操作来添加该设置。在此之前，您不能向集群添加任何 Aurora Serverless v2 数据库实例。如果您尝试添加 `db.serverless` 数据库实例，会出现错误。

创建 Aurora Serverless v2 写入器数据库实例

尽管您在创建 Aurora 集群时指定了 Aurora Serverless v2 容量范围，但您可以为集群中的每个数据库实例选择是使用 Aurora Serverless v2 还是预调配。要立即开始在数据库集群中使用 Aurora Serverless v2，请添加使用 `db.serverless` 实例类的写入器数据库实例。在控制台中，您通常可以在创建数据库集群的工作流程中做出此选择。因此，此过程主要适用于通过 CLI 进行设置时。

控制台

使用 AWS Management Console 创建数据库集群时，您可以同时指定写入器数据库实例的属性。为了让写入器数据库实例使用 Aurora Serverless v2，请选择 Serverless（无服务器）数据库实例类。

然后，您可以通过指定最小和最大 Aurora 容量单位（ACU）值来设置集群的容量范围。这些最小和最大值适用于集群中的每个 Aurora Serverless v2 数据库实例。有关该过程以及最小和最大容量设置的意义，请参阅创建 Aurora Serverless v2 数据库集群。

如果在首次创建集群时没有创建 Aurora Serverless v2 数据库实例，您可以在以后添加一个或多个 Aurora Serverless v2 数据库实例。为此，请按照添加 Aurora Serverless v2 读取器和将预置写入器或读取器转换为 Aurora Serverless v2中的过程进行操作。在将第一个 Aurora Serverless v2 数据库实例添加到集群时指定容量范围。您可以在以后按照 为集群设置 Aurora Serverless v2 容量范围 中的步骤更改容量范围。

CLI

当您使用 AWS CLI 创建 Aurora Serverless v2 数据库集群时，应使用 create-db-instance 命令显式添加写入器数据库实例。包括以下参数：

- `--db-instance-class db.serverless`

下面的示例演示如何创建 Aurora Serverless v2 写入器数据库实例。

```
aws rds create-db-instance \
    --db-cluster-identifier my-serverless-v2-cluster \
    --db-instance-identifier my-serverless-v2-instance \
    --db-instance-class db.serverless \
    --engine aurora-mysql
```

管理 Aurora Serverless v2 数据库集群

借助 Aurora Serverless v2，您的集群可以与预置集群互换。Aurora Serverless v2 属性应用于集群中的一个或多个数据库实例。因此，创建集群、修改集群、创建和恢复快照等过程基本上与其他类型的 Aurora 集群相同。有关管理 Aurora 集群和数据库实例的一般过程，请参阅管理 Amazon Aurora 数据库集群。

在接下来的主题中，您可以了解包含 Aurora Serverless v2 数据库实例的集群的管理注意事项。

主题

- 为集群设置 Aurora Serverless v2 容量范围
- 检查 Aurora Serverless v2 的容量范围
- 添加 Aurora Serverless v2 读取器

- 将预置写入器或读取器转换为 Aurora Serverless v2
- 将 Aurora Serverless v2 写入器或读取器转换为预置
- 为 Aurora Serverless v2 读取器选择提升层
- 将 TLS/SSL 与 Aurora Serverless v2 结合使用
- 查看 Aurora Serverless v2 写入器和读取器
- Aurora Serverless v2 的日志记录

为集群设置 Aurora Serverless v2 容量范围

要修改包含 Aurora Serverless v2 数据库实例的集群或数据库实例本身的配置参数或其他设置，请遵循与预置集群相同的常规过程。有关详细信息，请参阅修改 Amazon Aurora 数据库集群。

Aurora Serverless v2 所特有的最重要设置是容量范围。为 Aurora 集群设置最小和最大 Aurora 容量单位（ACU）值后，无需主动调整集群中的 Aurora Serverless v2 数据库实例容量。Aurora 为您执行调整。此设置在集群级别进行管理。相同的最小和最大 ACU 值适用于集群中的每个 Aurora Serverless v2 数据库实例。

可以设置以下特定值：

- 最小 ACU – Aurora Serverless v2 数据库实例可以将容量减少到这个数量的 ACU。
- 最大 ACU – Aurora Serverless v2 数据库实例可以将容量增大到这个数量的 ACU。

> **Note**
>
> 当您修改 Aurora Serverless v2 数据库集群的容量范围时，无论您是选择立即应用更改，还是在下一个计划维护时段内应用更改，更改都会立即发生。

有关容量范围的影响以及如何监控和微调容量范围的详细信息，请参阅适用于 Aurora Serverless v2 的重要 Amazon CloudWatch 指标和Aurora Serverless v2 的性能和扩缩。您的目标是确保集群的最大容量足够大，可以应对工作负载峰值，而且最低容量足够低，可以在集群不忙碌时最大限度地降低成本。

假设您根据监控情况确定集群的 ACU 范围应该更大、更小、更宽或更窄。您可以使用 AWS Management Console、AWS CLI 或 Amazon RDS API 将 Aurora 集群的容量设置为特定范围的 ACU。此容量范围适用于集群中的每个 Aurora Serverless v2 数据库实例。

例如，假设您的集群的容量范围为 1-16 个 ACU，并包含两个 Aurora Serverless v2 数据库实例。然后，集群作为一个整体消耗介于 2 个 ACU（空闲时）和 32 个 ACU（充分利用时）之间的某个容量。如果您将容量范围从 8 个更改为 20.5 个 ACU，现在集群在空闲时消耗 16 个 ACU，充分利用时最多消耗 41 个 ACU。

Aurora 会根据容量范围内的最大 ACU 值自动设置 Aurora Serverless v2 数据库实例的某些参数值。有关此类参数的列表，请参阅 Aurora Serverless v2 的最大连接数。对于依赖此类计算的静态参数，重启数据库实例时会再次计算该值。因此，您可以在更改容量范围后重启数据库实例来更新此类参数的值。要检查是否需要重启数据库实例以实现此类参数更改，请检查数据库实例的 `ParameterApplyStatus` 属性。值为 `pending-reboot` 表示重新启动会对某些参数值应用更改。

控制台

您可以使用 AWS Management Console 为包含 Aurora Serverless v2 数据库实例的集群设置容量范围。

使用控制台时，您可以在将第一个 Aurora Serverless v2 数据库实例添加到该集群时设置集群的容量范围。创建集群时，如果您为写入器数据库实例选择 Serverless v2（无服务器 v2）数据库实例类，可以设置容量范围。或者，在将 Aurora Serverless v2 读取器数据库实例添加到集群时，如果您选择 Serverless（无服务器）数据库实例类，可以设置容量范围。或者，将集群中现有的预置数据库实例转换为 Serverless（无服务器）数据库实例类时，可以设置容量范围。有关这些过程的完整版本，请参阅创建 Aurora Serverless v2 写入器数据库实例、添加 Aurora Serverless v2 读取器和将预置写入器或读取器转换为 Aurora Serverless v2。

您在集群级别设置的所有容量范围都适用于集群中的所有 Aurora Serverless v2 数据库实例。下图显示了一个具有多个 Aurora Serverless v2 读取器数据库实例的集群。每个实例都有完全相同的容量范围，即 2-64 个 ACU。



修改 Aurora Serverless v2 集群的容量范围

1. 通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Databases（数据库）。

3. 从列表中选择包含 Aurora Serverless v2 数据库实例的集群。该集群必须已包含至少一个 Aurora Serverless v2 数据库实例。否则，Aurora 不会显示 Capacity range（容量范围）部分。
4. 对于 Actions（操作），选择 Modify（修改）。
5. 在 Capacity range（容量范围）部分，选择以下选项：

   a. 在 Minimum ACUs（最小 ACU）中输入一个值。控制台会显示允许的值范围。您可以从 0.5-128 个 ACU 中选择最小容量。您可以从 1-128 个 ACU 中选择最大容量。您能够以 0.5 ACU 为增量调整容量值。

   b. 在 Maximum ACUs（最大 ACU）中输入一个值。此值必须大于或等于 Minimum ACUs（最小 ACU）。控制台会显示允许的值范围。下图显示了该选项。



6. 选择 Continue（继续）。此时将显示 Summary of modifications（修改摘要）页面。
7. 选择立即应用。

   容量修改会立即发生，无论您是选择立即应用更改，还是在下一个计划维护时段内应用更改。

8. 选择 Modify cluster（修改集群）以接受修改摘要。您还可以选择 Back（上一步）以修改更改，或选择 Cancel（取消）以放弃更改。

AWS CLI

使用 AWS CLI，运行 modify-db-cluster AWS CLI 命令，设置打算使用 Aurora Serverless v2 数据库实例的集群的容量。指定 `--serverless-v2-scaling-configuration` 选项。集群可能已经包含一个或多个 Aurora Serverless v2 数据库实例，或者您可以稍后添加数据库实例。`MinCapacity` 和 `MaxCapacity` 字段的有效值包括如下项：

- `0.5`、`1`、`1.5`、`2` 等等，增量为 0.5，最大为 128。

在此示例中，将一个名为 `sample-cluster` 的 Aurora 数据库集群的 ACU 范围设置为至少 `48.5`，最大值为 64。

```
aws rds modify-db-cluster --db-cluster-identifier sample-cluster \
--serverless-v2-scaling-configuration MinCapacity=48.5,MaxCapacity=64
```

容量修改会立即发生，无论您是选择立即应用更改，还是在下一个计划维护时段内应用更改。

完成之后，您可以向集群中添加 Aurora Serverless v2 数据库实例，每个新数据库实例可以在 48.5 至 64 个 ACU 之间扩缩。新的容量范围也适用于群集中已存在的任何 Aurora Serverless v2 数据库实例。如有必要，数据库实例会扩展或缩减，以便处于新的容量范围内。

有关使用 CLI 设置容量范围的其他示例，请参阅选择 Aurora 集群的 Aurora Serverless v2 容量范围。

要使用 Aurora Serverless 修改 AWS CLI 数据库集群的扩展配置，请运行 modify-db-cluster AWS CLI 命令。指定 `--serverless-v2-scaling-configuration` 选项以配置最小容量和最大容量。有效的容量值包括：

- Aurora MySQL：`0.5`、`1`、`1.5`、`2` 等等，增量为 0.5 个 ACU，最大为 128。
- Aurora PostgreSQL：`0.5`、`1`、`1.5`、`2` 等等，增量为 0.5 个 ACU，最大为 128。

在以下示例中，修改名为 `sample-cluster` 的集群中名为 `sample-instance` 的 Aurora Serverless v2 数据库实例的扩缩配置。

对于 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster --db-cluster-identifier sample-cluster \
--serverless-v2-scaling-configuration MinCapacity=8,MaxCapacity=64
```

对于 Windows：

```
aws rds modify-db-cluster --db-cluster-identifier sample-cluster ^
--serverless-v2-scaling-configuration MinCapacity=8,MaxCapacity=64
```

RDS API

您可以使用 ModifyDBCluster API 操作设置 Aurora 数据库实例的容量。指定 `ServerlessV2ScalingConfiguration` 参数。`MinCapacity` 和 `MaxCapacity` 字段的有效值包括如下项：

- 0.5、1、1.5、2 等等，增量为 0.5，最大为 128。

您可以使用 ModifyDBCluster API 操作修改包含 Aurora Serverless v2 数据库实例的集群的扩缩配置。指定 ServerlessV2ScalingConfiguration 参数以配置最小容量和最大容量。有效的容量值包括：

- Aurora MySQL：0.5、1、1.5、2 等等，增量为 0.5 个 ACU，最大为 128。
- Aurora PostgreSQL：0.5、1、1.5、2 等等，增量为 0.5 个 ACU，最大为 128。

容量修改会立即发生，无论您是选择立即应用更改，还是在下一个计划维护时段内应用更改。

检查 Aurora Serverless v2 的容量范围

在检查 Aurora Serverless v2 群集的容量范围过程中，您需要先设置容量范围。如果尚未执行此操作，请按照为集群设置 Aurora Serverless v2 容量范围中的步骤进行操作。

您在集群级别设置的所有容量范围都适用于集群中的所有 Aurora Serverless v2 数据库实例。下图显示了一个具有多个 Aurora Serverless v2 数据库实例的集群。每个实例都有完全相同的容量范围。



您也可以查看集群中的任意 Aurora Serverless v2 数据库实例的详细信息页面。Configuration（配置）选项卡会显示数据库实例的容量范围。



您还可以在集群的 Modify（修改）页面上查看该集群的当前容量范围。下图显示了如何操作。此时，您可以更改容量范围。有关设置或更改容量范围的所有方法，请参阅为集群设置 Aurora Serverless v2 容量范围。



检查 Aurora 集群的当前容量范围

您可以通过检查集群的 `ServerlessV2ScalingConfiguration` 属性来查看为集群中的 Aurora Serverless v2 数据库实例配置的容量范围。以下 AWS CLI 示例显示的集群最小容量为 0.5 个 Aurora 容量单位（ACU），最大容量为 16 个 ACU。

```
$ aws rds describe-db-clusters --db-cluster-identifier serverless-v2-64-acu-cluster \
  --query 'DBClusters[*].[ServerlessV2ScalingConfiguration]'
[
    [
        {
            "MinCapacity": 0.5,
            "MaxCapacity": 16.0
        }
    ]
]
```

添加 Aurora Serverless v2 读取器

要向集群添加 Aurora Serverless v2 读取器数据库实例，请遵循与将 Aurora 副本添加到数据库集群中相同的常规过程。选择新数据库实例的 Serverless v2（无服务器 v2）实例类。

如果读取器数据库实例是集群中的第一个 Aurora Serverless v2 数据库实例，您还可以选择容量范围。下图显示了用于指定最小和最大 Aurora 容量单位（ACU）的控件。此设置适用于此读取器数据库实例以及您添加到集群的任何其他 Aurora Serverless v2 数据库实例。每个 Aurora Serverless v2 数据库实例可以在最小和最大 ACU 值之间进行扩缩。



如果您已经将任何 Aurora Serverless v2 数据库实例添加到集群中，则添加另一个 Aurora Serverless v2 读取器数据库实例会显示当前的容量范围。下图显示了这些只读控件。



如果要更改集群的容量范围，请按照中为集群设置 Aurora Serverless v2 容量范围的步骤进行操作。

对于包含多个读取器数据库实例的集群，各个 Aurora Serverless v2 读取器数据库实例的故障转移优先级在该数据库实例如何扩展和缩减方面起着重要作用。在最初创建集群时无法指定优先级。向集群添加第二个读取器数据库实例或后续数据库实例时，请记住此属性。有关更多信息，请参阅 为 Aurora Serverless v2 读取器选择提升层。

有关查看集群当前容量范围的其他方法，请参阅检查 Aurora Serverless v2 的容量范围。

将预置写入器或读取器转换为 Aurora Serverless v2

您可以将预置数据库实例转换为使用 Aurora Serverless v2。为此，请按照修改数据库集群中的数据库实例中的步骤操作。集群必须满足 Aurora Serverless v2 的要求和限制中的要求。例如，Aurora Serverless v2 数据库实例要求集群运行某些最低引擎版本。

假设您正在转换运行中的预置集群以利用 Aurora Serverless v2。在这种情况下，您可以在转换过程中的第一步将数据库实例转换为 Aurora Serverless v2，从而更大限度地减少停机时间。有关完整过程，请参阅从预置集群切换到 Aurora Serverless v2。

如果您转换的数据库实例是集群中的第一个 Aurora Serverless v2 数据库实例，您可以在执行 Modify（修改）操作时为集群选择容量范围。此容量范围适用于您添加到集群中的每个 Aurora Serverless v2 数据库实例。下图显示了用于指定最小和最大 Aurora 容量单位（ACU）的页面。



有关容量范围意义的详细信息，请参阅 Aurora Serverless v2 容量。

如果集群已包含一个或多个 Aurora Serverless v2 数据库实例，您可以在执行 Modify（修改）操作时查看现有的容量范围。下图显示了该信息面板的示例。



如果要在添加多个 Aurora Serverless v2 数据库实例之后更改集群的容量范围，请按照为集群设置 Aurora Serverless v2 容量范围中的步骤进行操作。

将 Aurora Serverless v2 写入器或读取器转换为预置

您可以将 Aurora Serverless v2 数据库实例转换为预置数据库实例。为此，请按照修改数据库集群中的数据库实例中的步骤操作。选择除 Serverless（无服务器）之外的其他数据库实例类。

如果需要的容量比 Aurora Serverless v2 数据库实例的最大 Aurora 容量单位（ACU）提供的容量更大，您可以将 Aurora Serverless v2 数据库实例转换为预置实例。例如，最大的 db.r5 和 db.r6g 数据库实例类的内存容量比 Aurora Serverless v2 数据库实例可扩展的容量更大。

> **Tip**
>
> 一些较旧的数据库实例类（例如 db.r3 和 db.t2）不适用于您与 Aurora Serverless v2 配合使用的 Aurora 版本。要查看在将 Aurora Serverless v2 数据库实例转换为预置实例时可以使用哪些数据库实例类，请参阅数据库实例类支持的数据库引擎。

如果要将集群的写入器数据库实例从 Aurora Serverless v2 转换为预置实例，您可以按照与从预置集群切换到 Aurora Serverless v2中相反的步骤进行操作。将集群中的其中一个读取器数据库实例从 Aurora Serverless v2 切换为预置。然后执行故障转移以使预置的数据库实例放入写入器。

即使从集群中移除所有 Aurora Serverless v2 数据库实例，之前为集群指定的任何容量范围会保持不变。如果要更改容量范围，您可以修改集群，如为集群设置 Aurora Serverless v2 容量范围中所述。

为 Aurora Serverless v2 读取器选择提升层

对于包含多个 Aurora Serverless v2 数据库实例或预置实例和 Aurora Serverless v2 数据库实例混用的集群，请注意每个 Aurora Serverless v2 数据库实例的提升层设置。此设置控制的 Aurora Serverless v2 数据库实例行为比预置数据库实例更多。

在 AWS Management Console 中，使用 Create database（创建数据库）、Modify instance（修改实例）和 Add reader（添加读取器）页面的 Additional configuration（额外配置）下面的 Failover priority（故障转移优先级）选项来指定此设置。您可以在 Databases（数据库）页面上可选的 Priority tier（优先级层）中看到现有数据库实例的这个属性。您还可以在数据库集群或数据库实例的详细信息页面上看到此属性。

对于预置的数据库实例，0—15 层的选择仅决定在故障转移操作期间 Aurora 选择将读取器数据库实例提升为写入器的顺序。对于 Aurora Serverless v2 读取器数据库实例，层号还决定了数据库实例是扩展以匹配写入器数据库实例的容量，还是根据自己的工作负载单独扩展。第 0 层或第 1 层中的 Aurora Serverless v2 读取器数据库实例保持至少与写入器数据库实例同样高的最小容量。这样，它们就可以准备好在发生故障转移的情况下接管写入器数据库实例。如果写入器数据库实例是预置数据库实例，Aurora 会估算等效的 Aurora Serverless v2 容量。它使用该估计值作为 Aurora Serverless v2 读取器数据库实例的最小容量。

第 2-15 层中的 Aurora Serverless v2 读取器数据库实例对其最小容量没有这样的限制。当它们处于空闲状态时，它们可以缩减到集群的容量范围中指定的最小 Aurora 容量单位（ACU）值。

以下 Linux AWS CLI 示例显示了如何检查集群中所有数据库实例的提升层。对于写入器数据库实例，最后一个字段包含 `True` 值，对于所有读取器数据库实例，该字段包含 `False` 值。

```
$ aws rds describe-db-clusters --db-cluster-identifier promotion-tier-demo \
  --query 'DBClusters[*].DBClusterMembers[*].
[PromotionTier,DBInstanceIdentifier,IsClusterWriter]' \
  --output text

1   instance-192  True
1   tier-01-4840  False
10  tier-10-7425  False
15  tier-15-6694  False
```

以下 Linux AWS CLI 示例显示了如何更改集群中特定数据库实例的提升层。这些命令首先使用新的提升层修改数据库实例。然后，他们等待数据库实例再次变为可用，接着确认数据库实例的新提升层。

```
$ aws rds modify-db-instance --db-instance-identifier instance-192 --promotion-tier 0
$ aws rds wait db-instance-available --db-instance-identifier instance-192
$ aws rds describe-db-instances --db-instance-identifier instance-192 \
  --query '*[].[PromotionTier]' --output text
0
```

有关为不同使用案例指定提升层销级别的更多指导，请参阅 Aurora Serverless v2 扩缩。

将 TLS/SSL 与 Aurora Serverless v2 结合使用

Aurora Serverless v2 可以使用传输层安全性/安全套接字层（TLS/SSL）协议来加密客户端和 Aurora Serverless v2 数据库实例之间的通信。它支持 TLS/SSL 1.0、1.1 和 1.2 版本。有关将 TLS/SSL 与 Aurora 结合使用的一般信息，请参阅与 Aurora MySQL 数据库集群的 TLS 连接。

要了解有关使用 MySQL 客户端连接到 Aurora MySQL 数据库的更多信息，请参阅连接到运行 MySQL 数据库引擎的数据库实例。

Aurora Serverless v2 支持 MySQL 客户端（`mysql`）和 PostgreSQL 客户端（`psql`）可用的所有 TLS/SSL 模式，包括下表中列出的模式。

TLS/SSL 模式的说明	mysql	psql
不使用 TLS/SSL 进行连接。	DISABLED	disable
首先尝试使用 TLS/SSL 进行连接，但如有必要，请回退到非 SSL。	PREFERRED	首选（默认）

TLS/SSL 模式的说明	mysql	psql
使用 TLS/SSL 强制执行。	REQUIRED	require
强制 TLS/SSL 并验证证书颁发机构（CA）。	VERIFY_CA	verify-ca
强制执行 TLS/SSL，验证 CA 并验证 CA 主机名。	VERIFY_IDENTITY	verify-full

Aurora Serverless v2 使用通配符证书。如果您在使用 TLS/SSL 时指定“验证 CA”或“验证 CA 和 CA 主机名”选项，请首先从 Amazon Trust Services 下载 Amazon root CA 1 信任存储。完成此操作后，您可以在客户端命令中识别此 PEM 格式的文件。要使用 PostgreSQL 客户端执行此操作，请执行以下操作。

对于 Linux、macOS 或 Unix：

```
psql 'host=endpoint user=user sslmode=require sslrootcert=amazon-root-CA-1.pem
 dbname=db-name'
```

要了解有关通过 Postgres 客户端使用 Aurora PostgreSQL 数据库的更多信息，请参阅连接到运行 PostgreSQL 数据库引擎的数据库实例。

有关更多连接到 Aurora 数据库集群的一般信息，请参阅 连接到 Amazon Aurora 数据库集群。

支持的用于 Aurora Serverless v2 数据库集群连接的密码套件

通过使用可配置的密码套件，您可以更好地控制数据库连接的安全性。您可以指定想要允许保护客户端与数据库的 TLS/SSL 连接的密码套件列表。使用可配置的密码套件，您现在可以控制数据库服务器接受的连接加密。这样做可以防止使用不安全或不再使用的密码。

基于 Aurora MySQL 的 Aurora Serverless v2 数据库集群支持与 Aurora MySQL 预配的数据库集群相同的密码套件。有关这些密码套件的信息，请参阅配置密码套件以连接到 Aurora MySQL 数据库集群。

基于 Aurora PostgreSQL 的 Aurora Serverless v2 数据库集群支持与 Aurora PostgreSQL 预配的数据库集群相同的密码套件。有关这些密码套件的信息，请参阅配置密码套件以连接到 Aurora PostgreSQL 数据库集群。

查看 Aurora Serverless v2 写入器和读取器

您可以使用查看预置数据库实例的相同方式来查看 Aurora Serverless v2 数据库实例的详细信息。为此，请按照查看 Amazon Aurora 数据库集群中的常规过程进行操作。一个集群可能包含所有 Aurora Serverless v2 数据库实例、所有预置数据库实例或这两类中的一部分实例。

在创建一个或多个 Aurora Serverless v2 数据库实例后，您可以查看哪些数据库实例是 Serverless（无服务器）类型，哪些是 Instance（实例）类型。您还可以查看 Aurora Serverless v2 数据库实例可以使用的最小和最大 Aurora 容量单位（ACU）。每个 ACU 是处理器（CPU）和内存（RAM）容量的组合。此容量范围适用于集群中的每个 Aurora Serverless v2 数据库实例。有关检查集群或集群中的任何 Aurora Serverless v2 数据库实例的容量范围的过程，请参阅检查 Aurora Serverless v2 的容量范围。

在 AWS Management Console 中，在 Databases（数据库）页面中的 Size（大小）栏下面标记 Aurora Serverless v2 数据库实例。预置数据库实例显示数据库实例类的名称，例如 r6g.xlarge。Aurora Serverless 数据库实例为数据库实例类显示 Serverless（无服务器），以及数据库实例的最小和最大容量。例如，您可能会看到 Serverless v2 (4–64 ACUs) [无服务器 v2（4-64 个 ACU）] 或 Serverless v2 (1–40 ACUs) [无服务器 v2（1-40 个 ACU）]。

您可以在控制台中每个 Aurora Serverless v2 数据库实例的 Configuration（配置）选项卡上找到相同信息。例如，您可能会看到如下所示的 Instance type（实例类型）部分。在这里，Instance type（实例类型）值为 Serverless v2（无服务器 v2），Minimum capacity（最小容量）值为 2 ACU (4 GiB)，Maximum capacity（最大容量）值为 64 ACU (128 GiB)。



您可以监控每个 Aurora Serverless v2 数据库实例的容量随时间推移发生的变化。这样，您可以检查每个数据库实例消耗的最小、最大和平均 ACU。您还可以查看数据库实例距离其最小容量或最大容量有多近。要在 AWS Management Console 中查看此类详细信息，请查看数据库实例的 Monitoring（监控）选项卡上的 Amazon CloudWatch 指标图形。有关要查看的指标以及如何解读指标的信息，请参阅适用于 Aurora Serverless v2 的重要 Amazon CloudWatch 指标。

Aurora Serverless v2 的日志记录

要启用数据库日志记录，请使用自定义参数组中的配置参数指定要启用的日志。

对于 Aurora MySQL，您可以启用以下日志。

Aurora MySQL	描述
general_log	创建常规日志。设置为 1 以开启。默认为关闭 (0)。
log_queries_not_using_indexes	将任何查询记录到不使用索引的慢速查询日志中。默认为关闭 (0)。设置为 1 以开启此日志。
long_query_time	防止快速运行的查询记录在慢速查询日志中。可以设置为 0 到 31536000 之间的浮动值。默认值为 0（不活动）。
server_audit_events	要在日志中捕获的事件列表。支持的值有 CONNECT、QUERY、QUERY_DCL、QUERY_DDL、QUERY_DML 和 TABLE。
server_audit_logging	设置为 1 以打开服务器审核日志记录。如果启用此选项，则可以通过在 server_audit_events 参数中列出审核事件来指定要发送到 CloudWatch 的审计事件。
slow_query_log	创建慢速查询日志。设置为 1 以打开慢速查询日志。默认为关闭 (0)。

有关更多信息，请参阅 在 Amazon Aurora MySQL 数据库集群中使用高级审计。

对于 Aurora PostgreSQL，您可以对 Aurora Serverless v2 数据库实例启用以下日志。

Aurora PostgreSQL	描述
log_connections	记录每个成功的连接。
log_disconnections	记录会话结束，包括持续时间。

Aurora PostgreSQL	描述
log_lock_waits	默认值为 0（关闭）。设置为 1 以记录锁定等待。
log_min_duration_statement	语句在记录前运行的最短持续时间（以毫秒为单位）。
log_min_messages	设置记录的消息级别。支持的值为 debug5、debug4、debug3、debug2、debug1、inf 将性能数据记录到 postgres 日志，将值设置为 debug1。
log_temp_files	记录指定千字节 (kB) 以上的临时文件的使用情况。
log_statement	控制被记录的特定 SQL 语句。支持的值有 none、ddl、mod 和 all。默认为 none。

主题

- 使用 Amazon CloudWatch 进行日志记录
- 在 Amazon CloudWatch 中查看 Aurora Serverless v2 日志
- Amazon CloudWatch 所含的监控功能

使用 Amazon CloudWatch 进行日志记录

使用 Aurora Serverless v2 的日志记录中的步骤选择要打开哪些数据库日志之后，您可以选择将哪些日志上传（“发布”）到 Amazon CloudWatch。

您可以使用 Amazon CloudWatch 来分析日志数据、创建警报和查看指标。默认情况下，Aurora Serverless v2 的错误日志已启用并自动上传到 CloudWatch。您还可以将其他日志从 Aurora Serverless v2 数据库实例上传到 CloudWatch。

然后，使用 AWS Management Console中的 Log exports（日志导出）设置，或 AWS CLI 中的 `--enable-cloudwatch-logs-exports` 选项，选择将哪些日志上载到 CloudWatch。

您可以选择将哪些 Aurora Serverless v2 日志上传到 CloudWatch。有关更多信息，请参阅 在 Amazon Aurora MySQL 数据库集群中使用高级审计。

与任何类型的 Aurora 数据库集群一样，您无法修改默认数据库集群参数组。相反，您是根据数据库集群和引擎类型的默认参数来创建自己的数据库集群参数组。我们建议您在创建 Aurora Serverless v2 数据库集群之前创建您的自定义数据库集群参数组，以便在控制台上创建数据库时可以选择该参数组。

> **Note**
>
> 对于 Aurora Serverless v2，您可以同时创建数据库集群和数据库参数组。这与 Aurora Serverless v1 形成对比，在此版本中，您只能创建数据库集群参数组。

在 Amazon CloudWatch 中查看 Aurora Serverless v2 日志

使用 使用 Amazon CloudWatch 进行日志记录中的步骤选择要打开哪些数据库日志之后，您可以查看日志的内容。

有关将 CloudWatch 与 Aurora MySQL 和 Aurora PostgreSQL 日志配合使用的更多信息，请参阅在 Amazon CloudWatch 中监控日志事件和将 Aurora PostgreSQL 日志发布到 Amazon CloudWatch Logs。

要查看 Aurora Serverless v2 数据库集群的日志

1. 通过以下网址打开 CloudWatch 控制台：https://console.aws.amazon.com/cloudwatch/。
2. 选择您的 AWS 区域。
3. 选择 Log groups（日志组）。
4. 从列表中选择 Aurora Serverless v2 数据库集群日志。日志命名模式如下所示。

```
/aws/rds/cluster/cluster-name/log_type
```

> **Note**
>
> 对于 Aurora MySQL 兼容的 Aurora Serverless v2 数据库集群，即使没有错误，错误日志也包括缓冲池扩展事件。

Amazon CloudWatch 所含的监控功能

对于 Aurora Serverless v2，您可以使用 CloudWatch 来监控集群中的所有 Aurora Serverless v2 数据库实例的容量和使用率。您可以查看实例级指标，以便检查在每个 Aurora Serverless

v2 数据库实例扩展和缩减时的容量。您还可以将与容量相关的指标与其他指标进行比较，以了解工作负载的变化如何影响资源消耗。例如，您可以将 `ServerlessDatabaseCapacity` 与 `DatabaseUsedMemory`、`DatabaseConnections` 和 `DMLThroughput` 进行比较，以评估数据库集群在运营期间的响应情况。有关适用于 Aurora Serverless v2 与容量相关的指标的详细信息，请参阅适用于 Aurora Serverless v2 的重要 Amazon CloudWatch 指标。

监控 Aurora Serverless v2 数据库集群的容量

1. 通过以下网址打开 CloudWatch 控制台：https://console.aws.amazon.com/cloudwatch/。
2. 选择 Metrics（指标）。所有可用指标在控制台中显示为卡片，按服务名称进行分组。
3. 选择 RDS。
4. （可选）使用 Search（搜索）框以查找对 Aurora Serverless v2 特别重要的指标：`ServerlessDatabaseCapacity`、`ACUUtilization`、`CPUUtilization` 和 `FreeableMemory`。

我们建议您使用与容量相关的指标设置 CloudWatch 控制面板来监控 Aurora Serverless v2 数据库集群容量。要了解操作方法，请参阅使用 CloudWatch 构建仪表板。

要了解有关将 Amazon CloudWatch 与 Amazon Aurora 结合使用的更多信息，请参阅 将 Amazon Aurora MySQL 日志发布到 Amazon CloudWatch Logs。

Aurora Serverless v2 的性能和扩缩

以下程序和示例说明如何设置 Aurora Serverless v2 集群的容量范围及其关联的数据库实例。您还可以使用以下程序来监控数据库实例的繁忙程度。然后，您可以使用结果来确定是需要向上还是向下调整容量范围。

在使用这些程序之前，请确保熟知 Aurora Serverless v2 扩缩的工作原理。该扩缩机制不同于 Aurora Serverless v1 的扩缩机制。有关更多信息，请参阅 Aurora Serverless v2 扩缩。

目录

- 选择 Aurora 集群的 Aurora Serverless v2 容量范围
  - 选择集群的最小 Aurora Serverless v2 容量设置
  - 选择集群的最大 Aurora Serverless v2 容量设置
  - 示例：更改 Aurora MySQL 集群的 Aurora Serverless v2 容量范围
  - 示例：更改 Aurora PostgreSQL 集群的 Aurora Serverless v2 容量范围

- 使用 Aurora Serverless v2 的参数组
  - 默认参数值
  - Aurora Serverless v2 的最大连接数
  - Aurora 在 Aurora Serverless v2 扩展和缩减时调整的参数
  - Aurora 基于 Aurora Serverless v2 最大容量计算的参数
- 避免内存不足错误
- 适用于 Aurora Serverless v2 的重要 Amazon CloudWatch 指标
  - Aurora Serverless v2 指标如何适用于 AWS 账单
  - Aurora Serverless v2 指标的 CloudWatch 命令示例
- 使用性能详情监控 Aurora Serverless v2 性能
- Aurora Serverless v2 容量问题疑难解答

选择 Aurora 集群的 Aurora Serverless v2 容量范围

对于 Aurora Serverless v2 数据库实例，您可以在添加第一个 Aurora Serverless v2 数据库实例到数据库集群的时候设置适用于数据库集群中所有数据库实例的容量范围。有关执行此操作的程序，请参阅为集群设置 Aurora Serverless v2 容量范围。

您还可以更改现有集群的容量范围。以下部分更详细地介绍了如何选择适当的最小值和最大值，以及在更改容量范围时会发生什么。例如，更改容量范围会修改某些配置参数的默认值。应用所有参数更改需要重新启动每个 Aurora Serverless v2 数据库实例。

主题

- 选择集群的最小 Aurora Serverless v2 容量设置
- 选择集群的最大 Aurora Serverless v2 容量设置
- 示例：更改 Aurora MySQL 集群的 Aurora Serverless v2 容量范围
- 示例：更改 Aurora PostgreSQL 集群的 Aurora Serverless v2 容量范围

选择集群的最小 Aurora Serverless v2 容量设置

最好是始终为最小 Aurora Serverless v2 容量设置选择 0.5。该值允许数据库实例在完全空闲时缩减至最小容量，同时保持活动状态。您也可以通过指定最小容量为 0 ACU 来启用自动暂停行为，如针对 Aurora Serverless v2 缩减到零 ACU 从而自动暂停和恢复中所述。但是，根据您使用该集群的方式和配置的其他设置，采用一个不同的最小容量可能更有效。选择最小容量设置时，请考虑以下因素：

- Aurora Serverless v2 数据库实例的扩缩率取决于其当前容量。当前容量越大，扩展速度就越快。如果您需要数据库实例快速扩展到非常高的容量，请考虑将最小容量设置为其扩缩率可满足要求的值。
- 如果您通常在预期工作负载特别高或特别低的情况下修改数据库实例的数据库实例类，则可以利用这个经验粗略估计等效的 Aurora Serverless v2 容量范围。要确定在低流量时使用的内存大小，请参阅适用于 Aurora 的数据库实例类的硬件规格。

  例如，假设您在集群的工作负载较低时使用 db.r6g.xlarge 数据库实例类。该数据库实例类具有 32 GiB 的内存。因此，您可以将最小 Aurora 容量单位（ACU）设置指定为 16，从而设置一个可以缩减至大约是这个相同容量的 Aurora Serverless v2 数据库实例。那是因为每个 ACU 对应大约 2 GiB 的内存。如果 db.r6g.xlarge 数据库实例有时未充分利用，您可以指定一个稍低的值，以便进一步缩减数据库实例。
- 如果当数据库实例在缓冲区缓存中有一定量的数据时，应用程序的运行效率最高，请考虑指定一个最小 ACU 设置，让内存足够大，可以容纳经常访问的数据。否则，当 Aurora Serverless v2 数据库实例缩减至较小的内存大小时，一些数据会从缓冲区缓存中移出。然后，随着时间的推移，当数据库实例再扩展回来时，信息会读回到缓冲区缓存中。如果将数据带回缓冲区缓存的 I/O 量很大，则选择更高的最小 ACU 值可能会更有效。
- 如果您的 Aurora Serverless v2 数据库实例大部分时间都以特定容量运行，请考虑指定低于该基准但不要太低的最小容量设置。Aurora Serverless v2当前容量没有远低于所需容量时，Aurora Serverless v2 数据库实例可以最有效地估计扩展的规模和速度。
- 如果您的预置工作负载的内存要求对于 T3 或 T4g 等小型数据库实例类而言太高，请选择可提供与 R5 或 R6g 数据库实例的内存相当的最低 ACU 设置。

  特别是，我们建议在使用指定功能时使用以下最低容量（这些建议可能会发生变化）：

  - 性能详情– 2 ACU
  - Aurora Global Database – 8 ACU（仅适用于主要 AWS 区域）
- 在 Aurora 中，复制发生在存储层，因此读取器容量不会直接影响复制。但是，对于独立扩展的 Aurora Serverless v2 读取器数据库实例，请确保在有密集写入操作期间，最低容量足以处理工作负载，从而避免查询延迟。如果提升层 2-15 的读取器数据库实例遇到性能问题，请考虑增大集群的最小容量。有关选择读取器数据库实例是随写入器扩缩还是独立扩缩的详细信息，请参阅为 Aurora Serverless v2 读取器选择提升层。
- 如果您的数据库集群包含 Aurora Serverless v2 读取器数据库实例，则当读取器的提升层不是 0 或 1 时，读取器不会随写入器数据库实例一起扩展。在这种情况下，设置较低的最小容量会导致复制滞后过大。这是因为在数据库忙碌时，读取器可能没有足够的容量来应用来自写入器的更改。建议您将最小容量设置为一个值，该值表示与写入器数据库实例相当的内存量和 CPU 量。

- Aurora Serverless v2 数据库实例的 `max_connections` 参数的值基于从最大 ACU 得出的内存大小。但是，当您在 PostgreSQL 兼容的数据库实例上指定最小容量为 0 或 0.5 ACU 时，`max_connections` 最大值的上限为 2,000。

  如果您打算将 Aurora PostgreSQL 集群用于重要连接工作负载，请考虑使用值为 1 或更高的最低 ACU 设置。有关 Aurora Serverless v2 如何处理 `max_connections` 配置参数的详细信息，请参阅 Aurora Serverless v2 的最大连接数。
- Aurora Serverless v2 数据库实例从最小容量扩展到最大容量所花的时间取决于其最小和最大 ACU 值之间的差异。与数据库实例从小容量开始的情况相比，当数据库实例的当前容量很大时，Aurora Serverless v2 会以更大的增量扩展。因此，如果您指定了相对较大的最大容量，且数据库实例的大部分运行时间都接近该容量，则请考虑增大最小 ACU 设置。这样，空闲数据库实例可以更快地扩展回到最大容量。

选择集群的最大 Aurora Serverless v2 容量设置

最好是始终为最大 Aurora Serverless v2 容量设置选择一些较高的值。较大的最大容量让数据库实例可以在运行密集型工作负载时最大限度地扩展。使用较小的值可以避免产生意外费用。根据您使用该集群的方式以及配置的其他设置，最有效的值可能会比原先想象的值更高或更低。选择最大容量设置时，请考虑以下因素：

- 最大容量必须至少与最小容量相同。可以将最小容量和最大容量设置为完全相同。但是，在这种情况下，容量永远不会扩展或缩减。因此，除了在测试状况下，对最小和最大容量使用完全相同的值并不合适。
- 最大容量必须高于 0.5 ACU。在大多数情况下，可以将最小容量和最大容量设置为相同。但是，您不能将最小值和最大值同时指定为 0.5。请为最大容量使用 1 或更高的值。
- 如果您通常在预期工作负载特别高或特别低的情况下修改数据库实例的数据库实例类，则可以利用这个经验估计等效的 Aurora Serverless v2 容量范围。要确定在高流量时使用的内存大小，请参阅 适用于 Aurora 的数据库实例类的硬件规格。

  例如，假设您在集群的工作负载较高时使用 db.r6g.4xlarge 数据库实例类。该数据库实例类具有 128 GiB 的内存。因此，您可以将最大 ACU 设置指定为 64，从而设置一个可以扩展至大约是这个相同容量的 Aurora Serverless v2 数据库实例。那是因为每个 ACU 对应大约 2 GiB 的内存。如果 db.r6g.4xlarge 数据库实例有时没有足够的容量来有效处理工作负载，则可以指定一个稍高的值来让数据库实例进一步扩展。
- 如果您的数据库使用有预算上限，请选择一个即使所有 Aurora Serverless v2 数据库实例始终以最大容量运行时仍能保持在该上限内的值。请记住，如果您的集群中有 n 个 Aurora Serverless v2 数据

库实例，则该集群在任何时刻可以使用的理论最大 Aurora Serverless v2 容量是 n 乘以集群的最大 ACU 设置。（例如，如果一些读取器独立于写入器进行扩展，则实际消耗量可能更少。）

- 如果您使用 Aurora Serverless v2 读取器数据库实例从写入器数据库实例中分流一些只读工作负载，您可以选择较低的最大容量设置。这样做是为了反映这一点，每个读取器数据库实例不需要像集群只包含一个数据库实例那样扩展得那么高。
- 假设您想防止由于应用程序中的数据库参数配置错误或低效查询而导致过度使用。在这种情况下，您可以通过选择一个最大容量设置，使之低于可以设置的绝对最高容量，从而避免意外过度使用。
- 如果由于实际用户活动造成的峰值很少见但确实会发生，那么在选择最大容量设置时可以考虑这些情况。如果优先级是使应用程序以全面的性能和可扩展性保持运行，则可以指定比正常使用情况下的容量更大的最大容量设置。如果在非常极端的活动高峰期间应用程序可以降低吞吐量运行，则可以选择稍低的最大容量设置。确保选择的设置仍然有足够的内存和 CPU 资源以使应用程序保持运行。
- 如果您在集群中启用了增加每个数据库实例内存使用量的设置，请在决定最大 ACU 值时将该内存考虑在内。此类设置包括性能详情、Aurora MySQL 并行查询、Aurora MySQL Performance Schema 和 Aurora MySQL 二进制日志复制的设置。确保最大 ACU 值允许 Aurora Serverless v2 数据库实例扩展到足以在使用这些功能时处理工作负载。有关排除由于低最大 ACU 设置和增加内存开销的 Aurora 功能组合而引起的问题的信息，请参阅避免内存不足错误。

示例：更改 Aurora MySQL 集群的 Aurora Serverless v2 容量范围

以下 AWS CLI 示例显示了如何更新现有 Aurora MySQL 集群中的 Aurora Serverless v2 数据库实例的 ACU 范围。最初，集群的容量范围为 8-32 ACU。

```
aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query 'DBClusters[*].ServerlessV2ScalingConfiguration|[0]'
{
    "MinCapacity": 8.0,
    "MaxCapacity": 32.0
}
```

数据库实例处于空闲状态，并缩减至 8 ACU。此时，以下与容量相关的设置适用于数据库实例。为了使用简单易读的单位表示缓冲池的大小，我们将它除以 2 的 30 次方，得出以 GiB 为单位的测量值。这是因为 Aurora 的内存相关测量值使用 2 的幂，而不是 10 的幂为单位。

```
mysql> select @@max_connections;
+-------------------+
| @@max_connections |
+-------------------+
|              3000 |
```

```
+--------------------+
1 row in set (0.00 sec)

mysql> select @@innodb_buffer_pool_size;
+----------------------------+
| @@innodb_buffer_pool_size |
+----------------------------+
|                9294577664 |
+----------------------------+
1 row in set (0.00 sec)

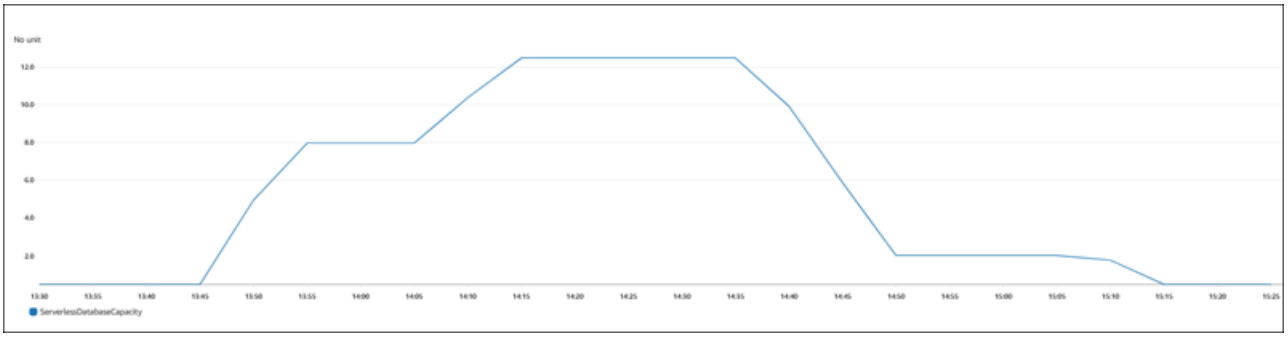
mysql> select @@innodb_buffer_pool_size / pow(2,30) as gibibytes;
+-----------+
| gibibytes |
+-----------+
|   8.65625 |
+-----------+
1 row in set (0.00 sec)
```

接下来，我们更改集群的容量范围。在 modify-db-cluster 命令完成后，集群的 ACU 范围为 12.5-80。

```
aws rds modify-db-cluster --db-cluster-identifier serverless-v2-cluster \
  --serverless-v2-scaling-configuration MinCapacity=12.5,MaxCapacity=80

aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query 'DBClusters[*].ServerlessV2ScalingConfiguration|[0]'
{
    "MinCapacity": 12.5,
    "MaxCapacity": 80.0
}
```

更改容量范围会导致某些配置参数的默认值发生更改。Aurora 可以立即应用其中一些新的默认值。但是，某些参数更改只有在重新启动后才会生效。pending-reboot 状态表示需要重新启动才能应用一些参数更改。

```
aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query '*[].{DBClusterMembers:DBClusterMembers[*].
{DBInstanceIdentifier:DBInstanceIdentifier,DBClusterParameterGroupStatus:DBClusterParameterGrou
[0]'
{
    "DBClusterMembers": [
```

```
        {
            "DBInstanceIdentifier": "serverless-v2-instance-1",
            "DBClusterParameterGroupStatus": "pending-reboot"
        }
    ]
}
```

此时，集群处于空闲状态且数据库实例 serverless-v2-instance-1 正在使用 12.5 ACU。innodb_buffer_pool_size 参数将根据数据库实例的当前容量进行调整。max_connections 参数仍然反映以前的最大容量的值。重置该值需要重新启动数据库实例。

> **Note**
>
> 如果您直接在自定义数据库参数组中设置 max_connections 参数，则无需重启。

```
mysql> select @@max_connections;
+-------------------+
| @@max_connections |
+-------------------+
|              3000 |
+-------------------+
1 row in set (0.00 sec)

mysql> select @@innodb_buffer_pool_size;
+---------------------------+
| @@innodb_buffer_pool_size |
+---------------------------+
|               15572402176 |
+---------------------------+
1 row in set (0.00 sec)

mysql> select @@innodb_buffer_pool_size / pow(2,30) as gibibytes;
+---------------+
| gibibytes     |
+---------------+
| 14.5029296875 |
+---------------+
1 row in set (0.00 sec)
```

现在，我们重新启动数据库实例，并等待实例再次变为可用。

```
aws rds reboot-db-instance --db-instance-identifier serverless-v2-instance-1
{
  "DBInstanceIdentifier": "serverless-v2-instance-1",
  "DBInstanceStatus": "rebooting"
}

aws rds wait db-instance-available --db-instance-identifier serverless-v2-instance-1
```

已清除 pending-reboot 状态。in-sync 值确认 Aurora 已应用所有待处理的参数更改。

```
aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query '*[].{DBClusterMembers:DBClusterMembers[*].
{DBInstanceIdentifier:DBInstanceIdentifier,DBClusterParameterGroupStatus:DBClusterParameterGrou
[0]'
{
    "DBClusterMembers": [
        {
            "DBInstanceIdentifier": "serverless-v2-instance-1",
            "DBClusterParameterGroupStatus": "in-sync"
        }
    ]
}
```

对于空闲数据库实例，innodb_buffer_pool_size 参数已增大至最终大小。max_connections 参数已增大，以便反映从最大 ACU 值派生的值。当内存大小增加一倍时，Aurora 用于 max_connections 的公式会导致增加 1,000。

```
mysql> select @@innodb_buffer_pool_size;
+---------------------------+
| @@innodb_buffer_pool_size |
+---------------------------+
|               16139681792 |
+---------------------------+
1 row in set (0.00 sec)

mysql> select @@innodb_buffer_pool_size / pow(2,30) as gibibytes;
+-----------+
| gibibytes |
+-----------+
|  15.03125 |
+-----------+
1 row in set (0.00 sec)
```

```
mysql> select @@max_connections;
+-------------------+
| @@max_connections |
+-------------------+
|              4000 |
+-------------------+
1 row in set (0.00 sec)
```

我们将容量范围设置为 0.5–128 ACU，然后重启数据库实例。现在，空闲数据库实例的缓冲区缓存大小不到 1 GiB，因此我们以 MiB 为单位来衡量它。max_connections 值 5000 来源于最大容量设置的内存大小。

```
mysql> select @@innodb_buffer_pool_size / pow(2,20) as mebibytes, @@max_connections;
+-----------+-------------------+
| mebibytes | @@max_connections |
+-----------+-------------------+
|       672 |              5000 |
+-----------+-------------------+
1 row in set (0.00 sec)
```

示例：更改 Aurora PostgreSQL 集群的 Aurora Serverless v2 容量范围

以下 CLI 示例显示如何更新现有 Aurora PostgreSQL 集群中 Aurora Serverless v2 数据库实例的 ACU 范围。

1. 集群的容量范围以 0.5–1 ACU 开始。
2. 将容量范围更改为 8–32 ACU。
3. 将容量范围更改为 12.5–80 ACU。
4. 将容量范围更改为 0.5–128 ACU。
5. 将容量恢复到其初始范围 0.5–1 ACU。

下图显示了 Amazon CloudWatch 中的容量变化。



数据库实例处于空闲状态，并缩减至 0.5 ACU。此时，以下与容量相关的设置适用于数据库实例。

```
postgres=> show max_connections;
 max_connections
-----------------
 189
(1 row)

postgres=> show shared_buffers;
 shared_buffers
----------------
 16384
(1 row)
```

接下来，我们更改集群的容量范围。在 `modify-db-cluster` 命令完成后，集群的 ACU 范围为 8.0–32。

```
aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query 'DBClusters[*].ServerlessV2ScalingConfiguration|[0]'
{
    "MinCapacity": 8.0,
    "MaxCapacity": 32.0
}
```

更改容量范围会导致某些配置参数的默认值发生更改。Aurora 可以立即应用其中一些新的默认值。但是，某些参数更改只有在重新启动后才会生效。`pending-reboot` 状态表示需要重新启动才能应用一些参数更改。

```
aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query '*[].{DBClusterMembers:DBClusterMembers[*].
{DBInstanceIdentifier:DBInstanceIdentifier,DBClusterParameterGroupStatus:DBClusterParameterGrou
[0]'
{
```

```
    "DBClusterMembers": [
        {
            "DBInstanceIdentifier": "serverless-v2-instance-1",
            "DBClusterParameterGroupStatus": "pending-reboot"
        }
    ]
}
```

此时，集群处于空闲状态且数据库实例 serverless-v2-instance-1 正在使用 8.0 ACU。shared_buffers 参数将根据数据库实例的当前容量进行调整。max_connections 参数仍然反映以前的最大容量的值。重置该值需要重新启动数据库实例。

> **Note**
>
> 如果您直接在自定义数据库参数组中设置 max_connections 参数，则无需重启。

```
postgres=> show max_connections;
 max_connections
-----------------
 189
(1 row)

postgres=> show shared_buffers;
 shared_buffers
----------------
 1425408
(1 row)
```

我们重启数据库实例，并等待实例再次变为可用。

```
aws rds reboot-db-instance --db-instance-identifier serverless-v2-instance-1
{
  "DBInstanceIdentifier": "serverless-v2-instance-1",
  "DBInstanceStatus": "rebooting"
}

aws rds wait db-instance-available --db-instance-identifier serverless-v2-instance-1
```

现在数据库实例已重新启动，pending-reboot 状态已清除。in-sync 值确认 Aurora 已应用所有待处理的参数更改。

```
aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query '*[].{DBClusterMembers:DBClusterMembers[*].
{DBInstanceIdentifier:DBInstanceIdentifier,DBClusterParameterGroupStatus:DBClusterParameterGrou
[0]'
{
    "DBClusterMembers": [
        {
            "DBInstanceIdentifier": "serverless-v2-instance-1",
            "DBClusterParameterGroupStatus": "in-sync"
        }
    ]
}
```

重启后，max_connections 显示新的最大容量中的值。

```
postgres=> show max_connections;
 max_connections
-----------------
 5000
(1 row)

postgres=> show shared_buffers;
 shared_buffers
----------------
 1425408
(1 row)
```

接下来，我们将集群的容量范围更改为 12.5–80 ACU。

```
aws rds modify-db-cluster --db-cluster-identifier serverless-v2-cluster \
  --serverless-v2-scaling-configuration MinCapacity=12.5,MaxCapacity=80

aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query 'DBClusters[*].ServerlessV2ScalingConfiguration|[0]'
{
    "MinCapacity": 12.5,
    "MaxCapacity": 80.0
}
```

此时，集群处于空闲状态且数据库实例 serverless-v2-instance-1 正在使用 12.5 ACU。shared_buffers 参数将根据数据库实例的当前容量进行调整。max_connections 值仍为 5000。

```
postgres=> show max_connections;
 max_connections
-----------------
 5000
(1 row)

postgres=> show shared_buffers;
 shared_buffers
----------------
 2211840
(1 row)
```

我们再次重启，但参数值保持不变。这是因为，对于运行 Aurora PostgreSQL 的 Aurora Serverless v2 数据库集群，max_connections 的最大值为 5000。

```
postgres=> show max_connections;
 max_connections
-----------------
 5000
(1 row)

postgres=> show shared_buffers;
 shared_buffers
----------------
 2211840
(1 row)
```

现在，我们将容量范围设置为 0.5 - 128 ACU。数据库集群缩减到 10 ACU，然后缩减到 2 个。我们重启数据库实例。

```
postgres=> show max_connections;
 max_connections
-----------------
 2000
(1 row)

postgres=> show shared_buffers;
 shared_buffers
----------------
 16384
(1 row)
```

Aurora Serverless v2 数据库实例的 max_connections 值基于从最大 ACU 得出的内存大小。但是，当您在 PostgreSQL 兼容的数据库实例上指定最小容量为 0 或 0.5 ACU 时，max_connections 最大值的上限为 2,000。

现在，我们将容量恢复到其初始范围 0.5–1 ACU，然后重启数据库实例。max_connections 参数已恢复为其原始值。

```
postgres=> show max_connections;
 max_connections
-----------------
 189
(1 row)

postgres=> show shared_buffers;
 shared_buffers
----------------
 16384
(1 row)
```

使用 Aurora Serverless v2 的参数组

当您创建您的 Aurora Serverless v2 数据库集群时，您可以选择特定 Aurora 数据库引擎和关联的数据库集群参数组。如果您不熟悉 Aurora 如何使用参数组在集群之间一致地应用配置设置，请参阅Amazon Aurora 的参数组。所有这些用于创建、修改、应用参数组和其他操作的过程都适用于 Aurora Serverless v2。

参数组功能在预置集群和包含 Aurora Serverless v2 数据库实例的集群中的工作方式通常相同：

- 集群中所有数据库实例的默认参数值由集群参数组定义。
- 您可以通过为特定数据库实例指定自定义数据库参数组来覆盖这些数据库实例的一些参数。您可以在特定数据库实例的调试或性能优化时执行此操作。例如，假设您有一个集群包含一些 Aurora Serverless v2 数据库实例和一些预置数据库实例。在这种情况下，您可以使用自定义数据库参数组为预置数据库实例指定一些不同的参数。
- 对于 Aurora Serverless v2，您可以使用在参数组的 SupportedEngineModes 属性中具有 provisioned 值的所有参数。在 Aurora Serverless v1 中，您只可以使用在 SupportedEngineModes 属性中具有 serverless 值的参数的子集。

主题

- 默认参数值
- Aurora Serverless v2 的最大连接数
- Aurora 在 Aurora Serverless v2 扩展和缩减时调整的参数
- Aurora 基于 Aurora Serverless v2 最大容量计算的参数

默认参数值

预置数据库实例和 Aurora Serverless v2 数据库实例之间的关键区别是，Aurora 会覆盖与数据库实例容量相关的某些参数的任何自定义参数值。自定义参数值仍适用于集群中的任何预置数据库实例。有关 Aurora Serverless v2 数据库实例如何解读 Aurora 参数组中的参数的更多信息，请参阅Aurora 集群的配置参数。对于 Aurora Serverless v2 覆盖的具体参数，请参阅 Aurora 在 Aurora Serverless v2 扩展和缩减时调整的参数和 Aurora 基于 Aurora Serverless v2 最大容量计算的参数。

通过使用 describe-db-cluster-parameters CLI 命令并查询 AWS 区域，您可以获得各种 Aurora 数据库引擎的默认参数组的默认值列表。以下是您可为与 Aurora Serverless v2 兼容的引擎版本的 `--db-parameter-group-family` 和 `-db-parameter-group-name` 选项使用的值。

数据库引擎和版本	参数组系列	默认参数组名称
Aurora MySQL 版本 3	`aurora-mysql8.0`	`default.aurora-mysql8.0`
Aurora PostgreSQL 版本 13.x	`aurora-postgresql13`	`default.aurora-postgresql13`
Aurora PostgreSQL 版本 14.x	`aurora-postgresql14`	`default.aurora-postgresql14`
Aurora PostgreSQL 版本 15.x	`aurora-postgresql15`	`default.aurora-postgresql15`
Aurora PostgreSQL 版本 16.x	`aurora-postgresql16`	`default.aurora-postgresql16`
Aurora PostgreSQL 版本 17.x	`aurora-postgresql17`	`default.aurora-postgresql17`

以下示例从 Aurora MySQL 版本 3 和 Aurora PostgreSQL 13 的默认数据库集群组中获取参数列表。这些是您与 Aurora Serverless v2 配合使用的 Aurora MySQL 和 Aurora PostgreSQL 版本。

对于 Linux、macOS 或 Unix：

```
aws rds describe-db-cluster-parameters \
  --db-cluster-parameter-group-name default.aurora-mysql8.0 \
  --query 'Parameters[*].
{ParameterName:ParameterName,SupportedEngineModes:SupportedEngineModes} |
    [?contains(SupportedEngineModes, `provisioned`) == `true`] | [*].[ParameterName]' \
  --output text

aws rds describe-db-cluster-parameters \
  --db-cluster-parameter-group-name default.aurora-postgresql13 \
  --query 'Parameters[*].
{ParameterName:ParameterName,SupportedEngineModes:SupportedEngineModes} |
    [?contains(SupportedEngineModes, `provisioned`) == `true`] | [*].[ParameterName]' \
  --output text
```

对于 Windows：

```
aws rds describe-db-cluster-parameters ^
  --db-cluster-parameter-group-name default.aurora-mysql8.0 ^
  --query 'Parameters[*].
{ParameterName:ParameterName,SupportedEngineModes:SupportedEngineModes} |
    [?contains(SupportedEngineModes, `provisioned`) == `true`] | [*].[ParameterName]' ^
  --output text

aws rds describe-db-cluster-parameters ^
  --db-cluster-parameter-group-name default.aurora-postgresql13 ^
  --query 'Parameters[*].
{ParameterName:ParameterName,SupportedEngineModes:SupportedEngineModes} |
    [?contains(SupportedEngineModes, `provisioned`) == `true`] | [*].[ParameterName]' ^
  --output text
```

Aurora Serverless v2 的最大连接数

对于 Aurora MySQL 和 Aurora PostgreSQL，Aurora Serverless v2 数据库实例使 `max_connections` 参数保持不变，所以在数据库实例缩减时不会断开连接。此参数的默认值源自基于数据库实例的内存大小的公式。有关公式和预置数据库实例类的默认值的详细信息，请参阅 至 Aurora MySQL 数据库实例的最大连接数和 至 Aurora PostgreSQL 数据库实例的最大连接数。

当 Aurora Serverless v2 计算公式时，它使用的内存大小基于数据库实例的最大 Aurora 容量单位（ACU），而不是当前 ACU 值。如果您更改默认值，我们建议使用该公式的变体，而不是指定常数值。这样，Aurora Serverless v2 可以根据最大容量使用适当的设置。

当您更改 Aurora Serverless v2 数据库集群的最大容量时，必须重启 Aurora Serverless v2 数据库实例才能更新 `max_connections` 值。这是因为 `max_connections` 是 Aurora Serverless v2 的一个静态参数。

下表显示了适用于 Aurora Serverless v2 的 `max_connections` 基于最大 ACU 值的默认值。

最大 ACU 数	Aurora MySQL 上的默认最大连接数	Aurora PostgreSQL 上的默认最大连接数
1	90	189
4	135	823
8	1000	1669
16	2000	3360
32	3000	5000
64	4,000	5000
128	5000	5000
192	6000	5000
256	6000	5000

> **Note**
>
> Aurora Serverless v2 数据库实例的 `max_connections` 值基于从最大 ACU 得出的内存大小。但是，当您在 PostgreSQL 兼容的数据库实例上指定最小容量为 0 或 0.5 ACU 时，`max_connections` 最大值的上限为 2,000。

有关显示 max_connections 如何随最大 ACU 值而变化的具体示例，请参阅 示例：更改 Aurora MySQL 集群的 Aurora Serverless v2 容量范围 和 示例：更改 Aurora PostgreSQL 集群的 Aurora Serverless v2 容量范围。

Aurora 在 Aurora Serverless v2 扩展和缩减时调整的参数

在弹性伸缩期间，Aurora Serverless v2 需要能够更改每个数据库实例的参数，以便在容量增加或减少时发挥最佳作用。因此，您无法覆盖与容量相关的某些参数。对于一些可以覆盖的参数，请避免直接写入固定值。以下注意事项适用于与容量相关的这些设置。

对于 Aurora MySQL，Aurora Serverless v2 在扩缩期间动态调整部分参数的大小。对于以下参数，Aurora Serverless v2 不使用您指定的任何自定义参数值：

- innodb_buffer_pool_size
- innodb_purge_threads
- table_definition_cache
- table_open_cache

对于 Aurora PostgreSQL，Aurora Serverless v2 在扩缩期间动态调整以下参数的大小。对于以下参数，Aurora Serverless v2 不使用您指定的任何自定义参数值：

- shared_buffers

对于除此处列出的参数以外的所有参数，Aurora Serverless v2 数据库实例与预调配数据库实例的工作方式相同。从集群参数组继承默认参数值。您可以使用自定义集群参数组修改整个集群的默认值。或者，您可以使用自定义数据库参数组修改某些数据库实例的默认值。动态参数会立即更新。对静态参数的更改仅在重新启动数据库实例后才会生效。

Aurora 基于 Aurora Serverless v2 最大容量计算的参数

对于以下参数，Aurora PostgreSQL 使用基于最大 ACU 设置的内存大小得出的默认值，与 max_connections 相同：

- autovacuum_max_workers
- autovacuum_vacuum_cost_limit
- autovacuum_work_mem
- effective_cache_size

- maintenance_work_mem

避免内存不足错误

如果您的其中一个 Aurora Serverless v2 数据库实例持续达到最大容量的限制，Aurora 会通过将数据库实例设置为 incompatible-parameters 状态来表明这种状况。数据库实例处于 incompatible-parameters 状态时，某些操作会被阻止。例如，您无法升级引擎版本。

通常，当由于内存不足错误而频繁重新启动时，数据库实例会进入此状态。发生这种类型的重启时，Aurora 会记录一个事件。您可以按照 查看 Amazon RDS 事件 中的步骤查看事件。由于开启性能详情和 IAM 身份验证等设置会产生开销，所以通常会出现高内存使用率。数据库实例上存在繁重的工作负载，或者要管理与大量架构对象相关联的元数据时，也会导致出现高内存使用率。

如果内存压力变得更低，以致数据库实例不会经常达到最大容量，Aurora 会自动将数据库实例状态更改回 available。

要从这种情况中恢复，您可以采取以下部分或全部操作：

- 通过更改集群的最小 Aurora 容量单位（ACU）值，提高 Aurora Serverless v2 数据库实例的容量下限。这样做可以避免空闲数据库缩减到某个容量，以至于内存太少，无法满足集群中启用的功能所需的内存。更改集群的 ACU 设置后，重新启动 Aurora Serverless v2 数据库实例。这样做可计算出 Aurora 是否可以将状态重置回 available。
- 通过更改集群的最大 ACU 值，提高 Aurora Serverless v2 数据库实例的容量上限。这样做可避免繁忙的数据库无法扩展到某个容量，以至于没有足够的内存来满足集群中启用的功能和数据库工作负载所需的内存。更改集群的 ACU 设置后，重新启动 Aurora Serverless v2 数据库实例。这样做可计算出 Aurora 是否可以将状态重置回 available。
- 关闭需要内存开销的配置设置。例如，假设您已启用 AWS Identity and Access Management（IAM）、性能详情或 Aurora MySQL 二进制日志复制等功能，但未使用它们。如果是这样，您可以关闭这些功能。或者，您可以将集群的最小和最大容量值调高，以便考虑到这些功能使用的内存。有关选择最小和最大容量设置的指引，请参阅选择 Aurora 集群的 Aurora Serverless v2 容量范围。
- 减少数据库实例上的工作负载。例如，您可以将读取器数据库实例添加到集群中，以便将只读查询的负载分散到更多数据库实例中。
- 调优应用程序使用的 SQL 代码，以使用更少的资源。例如，您可以检查查询计划、检查慢速查询日志或调整表的索引。您还可以执行其他传统类型的 SQL 优化。

适用于 Aurora Serverless v2 的重要 Amazon CloudWatch 指标

要开始为您的 Aurora Serverless v2 数据库实例使用 Amazon CloudWatch，请参阅在 Amazon CloudWatch 中查看 Aurora Serverless v2 日志。要详细了解如何通过 CloudWatch 监控 Aurora 数据库集群 ，请参阅在 Amazon CloudWatch 中监控日志事件。

您可以在 CloudWatch 中查看 Aurora Serverless v2 数据库实例，以便使用 `ServerlessDatabaseCapacity` 指标监控每个数据库实例使用的容量。您还可以监控所有标准 Aurora CloudWatch 指标，例如 `DatabaseConnections` 和 `Queries`。有关可为 Aurora 监控的 CloudWatch 指标的完整列表，请参阅Amazon Aurora 的 Amazon CloudWatch 指标。这些指标分为集群级和实例级指标，请分别参阅Amazon Aurora 的集群级指标和Amazon Aurora 的实例级指标。

为了了解 Aurora Serverless v2 数据库实例如何扩展和缩减，重要的是监控以下 CloudWatch 实例级指标。所有这些指标每秒计算一次。这样，您就可以监控 Aurora Serverless v2 数据库实例的当前状态。您可以设置警报，以便在任何 Aurora Serverless v2 数据库实例接近与容量相关的指标阈值时通知您。您可以确定最小和最大容量设置是否合适，或者是否需要调整它们。您可以确定将精力集中在哪方面来优化数据库的效率。

- `ServerlessDatabaseCapacity`。作为实例级指标，它会报告当前数据库实例容量所代表的 ACU 数量。作为集群级指标，它代表集群中所有 Aurora Serverless v2 数据库实例的 `ServerlessDatabaseCapacity` 值的平均值。在 Aurora Serverless v1 中，此指标只是一个集群级指标。在 Aurora Serverless v2 中，它可以在数据库实例级和集群级使用。
- `ACUUtilization`。此指标是 Aurora Serverless v2 中的新指标。此值以百分比表示。它的计算方法是 `ServerlessDatabaseCapacity` 指标的值除以数据库集群的最大 ACU 值。解读此指标并采取行动时考虑以下准则：
  - 如果此指标接近值 `100.0`，则数据库实例已扩展到它能达到的最大容量。考虑增大集群的最大 ACU 设置。这样，写入器和读取器数据库实例都可以扩展到更高的容量。
  - 假设只读工作负载导致读取器数据库实例的 `100.0` 属性接近 `ACUUtilization`，而写入器数据库实例未接近其最大容量。在这种情况下，请考虑向集群添加额外的读取器数据库实例。这样，您可以将工作负载的只读部分分散到更多数据库实例上，从而减少每个读取器数据库实例的负载。
  - 假设您正在运行生产应用程序，其中性能和可扩展性是主要考虑因素。在这种情况下，您可以将集群的最大 ACU 值设置为较高的数字。您的目标是让 `ACUUtilization` 指标始终低于 `100.0`。使用较高的最大 ACU 值，您可以确信有足够的空间来应对数据库活动中出现的意外高峰。您只需为实际使用的数据库容量付费。
- `CPUUtilization`。这个指标在 Aurora Serverless v2 中的解读不同于预置数据库实例中的解读。对于 Aurora Serverless v2，此值是一个百分比，其计算方法是当前使用的 CPU 量除以在数据库

集群的最大 ACU 值基础上提供的 CPU 容量。当数据库实例持续使用其 CPU 容量的很大一部分时，Aurora 会自动监控此值并扩展 Aurora Serverless v2 数据库实例。

如果此指标接近值 `100.0`，则该数据库实例已达到其最大 CPU 容量。考虑增大集群的最大 ACU 设置。如果读取器数据库实例上的这个指标接近值 `100.0`，请考虑向集群添加额外的读取器数据库实例。这样，您可以将工作负载的只读部分分散到更多数据库实例上，从而减少每个读取器数据库实例的负载。

- `FreeableMemory`。此值表示当 Aurora Serverless v2 数据库实例已扩展到其最大容量时，提供的未使用内存量。对于当前容量低于最大容量的每个 ACU，此值增加大约 2 GiB。因此，在数据库实例扩展到它可以达到的最大值之前，此指标不会接近零。

  如果此指标接近值 `0`，则数据库实例已扩展到它可以达到的最大值，并且已接近其可用内存的限制。考虑增大集群的最大 ACU 设置。如果读取器数据库实例上的这个指标接近值 `0`，请考虑向集群添加额外的读取器数据库实例。这样，工作负载的只读部分可以分散到更多数据库实例上，从而减少每个读取器数据库实例的内存使用。

- `TempStorageIOPS`。在连接到数据库实例的本地存储上完成的 IOPS 数。它包括读取和写入的 IOPS。此指标表示计数，每秒测量一次。这是一个 Aurora Serverless v2 的新指标。有关更多信息，请参阅 Amazon Aurora 的实例级指标。
- `TempStorageThroughput`。传入或传出与数据库实例关联的本地存储的数据量。此指标表示字节数，每秒测量一次。这是一个 Aurora Serverless v2 的新指标。有关更多信息，请参阅 Amazon Aurora 的实例级指标。

通常情况下，Aurora Serverless v2 数据库实例的大多数扩展是因内存使用和 CPU 活动而引起。`TempStorageIOPS` 和 `TempStorageThroughput` 指标可以帮助您诊断数据库实例和本地存储设备之间传输的网络活动导致意外容量增加的罕见情况。要监控其他网络活动，您可以使用以下现有指标：

- `NetworkReceiveThroughput`
- `NetworkThroughput`
- `NetworkTransmitThroughput`
- `StorageNetworkReceiveThroughput`
- `StorageNetworkThroughput`
- `StorageNetworkTransmitThroughput`

您可让 Aurora 将部分或所有数据库日志发布到 Amazon CloudWatch Logs。有关说明，请根据您的数据库引擎参阅以下内容：

- the section called “将 Aurora PostgreSQL 日志发布到 CloudWatch Logs”
- the section called “将 Aurora MySQL 日志发布到 CloudWatch Logs”

Aurora Serverless v2 指标如何适用于 AWS 账单

AWS 账单上的 Aurora Serverless v2 费用基于您可以监控的相同 `ServerlessDatabaseCapacity` 指标进行计算。如果使用 Aurora Serverless v2 容量不到一个小时，计费机制可能与此指标的已计算 CloudWatch 平均值不同。如果系统问题导致 CloudWatch 指标在短时间内不可用，计费机制也可能会有所不同。因此，如果您自己根据 `ServerlessDatabaseCapacity` 平均值进行计算，可能会在账单上看到 ACU 小时值略有不同。

Aurora Serverless v2 指标的 CloudWatch 命令示例

以下 AWS CLI 示例演示了如何监控与 Aurora Serverless v2 相关的最重要 CloudWatch 指标。在每种情况下，请使用您自己的 Aurora Serverless v2 数据库实例替换 `--dimensions` 参数的 `Value=` 字符串。

以下 Linux 示例显示了数据库实例的最小容量、最大容量和平均容量值，在一小时内每 10 分钟测量一次。Linux `date` 命令指定相对于当前日期和时间的开始时间和结束时间。`--query` 参数中的 `sort_by` 函数根据 `Timestamp` 字段按时间顺序对结果进行排序。

```
aws cloudwatch get-metric-statistics --metric-name "ServerlessDatabaseCapacity" \
  --start-time "$(date -d '1 hour ago')" --end-time "$(date -d 'now')" --period 600 \
  --namespace "AWS/RDS" --statistics Minimum Maximum Average \
  --dimensions Name=DBInstanceIdentifier,Value=my_instance \
  --query 'sort_by(Datapoints[*].
{min:Minimum,max:Maximum,avg:Average,ts:Timestamp},&ts)' --output table
```

以下 Linux 示例显示了监控集群中每个数据库实例的容量。它们衡量每个数据库实例的最小容量、最大容量和平均容量利用率。在三个小时内，每小时进行一次测量。这些示例使用 `ACUUtilization` 指标表示 ACU 上限的百分比，而不是使用 `ServerlessDatabaseCapacity` 表示固定数量的 ACU。这样，您就不需要知道容量范围内的最小和最大 ACU 值的实际数字。您可以看到 0 到 100 之间的百分比。

```
aws cloudwatch get-metric-statistics --metric-name "ACUUtilization" \
```

```
  --start-time "$(date -d '3 hours ago')" --end-time "$(date -d 'now')" --period 3600 \
  --namespace "AWS/RDS" --statistics Minimum Maximum Average \
  --dimensions Name=DBInstanceIdentifier,Value=my_writer_instance \
  --query 'sort_by(Datapoints[*].
{min:Minimum,max:Maximum,avg:Average,ts:Timestamp},&ts)' --output table

aws cloudwatch get-metric-statistics --metric-name "ACUUtilization" \
  --start-time "$(date -d '3 hours ago')" --end-time "$(date -d 'now')" --period 3600 \
  --namespace "AWS/RDS" --statistics Minimum Maximum Average \
  --dimensions Name=DBInstanceIdentifier,Value=my_reader_instance \
  --query 'sort_by(Datapoints[*].
{min:Minimum,max:Maximum,avg:Average,ts:Timestamp},&ts)' --output table
```

下面的 Linux 示例执行的测量与之前的示例类似。在本例中，测量值针对 CPUUtilization 指标。在 1 小时内，每 10 分钟进行一次测量。这些数字表示已使用的可用 CPU的百分比，基于数据库实例的最大容量设置中可用的 CPU 资源。

```
aws cloudwatch get-metric-statistics --metric-name "CPUUtilization" \
  --start-time "$(date -d '1 hour ago')" --end-time "$(date -d 'now')" --period 600 \
  --namespace "AWS/RDS" --statistics Minimum Maximum Average \
  --dimensions Name=DBInstanceIdentifier,Value=my_instance \
  --query 'sort_by(Datapoints[*].
{min:Minimum,max:Maximum,avg:Average,ts:Timestamp},&ts)' --output table
```

下面的 Linux 示例执行的测量与之前的示例类似。在本例中，测量值针对 FreeableMemory 指标。在 1 小时内，每 10 分钟进行一次测量。

```
aws cloudwatch get-metric-statistics --metric-name "FreeableMemory" \
  --start-time "$(date -d '1 hour ago')" --end-time "$(date -d 'now')" --period 600 \
  --namespace "AWS/RDS" --statistics Minimum Maximum Average \
  --dimensions Name=DBInstanceIdentifier,Value=my_instance \
  --query 'sort_by(Datapoints[*].
{min:Minimum,max:Maximum,avg:Average,ts:Timestamp},&ts)' --output table
```

使用性能详情监控 Aurora Serverless v2 性能

您可以使用性能详情来监控 Aurora Serverless v2 数据库实例的性能。有关性能详情的程序，请参阅[在 Amazon Aurora 上使用性能详情监控数据库负载]。

以下新性能详情计数器适用于 Aurora Serverless v2 数据库实例：

- os.general.serverlessDatabaseCapacity – 数据库实例以 ACU 表示的当前容量。该值对应于数据库实例的 ServerlessDatabaseCapacity CloudWatch 指标。
- os.general.acuUtilization – 当前容量占最大配置容量的百分比。该值对应于数据库实例的 ACUUtilization CloudWatch 指标。
- os.general.maxConfiguredAcu – 您为此 Aurora Serverless v2 数据库实例配置的最大容量。它用 ACU 来测量。
- os.general.minConfiguredAcu – 您为此 Aurora Serverless v2 数据库实例配置的最小容量。它用 ACU 来测量。

有关性能详情计数器的完整列表，请参阅 性能详情计数器指标。

在性能详情中为 Aurora Serverless v2 数据库实例显示 vCPU 值时，这些值表示基于数据库实例的 ACU 值的估计值。默认时间间隔为 1 分钟，任何小数 vCPU 值将向上舍入到最接近的整数。对于更长的时间间隔，显示的 vCPU 值是每分钟的整数 vCPU 值的平均值。

Aurora Serverless v2 容量问题疑难解答

在某些情况下，即使数据库上没有负载，Aurora Serverless v2 也不会缩减至最小容量。这可能是由于以下原因之一：

- 某些功能可以增加资源使用量并防止数据库缩减到最低容量。这些功能如下所示：
  - Aurora 全局数据库
  - 导出 CloudWatch Logs
  - 在与 Aurora PostgreSQL 兼容的数据库集群上启用 pg_audit
  - 增强监控
  - 性能详情

  有关更多信息，请参阅 选择集群的最小 Aurora Serverless v2 容量设置。
- 如果读取器实例未缩减到最小值，并保持在与写入器实例相同或更高的容量，请检查读取器实例的优先级层。层为 0 或 1 的 Aurora Serverless v2 读取器数据库实例保持在至少与写入器数据库实例一样高的最小容量。将读取器的优先级层更改为 2 或更高，这样它就可以独立于写入器而纵向扩展和缩减。有关更多信息，请参阅 为 Aurora Serverless v2 读取器选择提升层。
- 将影响共享内存大小的任何数据库参数设置为其默认值。将值设置为高于默认值会增加共享内存需求，并防止数据库缩减到最小容量。示例包括 max_connections 和 max_locks_per_transaction。

> **Note**
>
> 更新共享内存参数需要重新启动数据库才能使更改生效。

- 繁重的数据库工作负载可能会增加资源使用量。
- 较大的数据库卷可能会增加资源使用量。

  Amazon Aurora 使用内存和 CPU 资源进行数据库集群管理。Aurora 需要更多的 CPU 和内存来管理具有更大数据库卷的数据库集群。如果您的集群的最小容量低于集群管理所需的最小容量，则您的集群不会缩减到最小容量。
- 后台进程（如清除）也可能增加资源使用量。

如果数据库仍无法缩减到配置的最小容量，则停止并重新启动数据库，以回收可能随着时间推移而累积的所有内存碎片。停止和启动数据库会导致停机，因此我们建议谨慎执行此操作。

针对 Aurora Serverless v2 缩减到零 ACU 从而自动暂停和恢复

如果 Aurora Serverless v2 数据库实例在指定时间段内没有任何由用户活动启动的连接，则您可以指定数据库实例缩减到零 ACU 并自动暂停。为此，您可以将数据库集群的最小 ACU 值指定为零。当实例处于暂停状态时，您无需为实例容量付费。对于使用量较少或长时间处于非活动状态的 Aurora 集群，启用自动暂停和恢复功能可以帮助您管理数据库实例集的成本。

> **Note**
>
> 自动暂停功能同时适用于 Aurora PostgreSQL 和 Aurora MySQL 中的 Aurora Serverless v2。您可能需要升级 Aurora 数据库引擎版本才能利用此功能。有关哪些引擎版本中提供最小容量 0 ACU 的信息，请参阅 Aurora Serverless v2 容量。

主题

- Aurora Serverless v2 自动暂停功能概述
- Aurora Serverless v2 自动暂停功能的先决条件和限制
- 开启和关闭自动暂停功能
- Aurora Serverless v2 自动暂停功能的工作原理
- Aurora Serverless v2 自动暂停如何适用于不同类型的 Aurora 集群
- 监控使用自动暂停功能的 Aurora 集群
- Aurora Serverless v2 自动暂停故障排查
- Aurora Serverless v2 自动暂停功能的应用程序设计注意事项

Aurora Serverless v2 自动暂停功能概述

Aurora Serverless v2 数据库实例可以在没有用户连接的一段时间后自动暂停，并在连接请求到达时自动恢复。Aurora Serverless v2 自动暂停/恢复功能有助于管理那些没有严格服务级别目标（SLO）的系统的成本。例如，您可以为用于开发和测试的集群启用此功能，或者为在数据库恢复时可以接受短暂停顿的内部应用程序启用此功能。如果您的工作负载有时处于非活动状态，并且可以容忍在实例恢复时略微连接延迟，请考虑对您的 Aurora Serverless v2 实例使用自动暂停以降低成本。

您可以通过指定集群中的 Aurora Serverless v2 数据库实例能否自动暂停，以及每个实例在暂停之前必须处于空闲状态的时长来控制这种行为。要为 Aurora 集群中的所有 Aurora Serverless v2 数据库实例启用自动暂停行为，请将集群的最小容量值设置为零 ACU。

如果您以前使用过在闲置一段时间后缩减到零 ACU 的 Aurora Serverless v1 功能，则可以升级到 Aurora Serverless v2 并使用其相应的自动暂停功能。

自动暂停功能可以带来节省成本的好处，类似于使用停止/启动集群功能。Aurora Serverless v2 的自动暂停功能的另一个好处是，恢复速度比启动已停止的集群更快，并且可以自动判断何时暂停和恢复每个数据库实例。

自动暂停功能还为控制集群内计算资源的成本提供了更高的精度。您可以允许集群中一些读取器实例暂停，同时使集群中的写入器实例和其他读取器始终处于活动状态。方法是，为可以独立于其他实例暂停的读取器实例分配介于 2-15 范围内的失效转移优先级。

写入器实例和所有失效转移优先级为 0 和 1 的读取器实例始终同时暂停和恢复。因此，该组中的实例会在指定时间间隔内没有任何连接后暂停。

Aurora 数据库集群可以包含写入器和读取器数据库实例以及预调配实例和 Aurora Serverless v2 数据库实例的组合。因此，要有效使用此功能，了解自动暂停机制的以下方面会很有帮助：

- 数据库实例可能自动暂停的情况。
- 何时可能阻止数据库实例暂停。例如，在集群上启用某些 Aurora 功能或执行某些类型的操作可能会阻止实例暂停，即使与这些实例没有任何连接也是如此。
- 实例暂停期间对监控和计费的影响。
- 哪些操作会导致数据库实例恢复处理。
- 数据库实例的容量在暂停和恢复事件发生前后如何变化。
- 如何控制数据库实例暂停前的空闲间隔。
- 如何对应用程序逻辑进行编码，以处理数据库实例恢复处理的时段。

Aurora Serverless v2 自动暂停功能的先决条件和限制

在使用自动暂停功能之前，请检查要使用的引擎版本。此外，请检查自动暂停是否可以与您打算使用的其他 Aurora 功能结合使用。如果您使用的引擎版本不支持自动暂停，则无法开启它。对于不兼容的功能，如果将它们与自动暂停结合使用，您不会收到任何错误。如果集群使用任何不兼容的功能或设置，则 Aurora Serverless v2 实例不会自动暂停。

- 如果您使用的是 Aurora PostgreSQL，则运行的数据库引擎版本必须至少是 16.3、15.7、14.12 或 13.15。
- 如果您使用的是 Aurora MySQL，则运行的数据库引擎版本必须是 3.08.0 或以上。

- 有关引擎版本和提供此功能的 AWS 区域的完整列表，请参阅支持 Aurora Serverless v2 的区域和 Aurora 数据库引擎。
- 当 Aurora Serverless v2 实例恢复时，其容量可能会低于暂停时的容量。有关详细信息，请参阅Aurora Serverless v2 和 Aurora Serverless v1 在自动暂停行为上的区别。

某些情况或设置会阻止 Aurora Serverless v2 实例自动暂停。有关更多信息，请参阅 Aurora Serverless v2 不会自动暂停的情况。

开启和关闭自动暂停功能

您可以在集群级别开启和关闭自动暂停功能。为此，请使用与调整集群的最小容量和最大容量时相同的过程。自动暂停功能通过 0 ACU 最小容量表示。

主题

- 为集群中的 Aurora Serverless v2 实例开启自动暂停
- 可配置的 Aurora Serverless v2 自动暂停超时间隔
- 恢复自动暂停的 Aurora Serverless v2 实例
- 关闭集群中 Aurora Serverless v2 实例的自动暂停

为集群中的 Aurora Serverless v2 实例开启自动暂停

按照为集群设置 Aurora Serverless v2 容量范围中的程序操作。对于最小容量，请选择 0 ACU。当您选择的最小容量为 0 ACU 时，您还可以指定实例在自动暂停之前处于空闲状态的时间长度。

以下 CLI 示例显示了如何创建一个启用自动暂停功能且自动暂停间隔设置为十分钟（600 秒）的 Aurora 集群。

```
aws rds create-db-cluster \
    --db-cluster-identifier my-serverless-v2-cluster \
    --region eu-central-1 \
    --engine aurora-mysql \
    --engine-version 8.0 \
    --serverless-v2-scaling-configuration
 MinCapacity=0,MaxCapacity=4,SecondsUntilAutoPause=600 \
    --master-username myuser \
    --manage-master-user-password
```

以下 CLI 示例显示了如何为现有 Aurora 集群开启自动暂停功能。此示例将自动暂停间隔设置为一小时（3600 秒）。

```
aws rds modify-db-cluster --db-cluster-identifier serverless-v2-cluster \
  --serverless-v2-scaling-configuration
 MinCapacity=0,MaxCapacity=80,SecondsUntilAutoPause=3600

aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query 'DBClusters[*].ServerlessV2ScalingConfiguration|[0]'
{
    "MinCapacity": 0,
    "MaxCapacity": 80.0,
    "SecondsUntilAutoPause": 3600
}
```

可配置的 Aurora Serverless v2 自动暂停超时间隔

超时间隔在 Aurora 集群的 `ServerlessV2ScalingConfiguration` 属性中表示。如果最小容量设置为零 ACU，则可以在创建或修改 Aurora 集群时在 AWS Management Console中指定此间隔。当创建或修改 Aurora 集群时，可以通过使用 `--serverless-v2-scaling-configuration` 参数在 AWS CLI 中指定它。当创建或修改 Aurora 集群时，可以通过使用 `ServerlessV2ScalingConfiguration` 参数在 RDS API 中指定它。

您可以设置的最小间隔为 300 秒（5 分钟）。如果您未指定间隔，将采用默认值。您可以设置的最大间隔为 86,400 秒（一天）。

假设您通过将一个集群的最小容量更改为非零值来关闭该集群的自动暂停功能。在这种情况下，将从 `ServerlessV2ScalingConfiguration` 属性中移除间隔属性。缺少该属性也是在额外确认该集群的自动暂停功能已关闭。如果您稍后重新开启自动暂停功能，则届时可以再次指定任何自定义间隔。

恢复自动暂停的 Aurora Serverless v2 实例

- 当您连接到一个已暂停的 Aurora Serverless v2 实例时，它会自动恢复并接受连接。
- 不包含有效凭证的连接尝试仍会导致数据库实例恢复。
- 如果您通过写入器端点进行连接，Aurora 会恢复写入器数据库实例（如果它处于自动暂停状态）。同时，Aurora 会恢复所有自动暂停且失效转移优先级为 0 或 1 的读取器实例，这意味着它们的容量与写入器实例的容量挂钩。
- 如果您通过读取器端点进行连接，Aurora 会随机选择一个读取器实例。如果该读取器实例已暂停，Aurora 会恢复它。同样，Aurora 首先恢复写入器实例，因为只要有任何读取器实例处于活动状

态，则写入器实例肯定始终处于活动状态。当 Aurora 恢复该写入器实例时，这也会导致失效转移提升层级零和一中的所有读取器实例恢复。

- 如果您通过 RDS 数据 API 向集群发送请求，Aurora 将恢复写入器实例（如果它被暂停）。然后 Aurora 处理数据 API 请求。
- 当您更改数据库集群参数组中配置参数的值时，Aurora 会自动恢复使用该集群参数组的所有集群中任何已暂停的 Aurora Serverless v2 实例。同样，当您更改数据库参数组中的参数值时，Aurora 会自动恢复使用该数据库参数组的所有已暂停 Aurora Serverless v2 实例。当您修改集群以分配不同的集群参数组，或者修改实例以分配不同的数据库参数组时，同样的自动恢复行为也适用。
- 如果 Aurora Serverless v2 写入器实例已暂停，则执行回溯请求会自动恢复该写入器实例。Aurora 会在写入器实例恢复后处理回溯请求。您可以回溯到 Aurora Serverless v2 实例暂停的时间。
- 拍摄集群快照或删除快照不会导致任何 Aurora Serverless v2 实例恢复。
- 创建 Aurora 克隆会导致 Aurora 恢复正在克隆的集群的写入器实例。
- 如果暂停的实例在完成恢复之前收到大量连接请求，则某些会话可能无法连接。建议实施重试逻辑，以便连接到包含一些启用了自动暂停的 Aurora Serverless v2 实例的 Aurora 集群。例如，您可以重试任何失败的连接三次。
- Aurora 可以在不唤醒实例的情况下执行某些类型的次要内部维护。但是，在集群维护时段内发生的某些类型的维护确实需要 Aurora 恢复实例。维护完成后，如果在指定的间隔之后没有其他活动，则实例将再次自动暂停。

> **Note**
>
> Aurora 不会为了特定于引擎的计划作业（例如 PostgreSQL `pg_cron` 扩展或 MySQL 事件调度器中的作业）恢复暂停的实例。为确保此类作业运行，请在预定时间之前手动启动与实例的连接。在数据库实例暂停期间，Aurora 不会对计划时间发生的任何作业进行排队。当实例稍后恢复时，会跳过此类作业。

- 如果 Aurora 集群在 Aurora Serverless v2 实例自动暂停时进行失效转移，Aurora 可能会先恢复一个实例，然后将该实例提升为写入器。如果在实例暂停期间从集群中移除一个或多个数据库实例，也会发生同样的情况。在这种情况下，实例在恢复后立即成为写入器。
- 更改集群属性的操作还会导致任何自动暂停的 Aurora Serverless v2 实例恢复。例如，自动暂停的实例会因以下操作而恢复：
  - 更改集群的扩展范围。
  - 升级集群的引擎版本。
  - 描述或下载来自已暂停实例的日志文件。可以通过向 CloudWatch 上传日志并通过 CloudWatch 分析日志，来检查已暂停实例的历史日志数据。

关闭集群中 Aurora Serverless v2 实例的自动暂停

按照为集群设置 Aurora Serverless v2 容量范围中的程序操作。对于最小容量，请选择 0.5 或更大的值。关闭自动暂停功能后，实例空闲的时间间隔将被重置。如果再次开启自动暂停，则需要指定新的超时间隔。

以下 CLI 示例显示了如何关闭现有 Aurora 集群的自动暂停功能。`describe-db-clusters` 输出显示，当最小容量设置为非零值时，`SecondsUntilAutoPause` 属性被移除。

```
aws rds modify-db-cluster --db-cluster-identifier serverless-v2-cluster \
  --serverless-v2-scaling-configuration MinCapacity=2,MaxCapacity=80

aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query 'DBClusters[*].ServerlessV2ScalingConfiguration|[0]'
{
    "MinCapacity": 2,
    "MaxCapacity": 80.0
}
```

Aurora Serverless v2 自动暂停功能的工作原理

您可以使用以下信息来计划使用自动暂停功能。了解实例暂停、恢复或保持活跃状态的情况可以帮助您在可用性、响应能力和成本节约之间进行权衡。

主题

- 当 Aurora Serverless v2 实例暂停时会发生什么情况
- 当自动暂停的 Aurora Serverless v2 实例恢复时会发生什么情况
- 自动暂停的 Aurora Serverless v2 集群的实例计费方法
- Aurora Serverless v2 不会自动暂停的情况
- 自动暂停如何与集群停止/启动功能配合使用
- 自动暂停的 Aurora Serverless v2 集群的维护和升级工作原理
- Aurora Serverless v2 和 Aurora Serverless v1 在自动暂停行为上的区别

当 Aurora Serverless v2 实例暂停时会发生什么情况

当 Aurora Serverless v2 数据库实例在一段时间没有连接后暂停时：

- 无论实例当时有多少 ACU，只要经过指定的间隔没有与实例的连接，Aurora 就开始暂停实例。
- 暂停机制不是瞬间的。即将自动暂停的 Aurora Serverless v2 实例可能会稍等片刻才能跟上对 Aurora 存储的所有更改。
- 针对该实例的收费暂停。实例暂停时，`ServerlessV2Usage` 指标的值为 0。
- 实例的状态值不变。状态仍显示为“可用”。
- 实例停止写入数据库日志文件。它会停止向 CloudWatch 发送指标，除了将 `CPUUtilization` 和 `ACUUtilization` 记录为零百分比，以及将 `ServerlessDatabaseCapacity` 记录为零。
- 当 Aurora Serverless v2 数据库实例开始暂停、完成暂停以及暂停机制中断或失败时，Aurora 会发出事件。有关这些事件的详细信息，请参阅数据库实例事件。

当自动暂停的 Aurora Serverless v2 实例恢复时会发生什么情况

当 Aurora Serverless v2 数据库实例在自动暂停后恢复时，会发生以下情况：

- 当实例恢复时，会应用 `pending-reboot` 更改中的任何参数更改。
- 当每个 Aurora Serverless v2 数据库实例开始恢复、完成恢复以及实例由于某种原因无法恢复时，Aurora 会发出实例级别的事件。有关这些事件的详细信息，请参阅数据库实例事件。
- 当数据库实例完成恢复后，将建立任何请求的连接。由于恢复的时间通常约为 15 秒，所以建议您将任何客户端超时设置调整为比 15 秒更长一些。例如，在 JDBC 驱动程序设置中，可以将 `connectTimeout` 和 `sslResponseTimeout` 设置的值调整为超过 15 秒。

> **Note**
>
> 如果 Aurora Serverless v2 实例暂停超过 24 小时，Aurora 可以让该实例进入更深的睡眠状态，这就需要更长的时间才能恢复。在这种情况下，恢复时间可以是 30 秒或更长，大致相当于重启实例的时间。如果您的数据库的非活动时间超过一天，建议将连接超时设置为 30 秒或更长时间。

自动暂停的 Aurora Serverless v2 集群的实例计费方法

当 Aurora Serverless v2 实例自动暂停时，其实例费用为零。在此期间，`ServerlessV2Usage` 指标为零。AWS 仍会对 Aurora 存储和集群中与该特定数据库实例无关的其他方面收费。

Aurora Serverless v2 不会自动暂停的情况

- 如果您的数据库集群的最小容量值高于零 ACU，则集群中的 Aurora Serverless v2 实例不会自动暂停。如果您的现有集群中包含自动暂停功能推出之前的 Aurora Serverless v2 实例，则最低的最小容量设置为 0.5 ACU。要对此类集群使用自动暂停功能，请将最小容量设置修改为零 ACU。
- 如果任何用户发起的与 Aurora Serverless v2 实例的连接处于打开状态，则该实例不会暂停。在修补和升级等活动进行期间，实例也不会暂停。Aurora 用于运行状况检查的管理连接不算作活动，因此不会阻止实例暂停。
- 在启用了逻辑复制的 Aurora PostgreSQL 集群中，或者在启用了二进制日志复制的 Aurora MySQL 集群中，写入器实例和失效转移提升层级零和一中的任何读取器实例都不会自动暂停。Aurora 会为了检查复制连接的运行状况而执行一个恒定的最低量的活动。

  对于启用了复制功能的集群，您仍然可以在源集群中拥有自动暂停的 Aurora 读取器实例。为此，请将它们的失效转移优先级设置为零或一以外的值。这样，它们就可以独立于写入器实例暂停。
- 如果您的 Aurora 集群具有关联的 RDS 代理，则该代理会保持与集群中每个数据库实例的打开连接。因此，此类集群中的任何 Aurora Serverless v2 实例都不会自动暂停。
- 如果您的集群是 Aurora Global Database 中的主集群，则 Aurora 不会自动暂停 Aurora Serverless v2 写入器实例。这是因为写入器实例上需要一个恒定水平的活动来管理全局数据库中的其他集群。由于写入器实例保持活动状态，所以任何失效转移优先级为零或一的 Aurora Serverless v2 读取器实例也不会自动暂停。
- Aurora Global Database 的辅助集群中的 Aurora Serverless v2 实例不会自动暂停。如果数据库集群升级为独立集群，并且所有其他条件都满足，则自动暂停功能变为有效。
- 在具有与 Redshift 关联的零 ETL 集成的集群中，写入器实例和失效转移提升层级零和一中的任何读取器实例都不会自动暂停。
- 除了实例的主数据库端口上的活动外，如果 Aurora PostgreSQL 实例还启用了 Babelfish 功能，则 T-SQL 端口上的任何连接和活动都将阻止该实例自动暂停。

自动暂停如何与集群停止/启动功能配合使用

启用自动暂停功能后，您可以停止和启动 Aurora 集群。某些实例是否已暂停都没关系。当您再次启动集群时，任何已暂停的 Aurora Serverless v2 实例都会自动恢复。

当 Aurora 集群停止时，任何已暂停的 Aurora Serverless v2 实例都不会根据连接尝试自动恢复。集群再次启动后，将采用暂停和恢复 Aurora Serverless v2 实例的常用机制。

自动暂停的 Aurora Serverless v2 集群的维护和升级工作原理

- 当一个 Aurora Serverless v2 实例自动暂停时，如果您尝试升级 Aurora 集群，Aurora 会恢复该实例并升级它。
- Aurora 会定期恢复所有自动暂停的 Aurora Serverless v2 实例，以执行维护，例如次要版本升级和对参数组等属性进行更改。
- 当一个 Aurora Serverless v2 实例被唤醒以执行管理操作（例如升级或应用维护）后，Aurora 会等待至少 20 分钟才会再次暂停该实例。这将允许任何后台操作完成。如果实例连续经历多个管理操作，这二十分钟的时间段还能避免多次暂停和恢复实例。

Aurora Serverless v2 和 Aurora Serverless v1 在自动暂停行为上的区别

- 与 Aurora Serverless v1 相比，Aurora Serverless v2 中的恢复时间有所缩短。如果实例的暂停时间少于 24 小时，则恢复时间通常约为 15 秒。如果实例的暂停时间超过 24 小时，则恢复时间可能会更长。
- Aurora Serverless v2 应用于多可用区集群的方式意味着集群中一些数据库实例可能暂停，而另一些则处于活动状态。只要有读取器在运行，写入器实例就会恢复，因为需要写入器来协调集群中的某些活动。因为 Aurora Serverless v1 不使用读取器实例，所以整个集群将始终处于暂停或活动状态。
- 当读取器端点随机选择一个读取器实例来连接时，该读取器实例可能已经处于活动状态或可能已自动暂停。因此，访问读取器实例的时间可能不同且很难预测。使用 Aurora Serverless v2 和自动暂停的多可用区集群因而可能受益于为特定的只读用例设置自定义端点，而不是将所有只读会话定向到读取器端点。
- Aurora Serverless v2 实例经历维护操作的频率与预调配实例相同。因为 Aurora 会在需要此类维护时自动恢复实例，所以您可能发现 Aurora Serverless v2 实例的恢复频率高于 Aurora Serverless v1 集群。

Aurora Serverless v2 自动暂停如何适用于不同类型的 Aurora 集群

自动暂停功能的注意事项取决于您的 Aurora 集群中有多少实例、读取器实例的失效转移提升层级以及实例都是 Aurora Serverless v2 实例还是 Aurora Serverless v2 实例与预调配实例的组合。

主题

- 使用自动暂停时推荐的 Aurora 集群布局
- Aurora Serverless v2 自动暂停如何用于数据库集群中的写入器实例
- Aurora Serverless v2 自动暂停如何用于多可用区集群

- Aurora Serverless v2 自动暂停如何用于包含预调配实例的集群

使用自动暂停时推荐的 Aurora 集群布局

启用自动暂停功能时，您可以安排 Aurora 集群布局，以便在高可用性、快速响应和可扩展性之间取得适当的平衡，来适应您的用例。为此，您可以为集群中的数据库实例选择 Aurora Serverless v2 实例、预调配实例和失效转移提升层级的组合。

以下类型的配置演示了集群的高可用性和成本优化之间的不同权衡：

- 对于开发和测试系统，您可以设置包含 Aurora Serverless v2 数据库实例的单可用区数据库集群。该单个实例可服务于所有读取和写入请求。当集群在很长一段时间内未使用时，数据库实例会暂停。此时，您的集群的数据库计算成本也会暂停。
- 对于运行优先考虑高可用性的应用程序、但集群仍有完全空闲时期的系统，您可以设置一个其中的写入器和读取器数据库实例都是 Aurora Serverless v2 的多可用区集群。将读取器实例设置为失效转移优先级为零或一，以便写入器和读取器实例同时暂停和恢复。现在，您可以在集群处于活动状态时获得快速失效转移的好处。当集群的闲置时间超过自动暂停阈值时，两个实例的数据库实例费用都将暂停。当集群恢复处理时，第一个数据库会话需要很短的时间即可连接。
- 假设您的集群一直处于活动状态，但活动量极少，并且任何连接都需要快速响应。在这种情况下，您可以创建一个包含多个 Aurora Serverless v2 读取器实例的集群，并将某些读取器实例的容量与写入器解耦。将写入器实例和某个读取器实例的失效转移优先级指定为零或一。为其他读取器实例指定大于一的优先级。这样，即使写入器和其中一个读取器保持活动状态，优先级较高的读取器实例也可以自动暂停。

  在这种情况下，您可以采用其他一些方法来确保集群保持持续可用，同时在空闲期间仍可缩减到低容量：

  - 您可以为写入器和优先级为 0 或 1 的读取器使用预调配实例。这样，两个数据库实例就永远不会自动暂停，从而始终可用于服务数据库流量和执行失效转移。
  - 您可以设置一个自定义端点，其中包含优先层级较高的 Aurora Serverless v2 实例，但不包含写入器或提升层级为 0 或 1 的读取器。这样，您就可以将对延迟不敏感的只读会话定向到可能被自动暂停的读取器。您可以避免将读取器端点用于此类请求，因为 Aurora 可能会将读取器端点连接定向到始终处于唤醒状态的读取器实例或自动暂停的实例之一。使用自定义端点，您可以根据自己对快速响应或额外扩展容量的偏好将连接定向到不同的实例组。

Aurora Serverless v2 自动暂停如何用于数据库集群中的写入器实例

当 Aurora 数据库集群仅包含一个数据库实例时，自动暂停和恢复数据库实例的机制非常简单。它仅取决于写入器实例上的活动。对于用于开发和测试的集群，或者为了运行高可用性不重要的应用程序，您可以采取此类配置。请注意，在单实例集群中，Aurora 会通过读取器端点将连接定向到写入器数据库实例。因此，对于单实例数据库集群来说，尝试连接到读取器端点会导致自动暂停的写入器数据库实例恢复。

以下额外因素适用于具有多个数据库实例的 Aurora 集群：

- 在 Aurora 数据库集群中，写入器数据库实例通常被频繁访问。因此，您可能会发现，即使一个或多个读取器数据库实例自动暂停，写入器数据库实例仍保持活动状态。
- 读取器数据库实例上的某些活动要求该写入器数据库实例可用。因此，写入器数据库实例要等到所有读取器实例都暂停后方能暂停。即使您的应用程序没有直接访问写入器实例，恢复任何读取器实例也会自动恢复写入器实例。
- 失效转移提升层级为零和一的 Aurora Serverless v2 读取器实例会进行扩展，以确保容量与写入器实例保持同步。因此，当 Aurora Serverless v2 写入器实例恢复时，提升层级为零或一的任何 Aurora Serverless v2 读取器实例也会恢复。

Aurora Serverless v2 自动暂停如何用于多可用区集群

在同时包含一个写入器和一个或多个读取器数据库实例的 Aurora 数据库集群中，某些 Aurora Serverless v2 数据库实例可能会暂停，而其他数据库实例则处于活动状态。写入器实例和任何失效转移优先级为 0 和 1 的读取器实例始终同时暂停和恢复。优先级不是 0 或 1 的读取器实例可以独立于其他实例暂停和恢复。

当您将此功能用于具有多个读取器实例的集群时，可以在不牺牲高可用性的情况下管理成本。写入器实例和另外一个或两个读取器实例可以始终保持活动状态，而其他读取器实例可以在不需要处理高容量读取流量时暂停。

读取器实例能否自动暂停取决于其容量能否独立扩展，或者其容量是否与写入器数据库实例的容量挂钩。该扩缩属性取决于读取器数据库实例的失效转移优先级。当读取器的优先级为零或一时，读取器的容量将跟踪写入器数据库实例的容量。因此，要允许读取器数据库实例在最广泛的情况下自动暂停，请将其优先级设置为大于 1 的值。

恢复读取器实例的时间可能比恢复写入器实例的时间稍长一些。如果实例可能暂停，要获得最快的响应，请连接到集群端点。

Aurora Serverless v2 自动暂停如何用于包含预调配实例的集群

Aurora 数据库集群中的任何预调配数据库实例都不会自动暂停。只有实例类为 `db.serverless` 的 Aurora Serverless v2 数据库实例才能使用自动暂停功能。

当您的 Aurora 集群包含任何预配置的数据库实例时，任何 Aurora Serverless v2 写入器实例都不会自动暂停。这是因为在任何读取器实例处于活动状态时，写入器实例必须保持可用。Aurora Serverless v2 写入器保持活动状态这一事实也意味着，在包含任何预调配实例的混合集群中，任何失效转移优先级为 0 和 1 的 Aurora Serverless v2 读取器实例都不会自动暂停。

监控使用自动暂停功能的 Aurora 集群

要监控 Aurora，您应该已经熟悉使用 Amazon CloudWatch 监控 Amazon Aurora 指标中的监控程序和 Amazon Aurora 的指标参考中列出的 CloudWatch 指标。请注意，监控使用自动暂停功能的 Aurora 集群时，有一些特殊注意事项：

- 因为实例暂停，会有一些时间段 Aurora Serverless v2 实例不记录日志数据和大多数指标。实例暂停期间发送到 CloudWatch 的指标只有 `CPUUtilization` 和 `ACUUtilization` 为零百分比以及 `ServerlessDatabaseCapacity` 为零。
- 您可以检查 Aurora Serverless v2 实例暂停的频率比您预计的高还是低。为此，请检查 `ServerlessDatabaseCapacity` 指标从非零值变为零的频率以及保持零的时间长度。如果实例的暂停时间没有您预计的那么长，表明您还没有节省尽可能多的成本。如果实例的暂停和恢复频率高于您的预期，表明您的集群在响应连接请求时可能存在不必要的延迟。有关影响 Aurora Serverless v2 实例能否暂停以及暂停频率的因素的信息，请参阅 Aurora Serverless v2 自动暂停功能的先决条件和限制、Aurora Serverless v2 不会自动暂停的情况和 Aurora Serverless v2 自动暂停故障排查。
- 您还可以检查记录 Aurora Serverless v2 实例自动暂停和恢复操作的日志文件。如果实例在超时间隔到期后没有暂停，则此日志文件还会包含未自动暂停的原因。有关更多信息，请参阅 ???。

主题

- 检查 Aurora Serverless v2 实例是否已暂停
- 自动暂停和自动恢复操作事件
- 自动暂停如何与性能详情和增强监控功能配合使用
- Aurora Serverless v2 自动暂停功能如何与 Aurora 指标交互

检查 Aurora Serverless v2 实例是否已暂停

要确定某个 Aurora Serverless v2 实例是否处于暂停状态，您可以观察该实例的 `ACUUtilization` 指标。如果实例暂停，则该指标的值为零。

当 Aurora Serverless v2 实例暂停时，其状态值仍列为可用。当暂停的 Aurora Serverless v2 实例正在恢复时也是如此。这是因为即使连接稍有延迟，您仍然可以成功连接到此类实例。

任何与 Aurora 实例可用性相关的指标都将实例暂停时期视为实例可用时期。

自动暂停和自动恢复操作事件

当自动暂停和自动恢复操作开始、完成或取消时，Aurora 会为 Aurora Serverless v2 实例发出事件。与自动暂停功能相关的事件为 `RDS-EVENT-0370` 至 `RDS-EVENT-0374`。有关这些事件的详细信息，请参阅数据库实例事件。

自动暂停如何与性能详情和增强监控功能配合使用

在 Aurora Serverless v2 实例暂停期间，Aurora 不会通过性能详情或增强监控功能收集该实例的监控信息。实例恢复后，可能会延迟一小会儿，然后 Aurora 才恢复收集此类监控信息。

Aurora Serverless v2 自动暂停功能如何与 Aurora 指标交互

当某个 Aurora Serverless v2 实例暂停时，它不会发出大多数 CloudWatch 指标，也不会向其数据库日志写入任何信息。实例暂停期间发送到 CloudWatch 的指标只有 `CPUUtilization` 和 `ACUUtilization` 为零百分比以及 `ServerlessDatabaseCapacity` 为零。

当 CloudWatch 计算与实例或集群可用性和正常运行时间相关的统计数据时，Aurora Serverless v2 实例被视为在暂停期间可用。

当您启动 AWS CLI 或 RDS API 操作来描述或下载已暂停 Aurora Serverless v2 实例的日志时，该实例会自动恢复以使日志信息可用。

CloudWatch 指标示例

以下 AWS CLI 示例显示了如何观察随时间推移而变化的实例容量。在这段时间内，实例会自动暂停，然后恢复。暂停时，`ServerlessDatabaseCapacity` 指标报告零值。为了确定该时间段内实例是否在任何时候暂停，我们会检查该时间段内的最小容量是否为零。

以下 Linux 示例表示一个已自动暂停一段时间的 Aurora Serverless v2 实例。我们在三分钟内每分钟对 `ServerlessDatabaseCapacity` 采样一次。最小 ACU 值为 0.0 表明实例在每分钟内的某个时刻处于暂停状态。

```
$ aws cloudwatch get-metric-statistics \
  --metric-name "ServerlessDatabaseCapacity" \
  --start-time "$(date -d '3 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --statistics Minimum \
  --namespace "AWS/RDS" --dimensions Name=DBInstanceIdentifier,Value=my-autopause-
instance \
  --output text | sort -k 3

ServerlessDatabaseCapacity
DATAPOINTS 0.0 2024-08-02T22:11:00+00:00 None
DATAPOINTS 0.0 2024-08-02T22:12:00+00:00 None
DATAPOINTS 0.0 2024-08-02T22:13:00+00:00 None
```

接下来，我们尝试与已暂停的 Aurora Serverless v2 实例建立连接。在此示例中，我们故意使用了错误的密码，以使连接尝试失败。尽管失败了，但连接尝试仍会导致 Aurora 恢复已暂停的实例。

```
$ mysql -h my_cluster_endpoint.rds.amazonaws.com -u admin -pwrong-password

ERROR 1045 (28000): Access denied for user 'admin'@'ip_address' (using password: YES)
```

以下 Linux 示例演示了暂停的实例被恢复，在闲置了大约五分钟后，又回到暂停状态。实例恢复时的容量为 2.0 ACU。然后它会稍微扩展，例如为了执行一些内部清理任务。由于该实例在五分钟的超时时间段内没有收到任何用户连接尝试，所以它从 4.0 ACU 直接进入暂停状态。

```
$ aws cloudwatch get-metric-statistics \
  --metric-name "ServerlessDatabaseCapacity" \
  --start-time "$(date -d '8 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --statistics Minimum \
  --namespace "AWS/RDS" --dimensions Name=DBInstanceIdentifier,Value=my-autopause-
instance \
  --output text | sort -k 3

ServerlessDatabaseCapacity
DATAPOINTS 0.0 2024-08-02T22:13:00+00:00 None
DATAPOINTS 2.0 2024-08-02T22:14:00+00:00 None
DATAPOINTS 3.0 2024-08-02T22:15:00+00:00 None
DATAPOINTS 3.0 2024-08-02T22:16:00+00:00 None
DATAPOINTS 4.0 2024-08-02T22:17:00+00:00 None
DATAPOINTS 4.0 2024-08-02T22:18:00+00:00 None
```

```
DATAPOINTS 4.0 2024-08-02T22:19:00+00:00 None
DATAPOINTS 0.0 2024-08-02T22:20:00+00:00 None
```

如果报告旨在显示实例如何快速扩展以处理工作负载，则我们可以为统计数据指定 `Maximum` 而不是 `Minimum`。对于容量规划和成本估算，我们可以指定更长的时间段并使用 `Average` 统计数据。这样，我们就可以确定高度活动、低度活动和暂停状态期间的典型容量值。为了检查暂停和恢复前后的精确时间段内的行为，我们可以指定一秒钟的时间段并检查更短的时间间隔。输出中的时间戳值（例如 `2024-08-02T22:13:00+00:00`）演示了为 `--start-time` 和 `--end-time` 选项指定精确参数的格式。

Aurora Serverless v2 自动暂停故障排查

如果您发现 Aurora Serverless v2 实例没有按您的预计频率暂停，请检查以下可能的原因：

- 确认您正在运行的 Aurora 版本确实支持零 ACU 的最小容量。有关不同 Aurora 版本的容量范围，请参阅 Aurora Serverless v2 容量。
- 确认集群的最小容量值已设置为零 ACU。
- 确认有问题的实例实际上使用 Aurora Serverless v2 实例类 `db.serverless`，而不是预调配的实例类之一。
- 确认集群未使用任何不兼容的功能或设置，详情参见 Aurora Serverless v2 自动暂停功能的先决条件和限制。
- 检查相关日志文件，其中会显示 Aurora Serverless v2 实例何时暂停、恢复或 Aurora 由于某种原因而无法暂停或恢复实例。有关更多信息，请参阅 ???。
- 检查是否有任何客户端或应用程序长时间保持连接打开。反过来，检查是否有使用 RDS 数据 API 或 Lambda 函数的应用程序频繁发送请求，导致实例一直闲不下来，无法暂停。您可以检查 CloudWatch 指标，例如 `ConnectionAttempts` 和 `DatabaseConnections`。有关更多信息，请参阅 Amazon Aurora 的实例级指标。
- 如果读取器实例极少暂停，请检查其失效转移优先级。如果读取器用于读取扩展，而不是在失效转移时用作备用读取器，请为其设置 2-15 范围的优先级。
- 如果写入器实例极少暂停，请检查您对读取器实例的使用情况。只有整个集群处于空闲状态时，写入器才能暂停。这是因为连接任何读取器实例时都会导致写入器恢复。
- 在 Aurora Serverless v2 实例正在恢复时，如果您的应用程序在连接请求期间收到超时，请考虑延长应用程序代码或底层数据库框架使用的超时间隔。在 Aurora Serverless v2 实例正在恢复时，较长的连接超时可降低连接失败的可能性。但是，较长的超时也会使您的应用程序检测集群中可用性问题的速度变慢。

反过来，可以考虑延长自动暂停间隔，这样应用程序就不会经常遇到暂停的实例。

如果应用程序的自动暂停行为和集群响应之间没有逻辑上的平衡，则该集群可能不适合使用自动暂停功能。

- 在估计 Aurora Serverless v2 实例将暂停多久时，请注意有些因素的存在会导致无法做出精确预测。
  - 实例可能会定期恢复，以执行维护、次要版本升级或应用参数组更改。
  - 对于多可用区集群，在某些情况下，恢复一个实例会导致其他实例也恢复。恢复任何读取器总是会导致恢复写入器。恢复写入器总是会导致恢复失效转移优先级为 0 和 1 的所有读取器。

  建议使用实际的工作负载来测定几天内的实际暂停时间。然后，使用这些测定值设置一个基准，来预计一个实例会暂停的时间比例。
- 您可能会发现，诸如 MySQL 清除、MySQL 事件调度器、PostgreSQL 自动清理或通过 `pg_cron` 扩展安排的 PostgreSQL 作业之类的内部操作没有运行或未完成。实例不会自动恢复以执行不涉及用户与数据库的连接的此类操作。如果在自动暂停超时到期时此类内部操作正在进行中，则会取消 MySQL 清除和 PostgreSQL 自动清理等维护任务。当 Aurora 启动暂停操作时，来自 MySQL 事件调度器或 PostgreSQL `pg_cron` 扩展的计划作业假如正在进行，则也会被取消。

  如果您需要确保实例定期被唤醒以执行计划操作，则可以在作业开始之前启动连接来恢复实例。您还可以延长自动暂停的超时间隔，以便像自动清理之类的操作可以在用户活动完成后运行更长的时间。您还可以使用诸如 Lambda 函数之类的机制来按计划执行数据库操作，且必要时可以自动恢复实例。

Aurora Serverless v2 自动暂停功能的应用程序设计注意事项

当 Aurora Serverless v2 数据库实例在自动暂停后恢复时，它先从相对较小的容量开始，然后在此基础上扩展。即使数据库实例在自动暂停之前不久还有很高的容量，也适用这一起始容量。

对于在建立连接时可以容忍大约 15 秒间隔的应用程序，请使用此功能。这反映了 Aurora Serverless v2 实例由于一个或少量传入连接而恢复的典型情况。如果实例的暂停时间超过 24 小时，则恢复的时间可能会更长。

如果您的应用程序已经在使用 Aurora Serverless v1 及其自动暂停功能，则您可能已经为连接尝试设置了这样的超时间隔。如果您已经在将 Aurora Serverless v2 与 Aurora 停止/启动集群功能结合使用，则自动暂停的 Aurora Serverless v2 实例的恢复时间通常比启动已停止的集群的时间短得多。

在应用程序中对连接逻辑进行编码时，如果第一次尝试返回的错误是暂时性的，请重试连接。（如果错误是由身份验证失败引起的，请在重试之前更正凭证。）恢复后紧接着发生的错误可能是超时，或者

是与数据库限制相关的错误。对于 Aurora Serverless v2 实例恢复是由同时存在大量连接请求引起的这种较罕见情形，重试可以解决问题。在这种情况下，某些连接的处理时间可能比平时长，或者可能在第一次尝试时超过同时连接的限制。

在应用程序开发和调试期间，不要留下与数据库的连接处于打开状态的客户端会话或编程工具。如果有任何用户启动的连接处于打开状态，Aurora 都不会暂停该实例，无论这些连接是否未运行任何 SQL 语句或事务。当 Aurora 集群中的一个 Aurora Serverless v2 实例无法暂停时，集群中的其他实例可能也无法暂停。有关更多信息，请参阅 Aurora Serverless v2 不会自动暂停的情况。

当 Aurora Serverless v2 数据库实例开始恢复、完成恢复以及实例由于某种原因无法恢复时，Aurora 会发出事件。有关这些事件的详细信息，请参阅数据库实例事件。

迁移到 Aurora Serverless v2

要将现有数据库集群转换为使用 Aurora Serverless v2，您可以执行以下操作：

- 从预置的 Aurora 数据库集群升级。
- 从 Aurora Serverless v1 集群升级。
- 从本地数据库迁移到 Aurora Serverless v2 集群。

当升级后的集群运行 Aurora Serverless v2 的要求和限制中列出的适当引擎版本时，您可以开始在其中添加 Aurora Serverless v2 数据库实例。您添加到升级后的集群的第一个数据库实例必须是预置数据库实例。然后，您可以将写入工作负载、读取工作负载或两者的处理切换到 Aurora Serverless v2 数据库实例。

目录

- 升级或切换现有集群以使用 Aurora Serverless v2
  - 升级 MySQL 兼容集群的路径以使用 Aurora Serverless v2
  - 升级 PostgreSQL 兼容集群以使用 Aurora Serverless v2 的路径
- 从预置集群切换到 Aurora Serverless v2
- 比较 Aurora Serverless v2 和 Aurora Serverless v1
  - Aurora Serverless v2 和 Aurora Serverless v1 要求的比较
  - Aurora Serverless v2 和 Aurora Serverless v1 扩缩和可用性的比较
  - Aurora Serverless v2 和 Aurora Serverless v1 功能支持的比较
  - 使 Aurora Serverless v1 使用案例适应 Aurora Serverless v2
- 从 Aurora Serverless v1 集群升级到 Aurora Serverless v2

- 与 Aurora MySQL 兼容的数据库集群
- 与 Aurora PostgreSQL 兼容的数据库集群
- 从本地数据库迁移到 Aurora Serverless v2

> **Note**
>
> 以下主题介绍如何转换现有数据库集群。有关创建新的 Aurora Serverless v2 数据库集群的信息，请参阅创建一个使用 Aurora Serverless v2 的数据库集群。

升级或切换现有集群以使用 Aurora Serverless v2

如果您的预置集群具有支持 Aurora Serverless v2 的引擎版本，则切换到 Aurora Serverless v2 不需要升级。在这种情况下，您可以将 Aurora Serverless v2 数据库实例添加到原始集群。您可以将集群切换为使用所有 Aurora Serverless v2 数据库实例。您还可以在同一个数据库集群中使用 Aurora Serverless v2 和预置数据库实例的组合。对于支持 Aurora Serverless v2 的 Aurora 引擎版本，请参阅支持 Aurora Serverless v2 的区域和 Aurora 数据库引擎。

如果您运行的是不支持 Aurora Serverless v2 的较低引擎版本，则需要采取以下一般步骤：

1. 升级集群。
2. 为升级后的集群创建预置写入器数据库实例。
3. 修改集群以使用 Aurora Serverless v2 数据库实例。

> **Important**
>
> 当您使用快照还原或克隆执行主要版本升级，升级到兼容 Aurora Serverless v2 的版本时，添加到新集群中的第一个数据库实例必须是预置数据库实例。这一添加开始了升级过程的最后阶段。
>
> 在最后阶段发生之前，集群不具备实现 Aurora Serverless v2 支持所需的基础设施。因此，这些升级后的集群始终从预置写入器数据库实例开始。然后，您可以将预置数据库实例转换为或失效转移到 Aurora Serverless v2 数据库实例。

从 Aurora Serverless v1 升级到 Aurora Serverless v2 的中间步骤包括创建预置集群。然后，执行的升级步骤与从预置集群开始时的升级步骤相同。

升级 MySQL 兼容集群的路径以使用 Aurora Serverless v2

如果您的原始集群正在运行 Aurora MySQL，请根据集群的引擎版本和引擎模式选择适当的程序。

如果您的原始 Aurora MySQL 集群是这样	执行此操作以切换到 Aurora Serverless v2
与 MySQL 8.0 兼容且运行 Aurora MySQL 版本 3 的预置集群	这是来自现有 Aurora MySQL 集群的所有转换的最后阶段。 如有必要，请执行次要版本升级，升级到 3.02.0 或更高版本。为写入器数据库实例使用预置数据库实例。添加一个 Aurora Serverless v2 读取器数据库实例。执行失效转移以使该实例变为写入器数据库实例。 （可选）将集群中的其他预调配数据库实例转换为 Aurora Serverless v2。或者添加新的 Aurora Serverless v2 数据库实例并移除预调配的数据库实例。 有关完整的程序和示例，请参阅从预置集群切换到 Aurora Serverless v2。
与 MySQL 5.7 兼容且运行 Aurora MySQL 版本 2 的预置集群	对 Aurora MySQL 版本 3.02.0 或更高版本执行主要版本升级。然后按照 Aurora MySQL 版本 3 的程序将集群切换为使用 Aurora Serverless v2 数据库实例。
与 MySQL 5.7 兼容且运行 Aurora MySQL 版本 2 的 Aurora Serverless v1 集群	为了帮助计划从 Aurora Serverless v1 转换，请先查阅比较 Aurora Serverless v2 和 Aurora Serverless v1。 然后，按照从 Aurora Serverless v1 集群升级到 Aurora Serverless v2中的程序进行操作。

升级 PostgreSQL 兼容集群以使用 Aurora Serverless v2 的路径

如果您的原始集群正在运行 Aurora PostgreSQL，请根据集群的引擎版本和引擎模式选择适当的程序。

如果您的原始 Aurora PostgreSQL 集群是这样	执行此操作以切换到 Aurora Serverless v2
运行 Aurora PostgreSQL 版本 13 的预置集群	这是来自现有 Aurora PostgreSQL 集群的所有转换的最后阶段。 如有必要，请执行次要版本升级，升级到 13.6 或更高版本。为写入器数据库实例添加一个预置数据库实例。添加一个 Aurora Serverless v2 读取器数据库实例。执行失效转移以使 Aurora Serverless v2 实例变为写入器数据库实例。 （可选）将集群中的其他预调配数据库实例转换为 Aurora Serverless v2。或者添加新的 Aurora Serverless v2 数据库实例并移除预调配的数据库实例。 有关完整的程序和示例，请参阅从预置集群切换到 Aurora Serverless v2。
运行 Aurora PostgreSQL 版本 11 或 12 的预置集群	对 Aurora PostgreSQL 版本 13.6 或更高版本执行主要版本升级。然后按照 Aurora PostgreSQL 版本 13 的程序将集群切换为使用 Aurora Serverless v2 数据库实例。
运行 Aurora PostgreSQL 版本 11 或 13 的 Aurora Serverless v1 集群	为了帮助计划从 Aurora Serverless v1 转换，请先查阅比较 Aurora Serverless v2 和 Aurora Serverless v1。 然后，按照从 Aurora Serverless v1 集群升级到 Aurora Serverless v2中的程序进行操作。

从预置集群切换到 Aurora Serverless v2

要将预置集群切换为使用 Aurora Serverless v2，请按照以下步骤操作：

1. 检查是否需要升级预置集群以与 Aurora Serverless v2 数据库实例一起使用。对于与 Aurora Serverless v2 兼容的 Aurora 版本，请参阅 Aurora Serverless v2 的要求和限制。

如果预置集群运行不适用于 Aurora Serverless v2 的引擎版本，则升级集群的引擎版本：

- 如果您有与 MySQL 5.7 兼容的预调配集群，请按照 Aurora MySQL 版本 3 的升级说明进行操作。按照如何执行就地升级中的程序进行操作。
- 如果您有运行 PostgreSQL 版本 11 或 12 且与 PostgreSQL 兼容的预调配集群，请按照 Aurora PostgreSQL 版本 13 的升级说明进行操作。按照执行主要版本升级中的程序进行操作。

2. 配置任何其他集群属性以匹配Aurora Serverless v2 的要求和限制中的 Aurora Serverless v2 要求。
3. 配置集群的扩缩配置。按照为集群设置 Aurora Serverless v2 容量范围中的程序进行操作。
4. 将一个或多个 Aurora Serverless v2 数据库实例添加到该集群。按照将 Aurora 副本添加到数据库集群中的常规程序进行操作。对于每个新的数据库实例，在 AWS Management Console 中指定特殊的数据库实例类名 Serverless（无服务器），或在 AWS CLI 或 Amazon RDS API 中指定 `db.serverless`。

   在某些情况下，您的集群中可能已经拥有一个或多个预置读取器数据库实例。如果已经拥有，您可以将其中一个读取器转换为 Aurora Serverless v2 数据库实例，而不是创建新的数据库实例。为此，请按照将预置写入器或读取器转换为 Aurora Serverless v2 中的程序操作。
5. 执行失效转移操作以使其中一个 Aurora Serverless v2 数据库实例成为集群的写入器数据库实例。
6. （可选）将任何预置数据库实例转换为 Aurora Serverless v2，或者将其从集群中删除。按照将预置写入器或读取器转换为 Aurora Serverless v2 和从 Aurora 数据库集群中删除数据库实例中的常规程序进行操作。

> **Tip**
>
> 删除预置数据库实例并非强制操作。您可以设置一个包含 Aurora Serverless v2 数据库实例和预置数据库实例的集群。但是，在您熟悉的 Aurora Serverless v2 数据库实例性能和扩缩特性之前，建议您使用相同类型的数据库实例配置集群。

以下 AWS CLI 示例显示了使用运行 Aurora MySQL 版本 3.02.0 的预置集群的切换过程。该集群名为 `mysql-80`。集群从两个名为 `provisioned-instance-1` 和 `provisioned-instance-2`（一个写入器和一个读取器）的预置数据库实例开始。它们都使用 `db.r6g.large` 数据库实例类。

```
$ aws rds describe-db-clusters --db-cluster-identifier mysql-80 \
  --query '*[].[DBClusterIdentifier,DBClusterMembers[*].
[DBInstanceIdentifier,IsClusterWriter]]' --output text
mysql-80
provisioned-instance-2     False
```

```
provisioned-instance-1     True

$ aws rds describe-db-instances --db-instance-identifier provisioned-instance-1 \
  --output text --query '*[].[DBInstanceIdentifier,DBInstanceClass]'
provisioned-instance-1     db.r6g.large

$ aws rds describe-db-instances --db-instance-identifier provisioned-instance-2 \
  --output text --query '*[].[DBInstanceIdentifier,DBInstanceClass]'
provisioned-instance-2     db.r6g.large
```

我们创建一个包含一些数据的表。这样，我们就可以确认集群的数据和运营在切换之前和之后是相同的。

```
mysql> create database serverless_v2_demo;
mysql> create table serverless_v2_demo.demo (s varchar(128));
mysql> insert into serverless_v2_demo.demo values ('This cluster started with a
 provisioned writer.');
Query OK, 1 row affected (0.02 sec)
```

首先，我们向集群添加容量范围。否则，在向集群添加任何 Aurora Serverless v2 数据库实例时都会出错。如果我们为此程序使用 AWS Management Console，那么当我们添加第一个 Aurora Serverless v2 数据库实例时，该步骤是自动的。

```
$ aws rds create-db-instance --db-instance-identifier serverless-v2-instance-1 \
  --db-cluster-identifier mysql-80 --db-instance-class db.serverless --engine aurora-
mysql

An error occurred (InvalidDBClusterStateFault) when calling the CreateDBInstance
 operation:
Set the Serverless v2 scaling configuration on the parent DB cluster before creating a
 Serverless v2 DB instance.

$ # The blank ServerlessV2ScalingConfiguration attribute confirms that the cluster
 doesn't have a capacity range set yet.
$ aws rds describe-db-clusters --db-cluster-identifier mysql-80 --query
 'DBClusters[*].ServerlessV2ScalingConfiguration'
[]

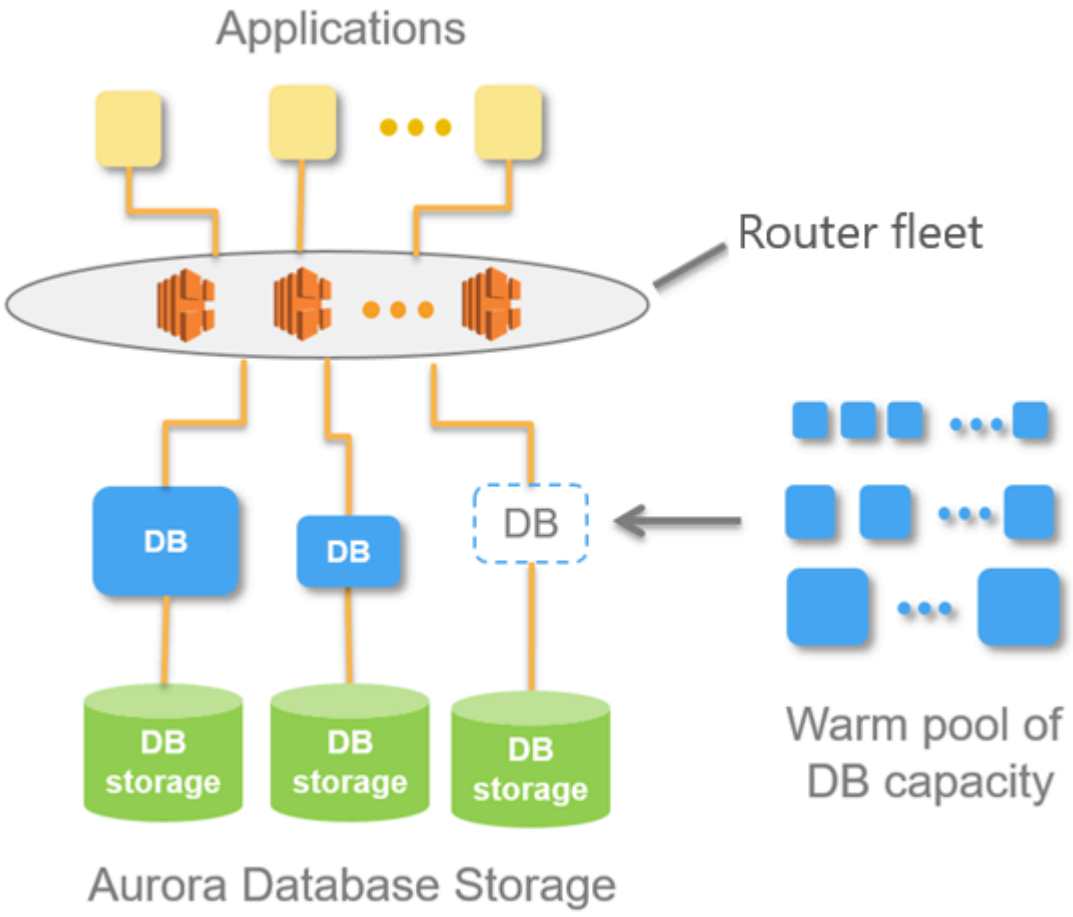
$ aws rds modify-db-cluster --db-cluster-identifier mysql-80 \
  --serverless-v2-scaling-configuration MinCapacity=0.5,MaxCapacity=16
{
  "DBClusterIdentifier": "mysql-80",
```

```
  "ServerlessV2ScalingConfiguration": {
    "MinCapacity": 0.5,
    "MaxCapacity": 16
  }
}
```

我们创建了两个 Aurora Serverless v2 读取器来取代原始数据库实例。我们通过为新的数据库实例指定 `db.serverless` 数据库实例类来实现。

```
$ aws rds create-db-instance --db-instance-identifier serverless-v2-instance-1 --db-
cluster-identifier mysql-80 --db-instance-class db.serverless --engine aurora-mysql
{
  "DBInstanceIdentifier": "serverless-v2-instance-1",
  "DBClusterIdentifier": "mysql-80",
  "DBInstanceClass": "db.serverless",
  "DBInstanceStatus": "creating"
}

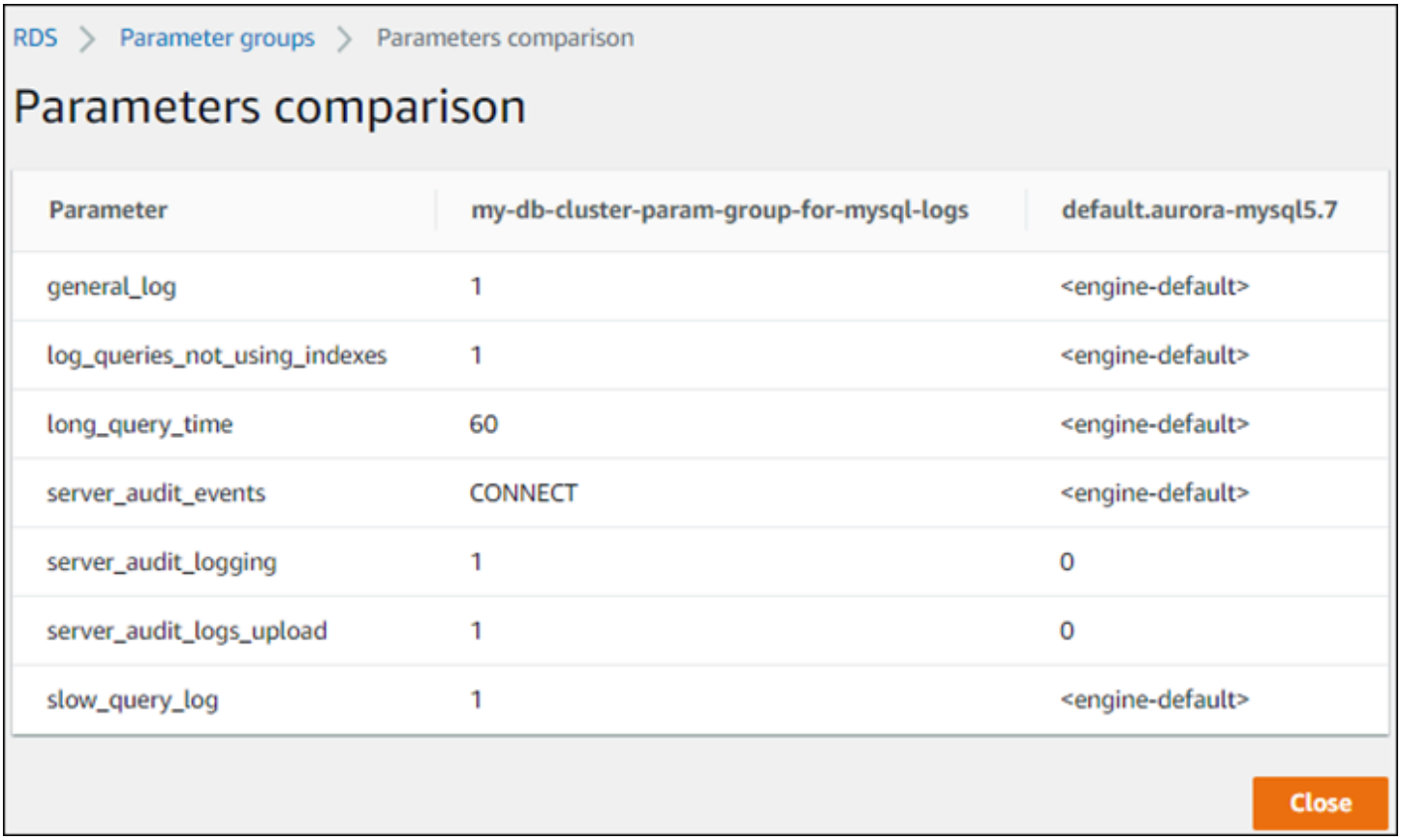
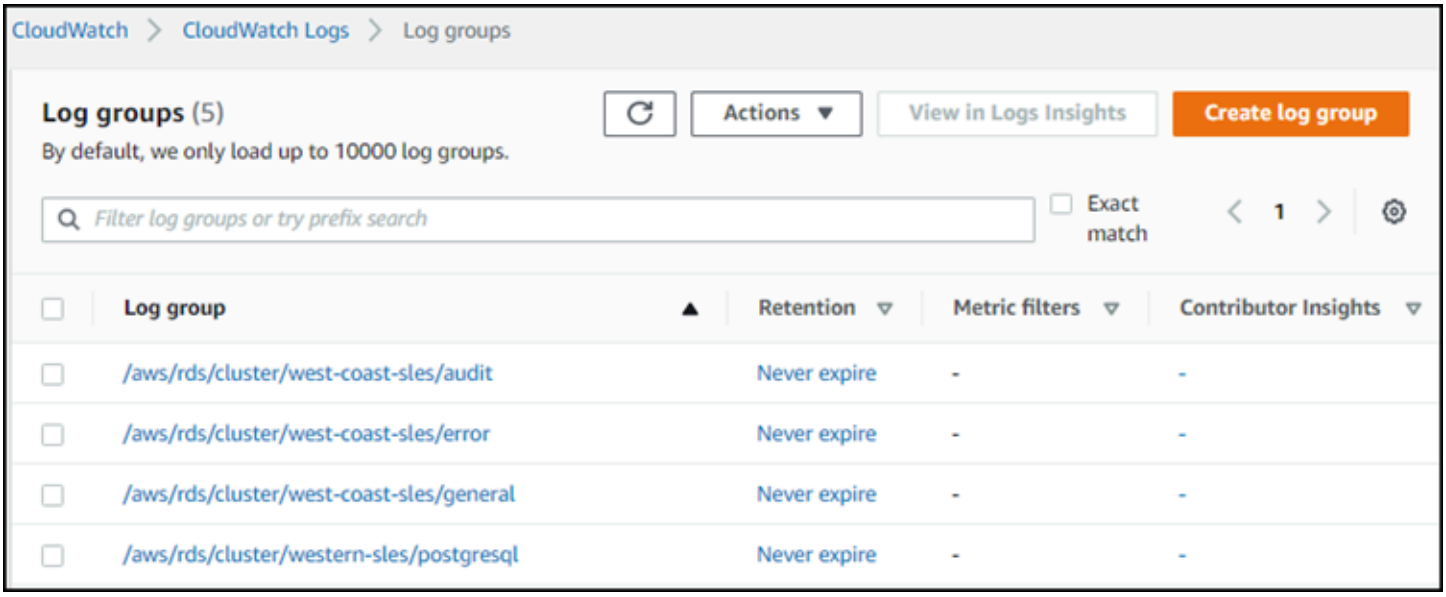
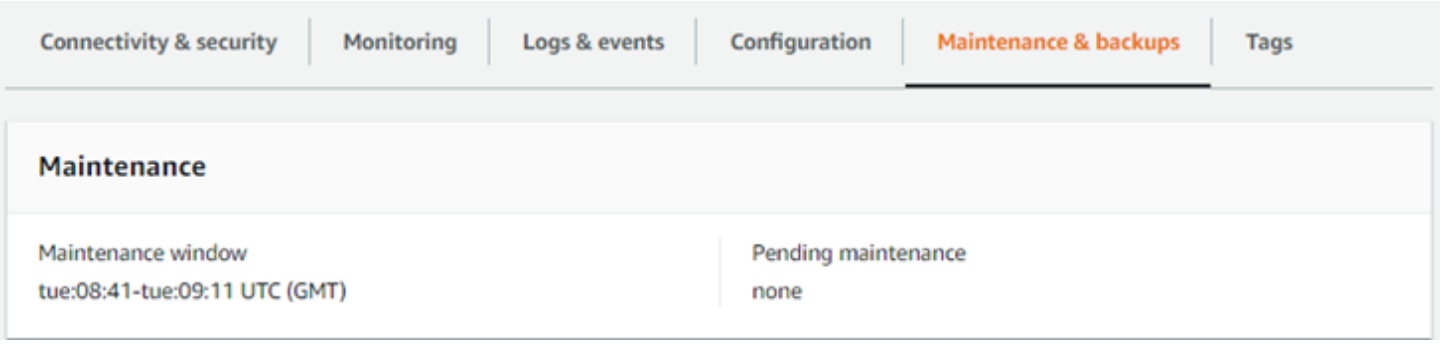
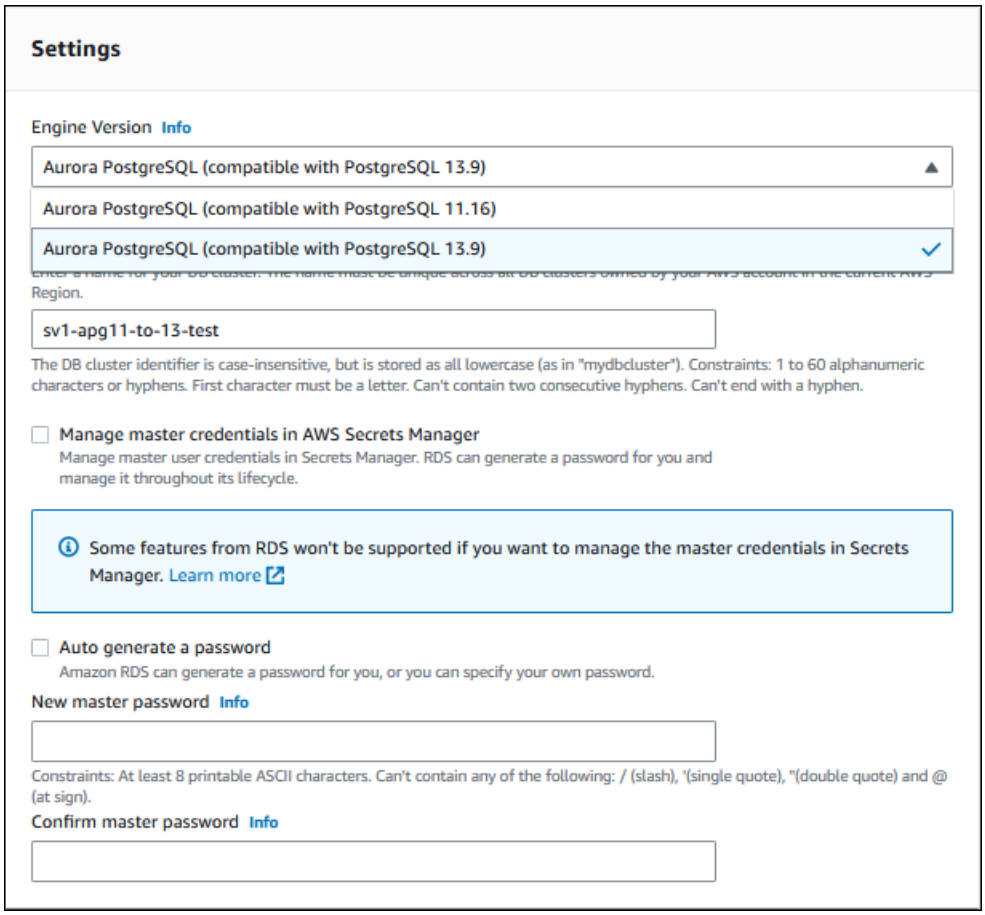
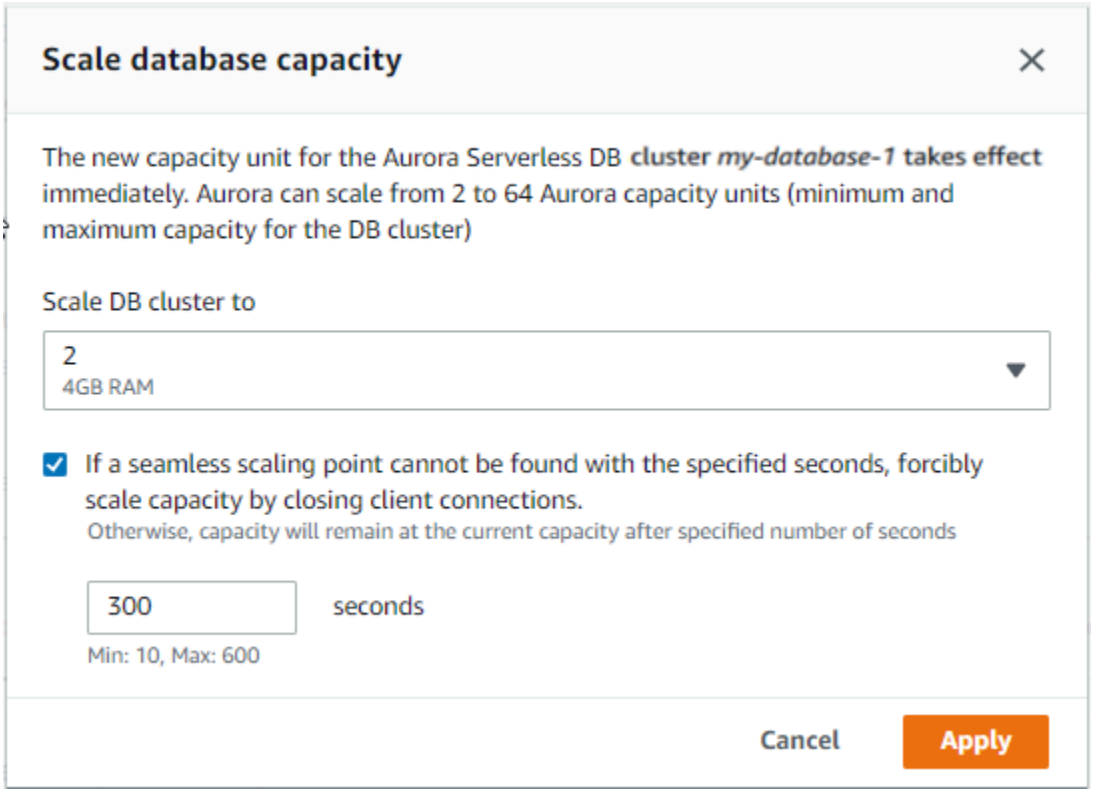
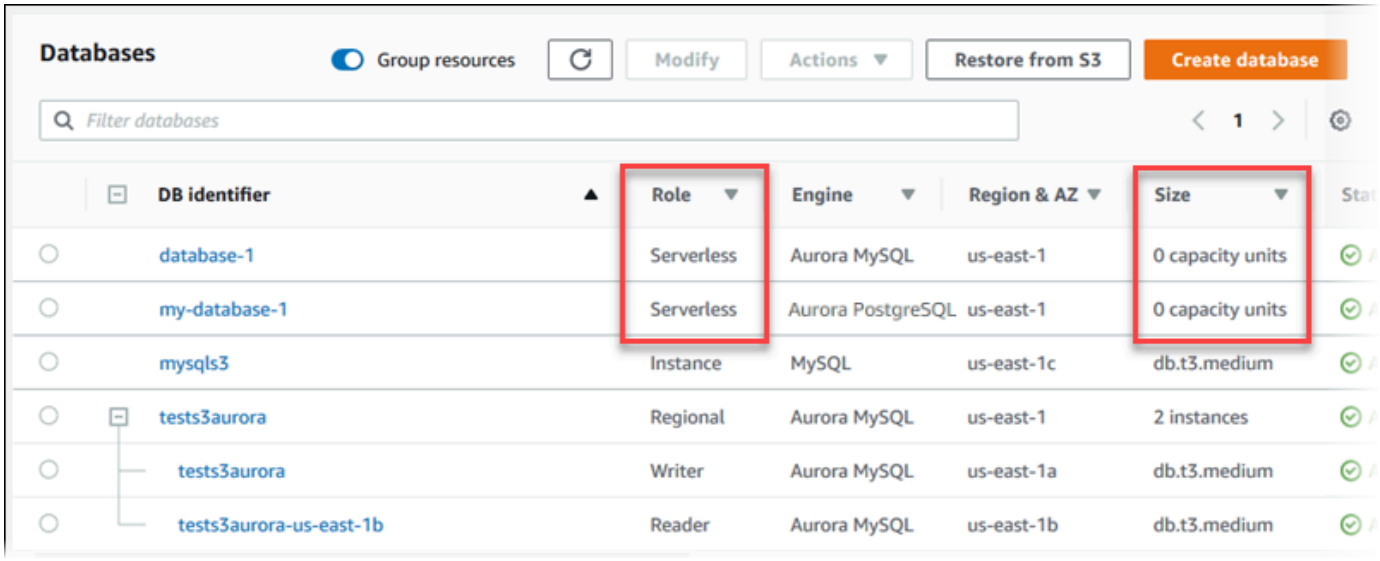
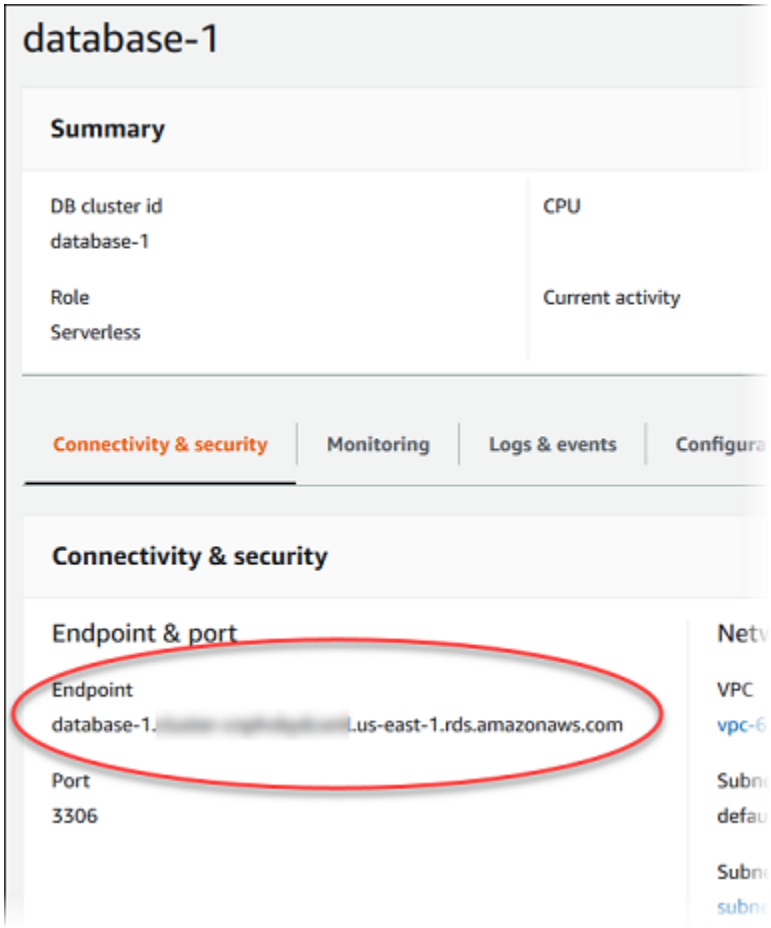
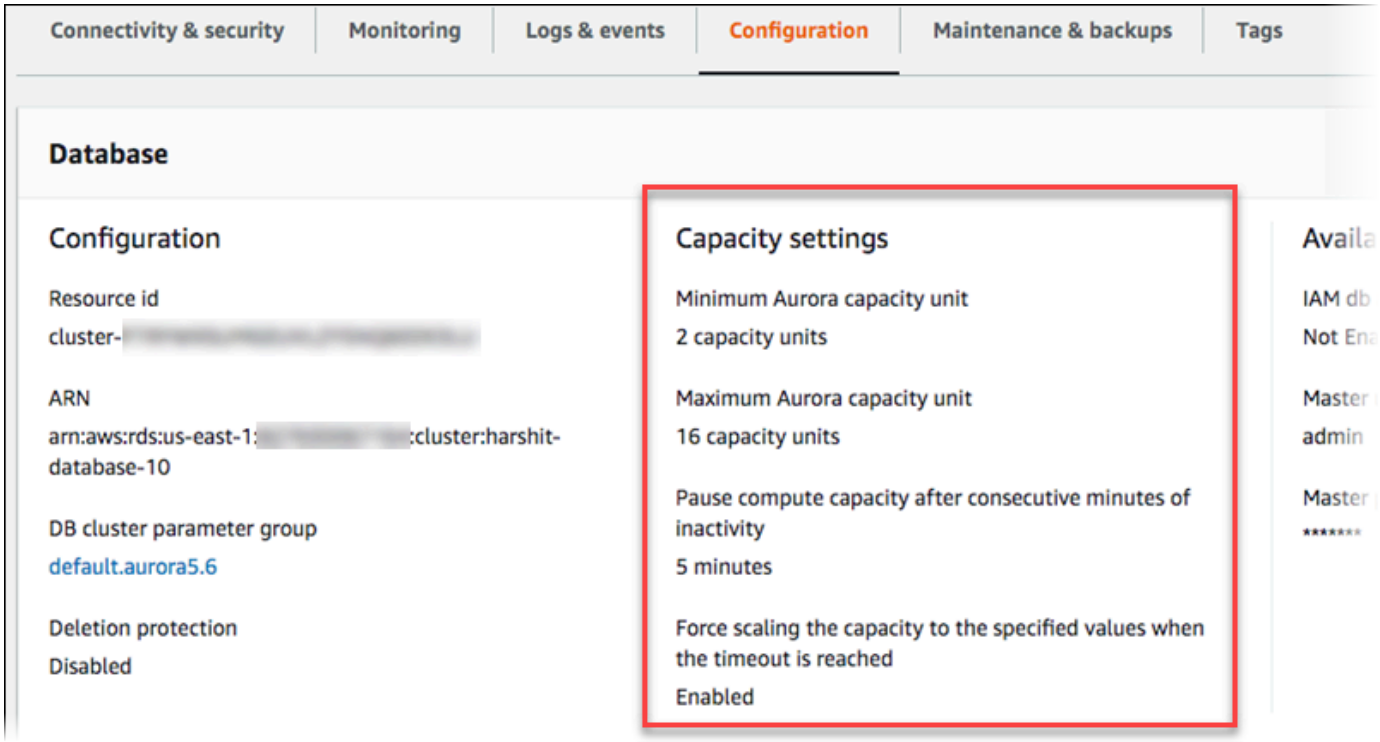
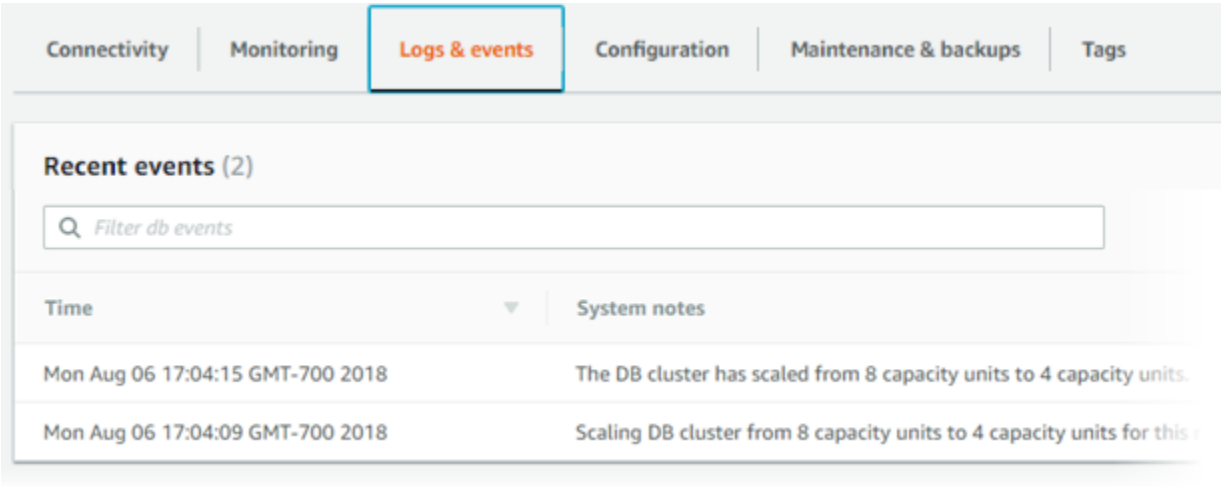
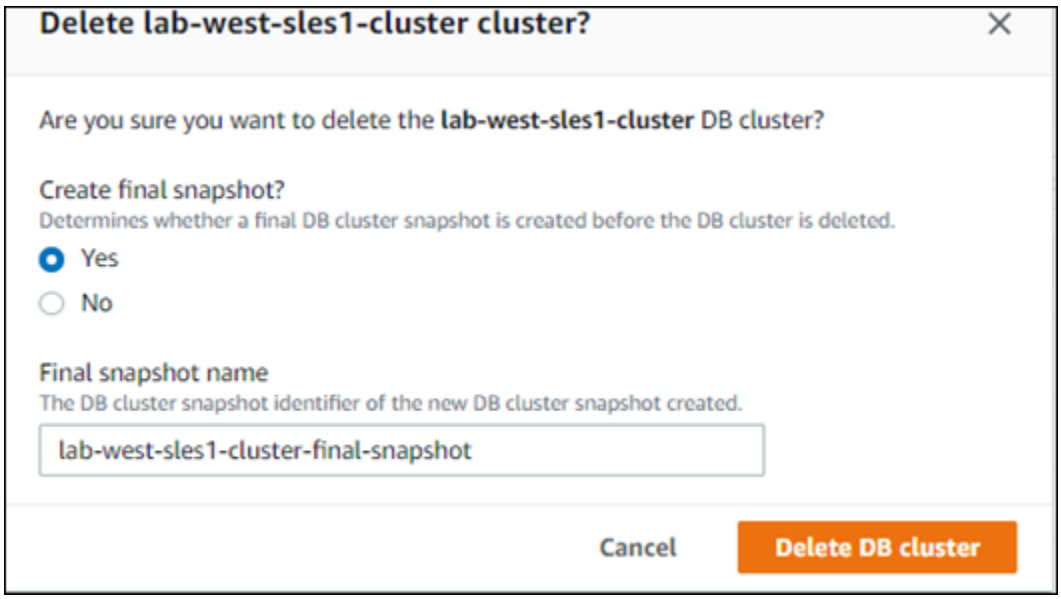
$ aws rds create-db-instance --db-instance-identifier serverless-v2-instance-2 \
  --db-cluster-identifier mysql-80 --db-instance-class db.serverless --engine aurora-
mysql
{
  "DBInstanceIdentifier": "serverless-v2-instance-2",
  "DBClusterIdentifier": "mysql-80",
  "DBInstanceClass": "db.serverless",
  "DBInstanceStatus": "creating"
}

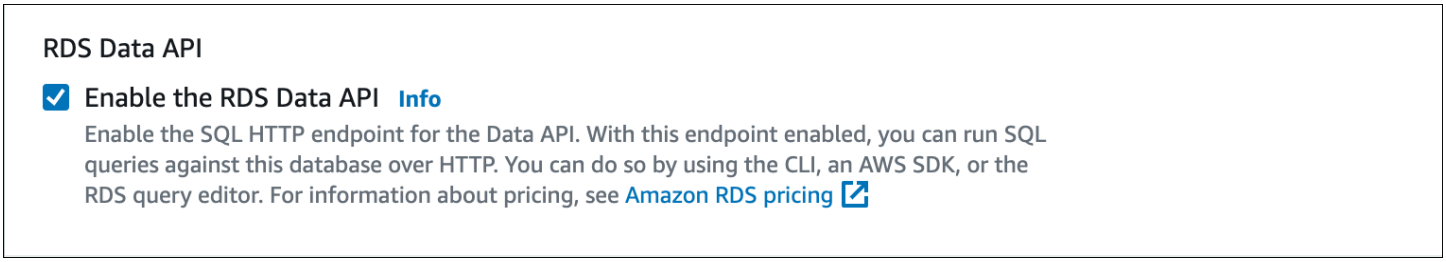
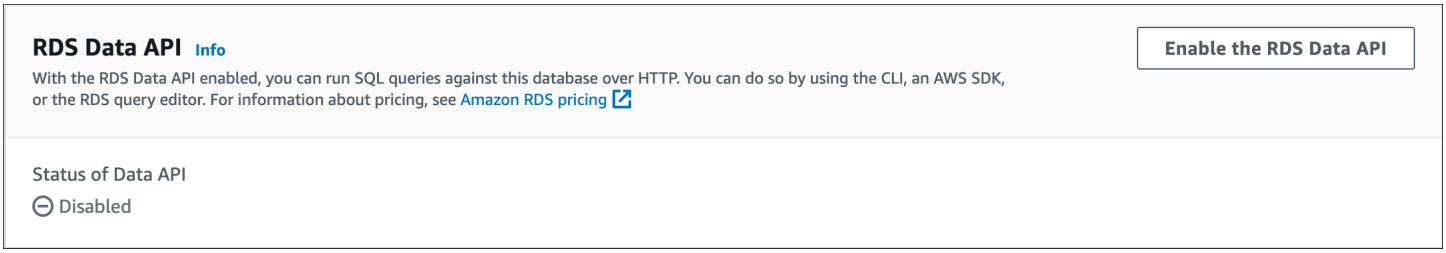
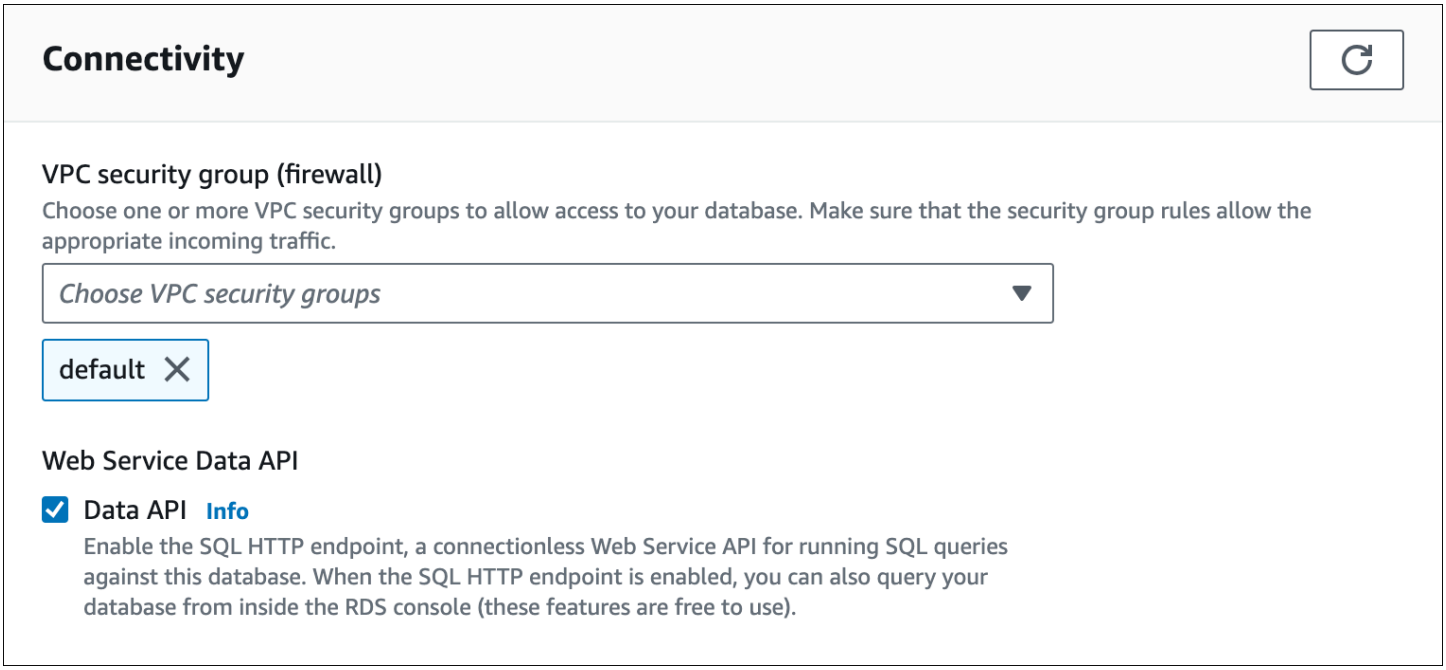
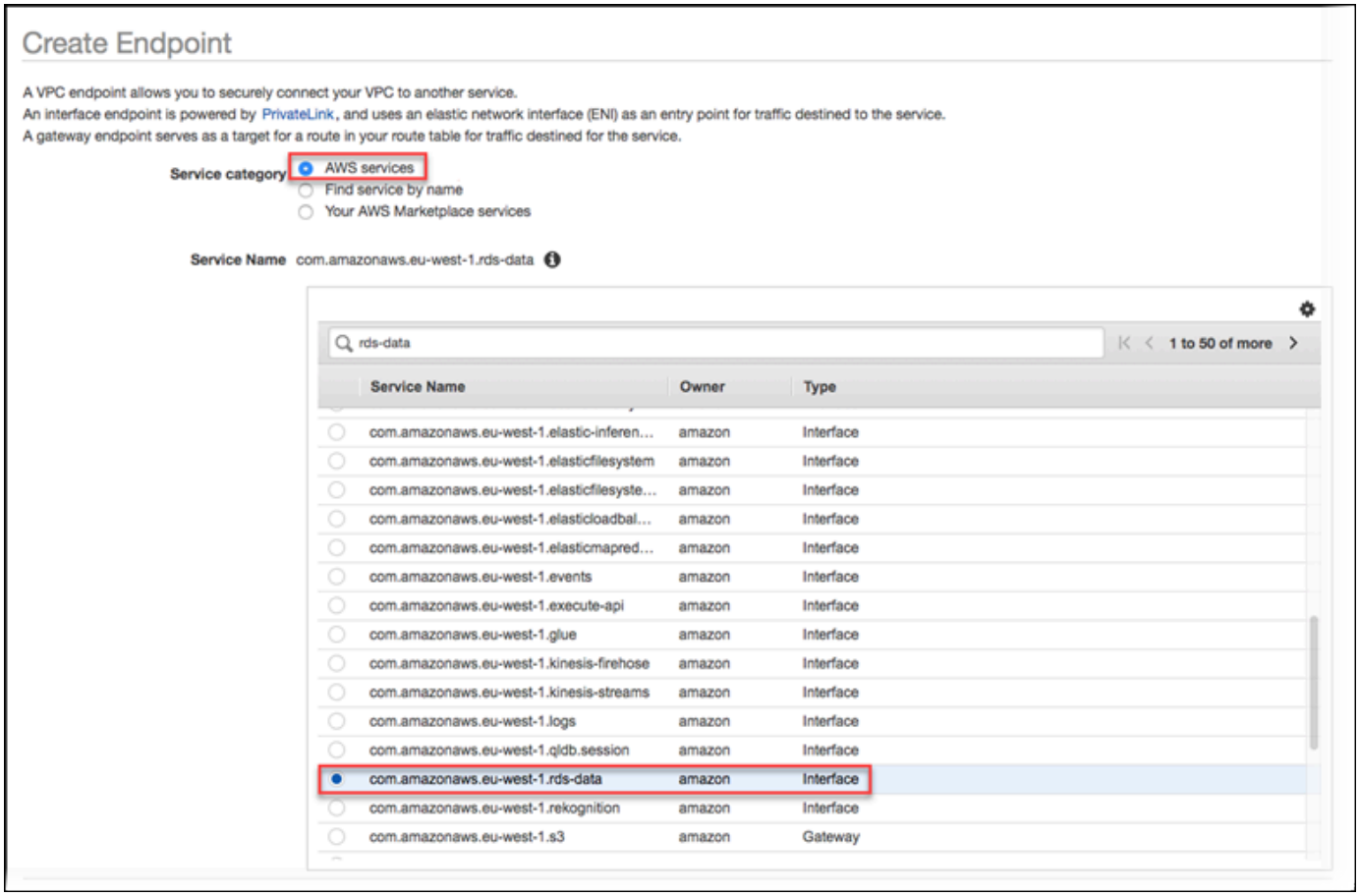
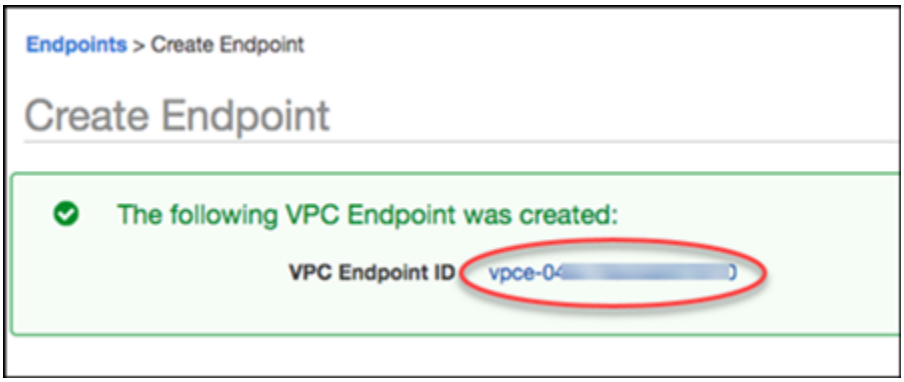
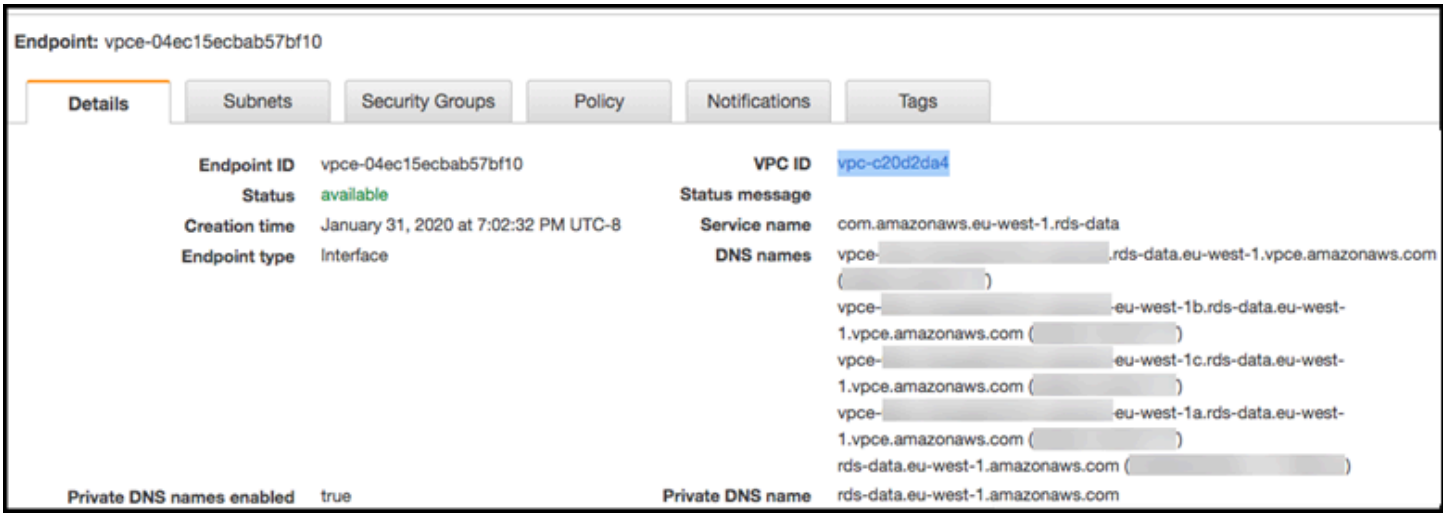
$ # Wait for both DB instances to finish being created before proceeding.
$ aws rds wait db-instance-available --db-instance-identifier serverless-v2-instance-1
 && \
  aws rds wait db-instance-available --db-instance-identifier serverless-v2-instance-2
```

我们执行失效转移以使其中一个 Aurora Serverless v2 数据库实例成为集群的新写入器。

```
$ aws rds failover-db-cluster --db-cluster-identifier mysql-80 \
  --target-db-instance-identifier serverless-v2-instance-1
{
  "DBClusterIdentifier": "mysql-80",
  "DBClusterMembers": [
    {
      "DBInstanceIdentifier": "serverless-v2-instance-1",
      "IsClusterWriter": false,
```

```
      "DBClusterParameterGroupStatus": "in-sync",
      "PromotionTier": 1
    },
    {
      "DBInstanceIdentifier": "serverless-v2-instance-2",
      "IsClusterWriter": false,
      "DBClusterParameterGroupStatus": "in-sync",
      "PromotionTier": 1
    },
    {
      "DBInstanceIdentifier": "provisioned-instance-2",
      "IsClusterWriter": false,
      "DBClusterParameterGroupStatus": "in-sync",
      "PromotionTier": 1
    },
    {
      "DBInstanceIdentifier": "provisioned-instance-1",
      "IsClusterWriter": true,
      "DBClusterParameterGroupStatus": "in-sync",
      "PromotionTier": 1
    }
  ],
  "Status": "available"
}
```

此更改需要几秒钟才能生效。这是，我们有了一个 Aurora Serverless v2 写入器和一个 Aurora Serverless v2 读取器。因此，我们不需要任何一个原始预置数据库实例。

```
$ aws rds describe-db-clusters --db-cluster-identifier mysql-80 \
  --query '*[].[DBClusterIdentifier,DBClusterMembers[*].
[DBInstanceIdentifier,IsClusterWriter]]' \
  --output text
mysql-80
serverless-v2-instance-1        True
serverless-v2-instance-2        False
provisioned-instance-2          False
provisioned-instance-1          False
```

切换程序的最后一步是删除两个预置数据库实例。

```
$ aws rds delete-db-instance --db-instance-identifier provisioned-instance-2 --skip-
final-snapshot
{
```

```
  "DBInstanceIdentifier": "provisioned-instance-2",
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql",
  "EngineVersion": "8.0.mysql_aurora.3.02.0",
  "DBInstanceClass": "db.r6g.large"
}

$ aws rds delete-db-instance --db-instance-identifier provisioned-instance-1 --skip-
final-snapshot
{
  "DBInstanceIdentifier": "provisioned-instance-1",
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql",
  "EngineVersion": "8.0.mysql_aurora.3.02.0",
  "DBInstanceClass": "db.r6g.large"
}
```

在最终检查时，我们确认原始表可以从 Aurora Serverless v2 写入器数据库实例访问和写入。

```
mysql> select * from serverless_v2_demo.demo;
+--------------------------------------------------+
| s                                                |
+--------------------------------------------------+
| This cluster started with a provisioned writer.  |
+--------------------------------------------------+
1 row in set (0.00 sec)

mysql> insert into serverless_v2_demo.demo values ('And it finished with a Serverless
 v2 writer.');
Query OK, 1 row affected (0.01 sec)

mysql> select * from serverless_v2_demo.demo;
+--------------------------------------------------+
| s                                                |
+--------------------------------------------------+
| This cluster started with a provisioned writer.  |
| And it finished with a Serverless v2 writer.     |
+--------------------------------------------------+
2 rows in set (0.01 sec)
```

我们还连接到 Aurora Serverless v2 读取器数据库实例并确认新写入的数据也在那里提供。

```
mysql> select * from serverless_v2_demo.demo;
```

```
+--------------------------------------------------+
| s                                                |
+--------------------------------------------------+
| This cluster started with a provisioned writer.  |
| And it finished with a Serverless v2 writer.     |
+--------------------------------------------------+
2 rows in set (0.01 sec)
```

比较 Aurora Serverless v2 和 Aurora Serverless v1

如果您已在使用 Aurora Serverless v1，您可以了解 Aurora Serverless v1 和 Aurora Serverless v2 之间的主要区别。架构差异（例如对读取器数据库实例的支持）开辟了新的使用场景类型。

您可以使用下表来帮助了解 Aurora Serverless v2 和 Aurora Serverless v1 之间的最重要区别。

主题

- Aurora Serverless v2 和 Aurora Serverless v1 要求的比较
- Aurora Serverless v2 和 Aurora Serverless v1 扩缩和可用性的比较
- Aurora Serverless v2 和 Aurora Serverless v1 功能支持的比较
- 使 Aurora Serverless v1 使用案例适应 Aurora Serverless v2

Aurora Serverless v2 和 Aurora Serverless v1 要求的比较

下表总结了使用 Aurora Serverless v2 或 Aurora Serverless v1 运行数据库的不同要求。与 Aurora Serverless v1 相比，Aurora Serverless v2 提供更高版本的 Aurora MySQL 和 Aurora PostgreSQL 数据库引擎。

功能	Aurora Serverless v2 要求	Aurora Serverless v1 要求
数据库引擎	Aurora MySQL 和 Aurora PostgreSQL	Aurora MySQL 和 Aurora PostgreSQL
支持的 Aurora MySQL 版本	请参阅 适用于 Aurora MySQL 的 Aurora Serverless v2。	请参阅 适用于 Aurora MySQL 的 Aurora Serverless v1。
支持的 Aurora PostgreSQL 版本	请参阅 适用于 Aurora PostgreSQL 的 Aurora Serverless v2。	请参阅 适用于 Aurora PostgreSQL 的 Aurora Serverless v1。

功能	Aurora Serverless v2 要求	Aurora Serverless v1 要求
升级数据库集群	与预调配的数据库集群类似，您可以手动执行升级，无需等待 Aurora 为您升级数据库集群。有关更多信息，请参阅 修改 Amazon Aurora 数据库集群。 **Note** 要对与 Aurora PostgreSQL 兼容的数据库集群执行从 13.x 到 14.x 或 15.x 的主要版本升级，集群的最大容量必须至少为 2 ACU。	次要版本升级将在可用时自动应用。有关更多信息，请参阅 Aurora Serverless v1 和 Aurora 数据库引擎版本。 您可以手动执行主要版本升级。有关更多信息，请参阅修改 Aurora Serverless v1 数据库集群。

功能	Aurora Serverless v2 要求	Aurora Serverless v1 要求
从预调配数据库集群进行转换	您可以使用以下方法： • 将一个或多个 Aurora Serverless v2 读取器数据库实例添加到现有预置集群。要为写入器使用 Aurora Serverless v2，请指定到其中一个 Aurora Serverless v2 数据库实例的失效转移。为了让整个集群使用 Aurora Serverless v2 数据库实例，在将 Aurora Serverless v2 数据库实例提升为写入器之后删除任何预置写入器数据库实例。 • 使用适当的数据库引擎和引擎版本创建新集群。使用任何标准方法。例如，还原集群快照或创建现有集群的克隆。为新集群中的部分或全部数据库实例选择 Aurora Serverless v2。 如果通过克隆创建新集群，则无法同时升级引擎版本。确保原始集群已经运行与 Aurora Serverless v2 兼容的引擎版本。	还原预置集群的快照以创建新的 Aurora Serverless v1 集群。
从 Aurora Serverless v1 集群进行转换	按照从 Aurora Serverless v1 集群升级到 Aurora Serverless v2中的程序操作。	不适用

功能	Aurora Serverless v2 要求	Aurora Serverless v1 要求
可用的数据库实例类	特殊数据库实例类 `db.serverless` 。在 AWS Management Console 中，它的名称为 Serverless（无服务器）。	不适用。Aurora Serverless v1 使用 `serverless` 引擎模式。
端口	与 MySQL 或 PostgreSQL 兼容的任何端口	仅限默认 MySQL 或 PostgreSQL 端口
是否允许公有 IP 地址？	是	否
是否需要 Virtual Private Cloud（VPC）？	是	是。每个 Aurora Serverless v1 集群占用分配给您的 VPC 的 2 个接口和网关负载均衡器端点。

Aurora Serverless v2 和 Aurora Serverless v1 扩缩和可用性的比较

下表汇总了 Aurora Serverless v2 和 Aurora Serverless v1 在可扩展性和可用性方面的区别。

与 Aurora Serverless v1 中的扩缩相比，Aurora Serverless v2 扩缩响应速度更快、更精细、中断更少。Aurora Serverless v2 可以通过更改数据库实例的大小和通过向数据库集群添加更多数据库实例来扩缩。也可以通过向 Aurora Global Database 添加其他 AWS 区域中的集群来扩缩。相比之下，Aurora Serverless v1 只能通过增大或减小写入器的容量来扩缩。Aurora Serverless v1 集群的所有计算在单个可用区和单个 AWS 区域中运行。

扩展和高可用性功能	Aurora Serverless v2 行为	Aurora Serverless v1 行为
最小 Aurora 容量单位（ACU）（Aurora MySQL）	集群运行时为 0.5，集群暂停时为 0。	集群运行时为 1，集群暂停时为 0。
最小 ACU（Aurora PostgreSQL）	集群运行时为 0.5，集群暂停时为 0。	集群运行时为 2，集群暂停时为 0。
最大 ACU（Aurora MySQL）	256	256

扩展和高可用性功能	Aurora Serverless v2 行为	Aurora Serverless v1 行为
最大 ACU（Aurora PostgreSQL）	256	384
停止集群	您可以通过使用与预置集群相同的集群停止和启动功能手动停止和启动集群。	超时后，集群会自动暂停。活动恢复后需要一段时间才能变得可用。
数据库实例的扩缩	以 0.5 ACU 的最小增量纵向扩展和缩减。	通过使 ACU 翻倍或减半来纵向扩展和缩减。
数据库实例数	与预置集群相同：1 个写入器数据库实例，最多 15 个读取器数据库实例。	1 个同时处理读取和写入的数据库实例。
当 SQL 语句正在运行时是否可以进行扩缩？	可以。Aurora Serverless v2 不需要等待安静点。	不可以。例如，扩缩会等待长时间运行的事务、临时表和表锁定完成。
读取器数据库实例与写入器一起扩缩	可选	不适用
最大存储空间	128 TiB	128 TiB
扩缩时保留缓冲区缓存	是。缓冲区缓存会动态调整大小。	否。缓冲区缓存在扩缩后重新调整。
失效转移	是，与预置集群相同。	仅尽最大努力，取决于容量供应情况。比在 Aurora Serverless v2 中更慢。
多可用区功能	是，与预置集群相同。多可用区集群需要在第二个可用区（AZ）中有读取器数据库实例。对于多可用区集群，Aurora 在发生可用区故障时执行多可用区失效转移。	Aurora Serverless v1 集群在一个可用区中运行所有计算。在可用区出现故障时的恢复为仅尽最大努力，并取决于容量供应情况。

扩展和高可用性功能	Aurora Serverless v2 行为	Aurora Serverless v1 行为
Aurora 全局数据库	是	否
根据内存压力进行扩缩	是	否
根据 CPU 负载进行扩缩	支持	是
根据网络流量进行扩缩	是，基于网络流量的内存和 CPU 开销。`max_connections` 参数保持不变，以避免在缩减时断开连接。	是，基于连接数。
扩缩事件的超时操作	否	是
通过 AWS Auto Scaling 向集群添加新的数据库实例	不适用。您可以在提升层 2-15 中创建 Aurora Serverless v2 读取器数据库实例，并将其缩减至低容量。	否。读取器数据库实例不可用。

Aurora Serverless v2 和 Aurora Serverless v1 功能支持的比较

下表对支持进行了汇总：

- 在 Aurora Serverless v2 中提供但在 Aurora Serverless v1 中不提供的功能
- 在 Aurora Serverless v1 和 Aurora Serverless v2 中以不同方式运行的功能
- 目前在 Aurora Serverless v2 中不可用的功能

Aurora Serverless v2 包括来自预置集群但不可用于 Aurora Serverless v1 的功能。

功能	Aurora Serverless v2 支持	Aurora Serverless v1 支持
集群拓扑	Aurora Serverless v2 是单个数据库实例的属性。一个集群可以包含多个 Aurora Serverless v2 数据库实例，或者包含	Aurora Serverless v1 集群不使用数据库实例的概念。创建集群之后不能更改 Aurora Serverless v1 属性。

功能	Aurora Serverless v2 支持	Aurora Serverless v1 支持
	Aurora Serverless v2 和预置数据库实例的组合。	
配置参数	几乎所有相同参数都可以像在预置集群中一样进行修改。有关详细信息，请参阅使用 Aurora Serverless v2 的参数组。	只可以修改参数的一个子集。
参数组	集群参数组和数据库参数组。提供在 `SupportedEngineModes` 属性中具有 `provisioned` 值的参数。这比 Aurora Serverless v1 中的参数多得多。	仅集群参数组。提供在 `SupportedEngineModes` 属性中具有 `serverless` 值的参数。
集群卷的加密	可选	必需。Amazon Aurora 加密的数据库集群的限制 中的限制适用于所有 Aurora Serverless v1 集群。
跨区域快照	是	必须使用自己的 AWS Key Management Service（AWS KMS）密钥进行加密的快照。
删除数据库集群后保留自动备份	是	否
TLS/SSL	是。支持与预置集群的支持相同。有关使用信息，请参阅将 TLS/SSL 与 Aurora Serverless v2 结合使用。	是。对预置集群的 TLS 支持有一些不同之处。有关使用信息，请参阅将 TLS/SSL 与 Aurora Serverless v1 结合使用。

功能	Aurora Serverless v2 支持	Aurora Serverless v1 支持
克隆	仅限克隆到与 Aurora Serverless v2 兼容的数据库引擎版本和从这个版本克隆。您不能使用克隆从 Aurora Serverless v1 或从早期版本的预置集群升级。	仅限克隆到与 Aurora Serverless v1 兼容的数据库引擎版本和从这个版本克隆。
与 Amazon S3 集成	支持	是
与 AWS Secrets Manager 集成	是	否
将数据库集群快照导出到 S3	是	否
关联 IAM 角色	是	否
将日志上传到 Amazon CloudWatch	可选。您可以选择要打开的日志以及要上传到 CloudWatch 的日志。	打开的所有日志都会自动上传到 CloudWatch。
可用的数据 API	支持	是
查询编辑器可用	支持	是
性能详情	是	否
Amazon RDS 代理可用	是	否
提供了适用于 Aurora PostgreSQL 的 Babelfish	是。对于同时与 Babelfish 和 Aurora Serverless v2 兼容的 Aurora PostgreSQL 版本，支持此功能。	否

使 Aurora Serverless v1 使用案例适应 Aurora Serverless v2

根据您的 Aurora Serverless v1 使用案例，您可以按如下方式调整这种方法以利用 Aurora Serverless v2 功能。

假设您有一个负载较轻的 Aurora Serverless v1 集群，您的首要任务是保持持续可用性，同时更大限度地降低成本。借助 Aurora Serverless v2，您可以配置一个较小的最小 ACU 设置，即 0.5 ACU，而与之相比，Aurora Serverless v1 最小为 1 ACU。您可以通过创建多可用区配置来提高可用性，读取器数据库实例也可以设置为最小 0.5 ACU。

假设您有一个在开发和测试场景中使用的 Aurora Serverless v1 集群。在这种情况下，成本也是一个高优先级问题，但集群并不需要始终可用。Aurora Serverless v2 可以在每个实例完全空闲时自动将其暂停。为此，您可以为集群指定最小容量 0 ACU，如针对 Aurora Serverless v2 缩减到零 ACU 从而自动暂停和恢复中所述。您也可以在不需要集群时手动停止它，然后到了下一个测试或开发周期时再启动它。

假设您有一个工作负载繁重的 Aurora Serverless v1 集群。使用 Aurora Serverless v2 的等效集群可以更精细地扩缩。例如，Aurora Serverless v1 通过将容量翻倍来扩展，例如从 64 ACU 增加到 128 ACU。相比之下，Aurora Serverless v2 数据库实例可按 0.5 ACU 为增量进行扩展。

假设您的工作负载需要的总容量高于 Aurora Serverless v1 中的可用容量。您可以使用多个 Aurora Serverless v2 读取器数据库实例，从写入器数据库实例中分流工作负载的读取密集型部分。您还可以将读取密集型工作负载划分到多个读取器数据库实例。

对于写入密集型工作负载，您可以将带有大型预调配数据库实例的集群配置为写入器。为此，您可以结合使用一个或多个 Aurora Serverless v2 读取器数据库实例。

从 Aurora Serverless v1 集群升级到 Aurora Serverless v2

> **Important**
>
> AWS 已宣布 Aurora Serverless v1 的生命周期终止日期为：2025 年 3 月 31 日。所有在 2025 年 3 月 31 日之前未迁移的 Aurora Serverless v1 集群都将在维护时段内迁移到 Aurora Serverless v2。如果升级失败，Amazon Aurora 会在维护时段内将 Serverless v1 集群转换为具有等效引擎版本的预置集群。如果适用，Amazon Aurora 将在 Amazon RDS 扩展支持中注册转换后的预置集群。有关更多信息，请参阅 RDS Extended Support。

将数据库集群从 Aurora Serverless v1 升级到 Aurora Serverless v2 的过程具有多个步骤。那是因为您无法直接从 Aurora Serverless v1 转换为 Aurora Serverless v2。始终有一个中间步骤涉及将 Aurora Serverless v1 数据库集群转换为预调配集群。

与 Aurora MySQL 兼容的数据库集群

您可以将 Aurora Serverless v1 数据库集群转换为预调配数据库集群，然后使用蓝绿部署对其进行升级并将其转换为 Aurora Serverless v2 数据库集群。我们建议将此过程用于生产环境。有关更多信息，请参阅 使用 Amazon Aurora 蓝绿部署进行数据库更新。

使用蓝绿部署升级运行 Aurora MySQL 版本 2（与 MySQL 5.7 兼容）的 Aurora Serverless v1 集群

1. 将 Aurora Serverless v1 数据库集群转换为预调配的 Aurora MySQL 版本 2 集群。按照从 Aurora Serverless v1 转换为预置中的程序操作。
2. 创建蓝绿部署。按照在 Amazon Aurora 中创建蓝绿部署中的程序操作。
3. 为绿色集群选择一个与 Aurora Serverless v2 兼容的 Aurora MySQL 版本，例如 3.04.1。

   有关兼容版本，请参阅 适用于 Aurora MySQL 的 Aurora Serverless v2。
4. 修改绿色数据库集群的写入器数据库实例，以使用 Serverless v2（db.serverless）数据库实例类。

   有关更多信息，请参阅 将预置写入器或读取器转换为 Aurora Serverless v2。
5. 升级后的 Aurora Serverless v2 数据库集群可用时，从蓝色集群切换到绿色集群。

与 Aurora PostgreSQL 兼容的数据库集群

您可以将 Aurora Serverless v1 数据库集群转换为预调配数据库集群，然后使用蓝绿部署对其进行升级并将其转换为 Aurora Serverless v2 数据库集群。我们建议将此过程用于生产环境。有关更多信息，请参阅 使用 Amazon Aurora 蓝绿部署进行数据库更新。

使用蓝绿部署升级运行 Aurora PostgreSQL 版本 11 的 Aurora Serverless v1 集群

1. 将 Aurora Serverless v1 数据库集群转换为预调配的 Aurora PostgreSQL 集群。按照从 Aurora Serverless v1 转换为预置中的程序操作。
2. 创建蓝绿部署。按照在 Amazon Aurora 中创建蓝绿部署中的程序操作。
3. 为绿色集群选择一个与 Aurora Serverless v2 兼容的 Aurora PostgreSQL 版本，例如 15.3。

   有关兼容版本，请参阅 适用于 Aurora PostgreSQL 的 Aurora Serverless v2。
4. 修改绿色数据库集群的写入器数据库实例，以使用 Serverless v2（db.serverless）数据库实例类。

   有关更多信息，请参阅 将预置写入器或读取器转换为 Aurora Serverless v2。

5. 升级后的 Aurora Serverless v2 数据库集群可用时，从蓝色集群切换到绿色集群。

您也可以将 Aurora Serverless v1 数据库集群直接从 Aurora PostgreSQL 版本 11 升级到版本 13，将其转换为预调配数据库集群，然后将预调配集群转换为 Aurora Serverless v2 数据库集群。

要进行升级，则转换运行 Aurora PostgreSQL 版本 11 的 Aurora Serverless v1 集群

1. 将 Aurora Serverless v1 数据库集群转换为预调配的 Aurora PostgreSQL 集群。按照从 Aurora Serverless v1 转换为预置中的程序操作。
2. 将 Aurora Serverless v1 集群升级到与 Aurora Serverless v2 兼容的 Aurora PostgreSQL 版本 13，例如 13.12。按照升级主要版本中的程序操作。

   有关兼容版本，请参阅 适用于 Aurora PostgreSQL 的 Aurora Serverless v2。
3. 向集群添加 Aurora Serverless v2 读取器数据库实例。有关更多信息，请参阅 添加 Aurora Serverless v2 读取器。
4. 失效转移到 Aurora Serverless v2 数据库实例：

   a. 选择数据库集群的写入器数据库实例。

   b. 对于 Actions（操作），请选择 Failover（失效转移）。

   c. 在确认页面上，选择失效转移。

对于运行 Aurora PostgreSQL 版本 13 的 Aurora Serverless v1 数据库集群，您可以将 Aurora Serverless v1 集群转换为预调配数据库集群，然后将预调配集群转换为 Aurora Serverless v2 数据库集群。

升级运行 Aurora PostgreSQL 版本 13 的 Aurora Serverless v1 集群

1. 将 Aurora Serverless v1 数据库集群转换为预调配的 Aurora PostgreSQL 集群。按照从 Aurora Serverless v1 转换为预置中的程序操作。
2. 向集群添加 Aurora Serverless v2 读取器数据库实例。有关更多信息，请参阅 添加 Aurora Serverless v2 读取器。
3. 失效转移到 Aurora Serverless v2 数据库实例：

   a. 选择数据库集群的写入器数据库实例。

   b. 对于 Actions（操作），请选择 Failover（失效转移）。

   c. 在确认页面上，选择失效转移。

4. 移除读取器实例。

从本地数据库迁移到 Aurora Serverless v2

您可以将本地数据库迁移到 Aurora Serverless v2，就像预调配的 Aurora MySQL 和 Aurora PostgreSQL 一样。

- 对于 MySQL 数据库，您可以使用 `mysqldump` 命令。有关更多信息，请参阅将数据导入 MySQL 或 MariaDB 数据库实例，减少停机时间。
- 对于 PostgreSQL 数据库，您可以使用 `pg_dump` 和 `pg_restore` 命令。有关更多信息，请参阅博客文章Best practices for migrating PostgreSQL databases to Amazon RDS and Amazon Aurora（将 PostgreSQL 数据库迁移到 Amazon RDS 和 Amazon Aurora 的最佳实践）。

使用 Amazon Aurora Serverless v1

> **Important**
>
> AWS 已宣布 Aurora Serverless v1 的生命周期终止日期为：2025 年 3 月 31 日。所有在 2025 年 3 月 31 日之前未迁移的 Aurora Serverless v1 集群都将在维护时段内迁移到 Aurora Serverless v2。如果升级失败，Amazon Aurora 会在维护时段内将 Serverless v1 集群转换为具有等效引擎版本的预置集群。如果适用，Amazon Aurora 将在 Amazon RDS 扩展支持中注册转换后的预置集群。有关更多信息，请参阅 RDS Extended Support。

Amazon Aurora Serverless v1（Amazon Aurora 无服务器版本 1）是适用于 Amazon Aurora 的按需自动扩展配置。Aurora Serverless v1数据库集群是根据应用程序需求向上和向下扩展计算容量的数据库集群。这与您手动管理容量的 Aurora 预置数据库集群形成对比。Aurora Serverless v1 为不频繁、间歇性或不可预测的工作负载提供了一种相对简单、经济高效的选择。它是经济高效的，因为它会自动启动、扩展计算容量以符合应用程序的使用情况并在未使用时关闭。

要了解有关定价的更多信息，请参阅 Amazon Aurora pricing页面上 MySQL-Compatible Edition（MySQL 兼容版）或 PostgreSQL-Compatible Edition（PostgreSQL 兼容版）下的 Serverless Pricing（无服务器定价）。

Aurora Serverless v1 集群具有预置数据库集群所使用的相同类型的高容量、分布式和高可用存储卷。

对于 Aurora Serverless v1 集群，集群卷始终是加密的。您可以选择加密密钥，但不能禁用加密。这意味着您可以像在加密快照上那样在 Aurora Serverless v1 上执行相同的操作。有关更多信息，请参阅 Aurora Serverless v1 和快照。

主题

- Aurora Serverless v1 的区域和版本可用性
- Aurora Serverless v1 的优点
- Aurora Serverless v1 的使用案例
- Aurora Serverless v1 的限制
- Aurora Serverless v1 的配置要求
- 将 TLS/SSL 与 Aurora Serverless v1 结合使用
- Aurora Serverless v1 的工作原理

- 创建 Aurora Serverless v1 数据库集群
- 还原 Aurora Serverless v1 数据库集群
- 修改 Aurora Serverless v1 数据库集群
- 手动扩展 Aurora Serverless v1 数据库集群容量
- 查看 Aurora Serverless v1 数据库集群
- 删除 Aurora Serverless v1 数据库集群
- Aurora Serverless v1 和 Aurora 数据库引擎版本

Aurora Serverless v1 的区域和版本可用性

功能可用性和支持因每个 Aurora 数据库引擎的特定版本以及 AWS 区域而异。有关 Aurora 和 Aurora Serverless v1 的版本和区域可用性的更多信息，请参阅 Aurora Serverless v1。

Aurora Serverless v1 的优点

Aurora Serverless v1 具备下述优点：

- 比预置更简单 - Aurora Serverless v1 消除了管理数据库实例和容量的大量复杂性。
- 可扩展 - Aurora Serverless v1 可根据需要无缝地扩展计算和内存容量，而无需中断客户端连接。
- 经济高效 – 在使用 Aurora Serverless v1 时，您只需为所使用的数据库资源按秒付费。
- 高度可用存储 – Aurora Serverless v1 使用具有容错能力的同一分布式存储系统，该系统具有作为 Aurora 的六向复制以防数据丢失。

Aurora Serverless v1 的使用案例

Aurora Serverless v1 专为以下使用案例而设计：

- 不常使用的应用程序 – 您有每天或每周仅使用数次几分钟的应用程序，如低容量博客网站。有了 Aurora Serverless v1，您只需为所使用的数据库资源按秒付费。
- 新应用程序 – 您正在部署新应用程序，但不确定所需的实例大小。使用 Aurora Serverless v1，您可以创建数据库终端节点并让数据库自动扩展到应用程序的容量需求。
- 可变工作负载 – 您正在运行一个使用率比较低的应用程序，高峰时间为每天数次 30 分钟到几个小时，或每年几次。示例包括人力资源、预算和运营报告应用程序。有了 Aurora Serverless v1，您不再需要为峰值或平均容量进行预置。

- 不可预测的工作负载 – 您每天运行的工作负载会突然出现无法预测的活动增加。例如，在开始下雨时显示活动涌现的流量网站。有了 Aurora Serverless v1，您的数据库可自动扩展容量以满足应用程序的峰值负载需求，然后在活动涌现结束时再收缩。
- 开发和测试数据库 – 您的开发人员在工作时间使用数据库，但在夜间或周末不需要它们。有了 Aurora Serverless v1，您的数据库会在未使用时自动关闭。
- 多租户应用程序 – 有了 Aurora Serverless v1，您无需为队列中的每个应用程序单独地管理数据库容量。Aurora Serverless v1 会为您管理各个数据库容量。

Aurora Serverless v1 的限制

以下限制适用于 Aurora Serverless v1：

- **Important**

  AWS 已宣布 Aurora Serverless v1 的生命周期终止日期为：2025 年 3 月 31 日。所有在 2025 年 3 月 31 日之前未迁移的 Aurora Serverless v1 集群都将在维护时段内迁移到 Aurora Serverless v2。如果升级失败，Amazon Aurora 会在维护时段内将 Serverless v1 集群转换为具有等效引擎版本的预置集群。如果适用，Amazon Aurora 将在 Amazon RDS 扩展支持中注册转换后的预置集群。有关更多信息，请参阅 RDS Extended Support。

- Aurora Serverless v1 不支持以下功能：
  - Aurora 全局数据库
  - Aurora 副本
  - AWS Identity and Access Management (IAM) 数据库身份验证
  - 正在 Aurora 中回溯
  - 数据库活动流
  - Kerberos 身份验证
  - 性能详情
  - RDS 代理
  - 在 AWS Management Console中查看日志
- 如果保持打开状态超过一天，与 Aurora Serverless v1 数据库集群的连接将自动关闭。
- 所有 Aurora Serverless v1 数据库集群都有以下限制：
  - 您无法将 Aurora Serverless v1 快照导出到 Amazon S3 存储桶。

- AWS Database Migration Service 数据库集群无法使用 Aurora Serverless v1 和更改数据捕获 (CDC)。只有预置的 Aurora 数据库集群支持将 CDC 和 AWS DMS 作为源。
- 您无法将数据保存到 Amazon S3 中的文本文件，也无法从 S3 中将文本文件数据加载到 Aurora Serverless v1。
- 您无法将 IAM 角色附加到 Aurora Serverless v1 数据库集群。但是，您可以通过将 Aurora Serverless v1 扩展与 `aws_s3` 函数和 `aws_s3.table_import_from_s3` 参数一起使用，将数据从 Amazon S3 加载到 `credentials`。有关更多信息，请参阅 将 Amazon S3 中的数据导入到 Aurora PostgreSQL 数据库集群。
- 使用查询编辑器时，会为数据库凭证创建一个 Secrets Manager 密钥来访问数据库。如果您从查询编辑器中删除凭证，则也会从 Secrets Manager 中删除关联的密钥。删除密钥后将无法恢复。

- 运行 Aurora Serverless v1 的基于 Aurora MySQL 的数据库集群不支持以下内容：
  - 从 Aurora MySQL 数据库集群中调用 AWS Lambda 函数。但是，AWS Lambda 函数可以调用 Aurora Serverless v1 数据库集群。
  - 从非 Aurora MySQL 或 RDS for MySQL 数据库实例还原快照。
  - 使用基于二进制日志 (binlog) 的复制来复制数据。无论您基于 Aurora MySQL 的数据库集群 Aurora Serverless v1 是复制的源还是目标，都存在此限制。要将数据从 Aurora 之外的 MySQL 数据库实例（例如在 Amazon EC2 上运行的实例）复制到 Aurora Serverless v1 数据库集群，建议您考虑使用 AWS Database Migration Service。有关更多信息，请参阅 AWS Database Migration Service 用户指南。
  - 创建具有基于主机的访问权限的用户（`'username'@'IP_address'`）。这是因为 Aurora Serverless v1 在客户端和数据库主机之间使用路由器队列进行无缝扩展。Aurora Serverless 数据库集群看到的 IP 地址是路由器主机（而不是您的客户端）的 IP 地址。有关更多信息，请参阅 Aurora Serverless v1 架构。

    而是使用通配符（`'username'@'%'`）。
- 运行 Aurora Serverless v1 的基于 Aurora PostgreSQL 的数据库集群具有以下限制：
  - Aurora PostgreSQL 查询计划管理（`apg_plan_management` 扩展）不受支持。
  - 不支持 Amazon RDS PostgreSQL 和 Aurora PostgreSQL 中可用的逻辑复制功能。
  - 不支持出站通信（例如 Amazon RDS for PostgreSQL 扩展启用的出站通信）。例如，您无法通过 `postgres_fdw/dblink` 扩展访问外部数据。有关 RDS PostgreSQL 扩展的更多信息，请参阅 RDS 用户指南中的 Amazon RDS 上的 PostgreSQL。
  - 目前，不推荐使用某些 SQL 查询和命令。这些包括会话级别的咨询锁、临时关系、异步通知 (`LISTEN`) 和带有保留 (`DECLARE name ... CURSOR WITH HOLD FOR query`) 的游标。此外，`NOTIFY` 命令会阻止缩放，因此不推荐使用。

有关更多信息，请参阅 Aurora Serverless v1 的自动扩展。

- 您无法为 Aurora Serverless v1 数据库集群设置首选的自动备份时段。
- 您可以为 Aurora Serverless v1 数据库集群设置维护时段。有关更多信息，请参阅 调整首选数据库集群维护时段。

Aurora Serverless v1 的配置要求

创建 Aurora Serverless v1 数据库集群时，请注意以下要求：

- 为每个数据库引擎使用以下特定端口号：
  - Aurora MySQL – 3306
  - Aurora PostgreSQL – 5432
- 在 Virtual Private Cloud (VPC) 中基于 Amazon VPC 服务创建 Aurora Serverless v1 数据库集群。在 VPC 中创建 Aurora Serverless v1 数据库集群时，您将使用分配给 VPC 的五十 (50) 个接口和网关负载均衡器终端节点中的两 (2) 个。这些终端节点是自动为您创建的。要增加配额，您可以联系支持。有关更多信息，请参阅 Amazon VPC 配额。
- 您无法为 Aurora Serverless v1 数据库集群提供公有 IP 地址。您只能从 VPC 内部访问 Aurora Serverless v1 数据库集群。
- 在不同的可用区中为用于 Aurora Serverless v1 数据库集群的数据库子网组创建子网。换句话说，同一可用区内不能有多于一个的子网。
- 对 Aurora Serverless v1 数据库集群使用的子网组的更改不会应用于集群。
- 您可以从 Aurora Serverless v1 中访问 AWS Lambda 数据库集群。为此，您必须将 Lambda 函数配置为与 Aurora Serverless v1 数据库集群在相同的 VPC 中运行。有关使用 AWS Lambda 的更多信息，请参阅 AWS Lambda 开发人员指南中的配置 Lambda 函数以访问 Amazon VPC 中的资源。

将 TLS/SSL 与 Aurora Serverless v1 结合使用

默认情况下，Aurora Serverless v1 使用传输层安全性/安全套接字层 (TLS/SSL) 协议来加密客户端和 Aurora Serverless v1 数据库集群之间的通信。它支持 TLS/SSL 1.0、1.1 和 1.2 版本。您无需将 Aurora Serverless v1 数据库集群配置为使用 TLS/SSL。

但是，以下限制适用：

- 中国（北京）AWS 区域目前不提供针对 Aurora Serverless v1 数据库集群的 TLS/SSL 支持。

- 为基于 Aurora MySQL 的 Aurora Serverless v1 数据库集群创建数据库用户时，请勿使用 REQUIRE 子句获取 SSL 权限。这样做会阻止用户连接到 Aurora 数据库实例。
- 对于 MySQL Client 和 PostgreSQL 客户端实用程序，在客户端和 Aurora Serverless v1 之间使用 TLS/SSL 时，您可能在其他环境中使用的会话变量不起作用。
- 对于 MySQL 客户端，当使用 TLS/SSL 的 VERIFY_IDENTITY 模式进行连接时，您目前需要使用兼容 MySQL 8.0 的 mysql 命令。有关更多信息，请参阅连接到运行 MySQL 数据库引擎的数据库实例。

根据用于连接到 Aurora Serverless v1 数据库集群的客户端，您可能无需指定 TLS/SSL 即可获取加密连接。例如，要使用 PostgreSQL 客户端连接到运行 Aurora PostgreSQL 兼容版的 Aurora Serverless v1 数据库集群，请像平常一样进行连接。

```
psql -h endpoint -U user
```

输入密码后，PostgreSQL 客户端会显示您看到连接详细信息，包括 TLS/SSL 版本和密码。

```
psql (12.5 (Ubuntu 12.5-0ubuntu0.20.04.1), server 10.12)
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256,
 compression: off)
Type "help" for help.
```

Important

默认情况下，Aurora Serverless v1 使用传输层安全性/安全套接字层（TLS/SSL）协议来加密连接，除非客户端应用程序禁用了 SSL/TLS。TLS/SSL 连接在路由器队列中终止。路由器队列与 Aurora Serverless v1 数据库集群之间的通信发生在服务的内部网络边界内。

您可以检查客户端连接的状态，以检查与 Aurora Serverless v1 的连接是否为 TLS/SSL 加密。对于客户端应用程序和 Aurora Serverless v1 之间的通信，PostgreSQL pg_stat_ssl 和 pg_stat_activity 表及其 ssl_is_used 函数不显示 TLS/SSL 状态。同样，TLS/SSL 状态也不能通过 MySQL status 语句派生。

force_ssl for PostgreSQL 和 require_secure_transport for MySQL 的 Aurora 集群参数以前在 Aurora Serverless v1 中不支持。这些参数现在可用于 Aurora Serverless v1。有关 Aurora Serverless v1 支持的参数的完整列表，请调用 DescribeEngineDefaultClusterParameters API 操作。有关参数组和 Aurora Serverless v1 的更多信息，请参阅 Aurora Serverless v1 的参数组。

要使用 MySQL 客户端连接到运行 Aurora MySQL 兼容版的 Aurora Serverless v1 数据库集群，请在请求中指定 TLS/SSL。以下示例包括从 Amazon Trust Services 下载的 Amazon root CA 1 信任存储，这是成功连接所必需的。

```
mysql -h endpoint -P 3306 -u user -p --ssl-ca=amazon-root-CA-1.pem --ssl-mode=REQUIRED
```

出现提示时请输入密码。MySQL 显示器将很快打开。您可以使用 `status` 命令确认会话是否已加密。

```
mysql> status
--------------
mysql  Ver 14.14 Distrib 5.5.62, for Linux (x86_64) using readline 5.1
Connection id:          19
Current database:
Current user:           ***@*******
SSL:                    Cipher in use is ECDHE-RSA-AES256-SHA
...
```

要了解有关使用 MySQL 客户端连接到 Aurora MySQL 数据库的更多信息，请参阅连接到运行 MySQL 数据库引擎的数据库实例。

Aurora Serverless v1 支持 MySQL 客户端 (`mysql`) 和 PostgreSQL 客户端 (`psql`) 可用的所有 TLS/SSL 模式，包括下表中列出的模式。

TLS/SSL 模式的说明	mysql	psql
不使用 TLS/SSL 进行连接。	DISABLED	disable
首先尝试使用 TLS/SSL 进行连接，但如有必要，请回退到非 SSL。	PREFERRED	首选（默认）
使用 TLS/SSL 强制执行。	REQUIRED	require
强制执行 TLS/SSL 并验证 CA。	VERIFY_CA	verify-ca
强制执行 TLS/SSL，验证 CA 并验证 CA 主机名。	VERIFY_IDENTITY	verify-full

Aurora Serverless v1 使用通配符证书。如果您在使用 TLS/SSL 时指定“验证 CA”或“验证 CA 和 CA 主机名”选项，请首先从 Amazon Trust Services 下载 Amazon root CA 1 信任存储。完成此操作后，您可以在客户端命令中识别此 PEM 格式的文件。要使用 PostgreSQL 客户端执行此操作：

对于 Linux、macOS 或 Unix：

```
psql 'host=endpoint user=user sslmode=require sslrootcert=amazon-root-CA-1.pem
 dbname=db-name'
```

要了解有关通过 Postgres 客户端使用 Aurora PostgreSQL 数据库的更多信息，请参阅连接到运行 PostgreSQL 数据库引擎的数据库实例。

有关更多连接到 Aurora 数据库集群的一般信息，请参阅 连接到 Amazon Aurora 数据库集群。

支持的用于 Aurora Serverless v1 数据库集群连接的密码套件

通过使用可配置的密码套件，您可以更好地控制数据库连接的安全性。您可以指定想要允许保护客户端与数据库的 TLS/SSL 连接的密码套件列表。使用可配置的密码套件，您现在可以控制数据库服务器接受的连接加密。这样做可以防止使用不安全或不再使用的密码。

基于 Aurora MySQL 的 Aurora Serverless v1 数据库集群支持与 Aurora MySQL 预配的数据库集群相同的密码套件。有关这些密码套件的信息，请参阅 配置密码套件以连接到 Aurora MySQL 数据库集群。

基于 Aurora PostgreSQL 的 Aurora Serverless v1 数据库集群不支持密码套件。

Aurora Serverless v1 的工作原理

> **Important**
>
> AWS 已宣布 Aurora Serverless v1 的生命周期终止日期为：2025 年 3 月 31 日。所有在 2025 年 3 月 31 日之前未迁移的 Aurora Serverless v1 集群都将在维护时段内迁移到 Aurora Serverless v2。如果升级失败，Amazon Aurora 会在维护时段内将 Serverless v1 集群转换为具有等效引擎版本的预置集群。如果适用，Amazon Aurora 将在 Amazon RDS 扩展支持中注册转换后的预置集群。有关更多信息，请参阅 RDS Extended Support。

接下来，您可以了解 Aurora Serverless v1 的工作原理。

主题

- Aurora Serverless v1 架构
- Aurora Serverless v1 的自动扩展
- 容量更改超时操作
- 暂停和恢复 Aurora Serverless v1
- 确定 Aurora Serverless v1 的最大数据库连接数
- Aurora Serverless v1 的参数组
- Aurora Serverless v1 的日志记录
- Aurora Serverless v1 和维护
- Aurora Serverless v1 和失效转移
- Aurora Serverless v1 和快照

Aurora Serverless v1 架构

下图显示了 Aurora Serverless v1 架构的概览。



您可指定 Aurora 容量单元 (ACU)，而不是配置和管理数据库服务器。每个 ACU 是约 2 GB 的内存、相应的 CPU 和网络的组合。数据库存储从 10 GiB 自动扩展到 128 tebibytes (TiB)，与标准 Aurora 数据库集群中的存储相同。

可以指定最小和最大 ACU。最小 Aurora 容量单元 是数据库集群可缩减到的最低 ACU。最大 Aurora 容量单元 是数据库集群可扩展到的最高 ACU。根据您的设置，Aurora Serverless v1 自动创建 CPU 使用率、连接和可用内存阈值的扩展规则。

Aurora Serverless v1 管理 AWS 区域中的热资源池以最大程度地减少扩展时间。在 Aurora Serverless v1 将新资源添加到 Aurora 数据库集群时，它使用路由器机群将活动客户端连接切换到新资源。在任何给定时间，您都只需为您的 Aurora 数据库集群中正在主动使用的 ACU 付费。

Aurora Serverless v1 的自动扩展

分配给 Aurora Serverless v1 数据库集群的容量可根据客户端应用程序生成的负载无缝向上和向下扩展。在这里，负载是 CPU 利用率和连接数。当容量受这两个因素之一的限制时，Aurora Serverless v1 可以向上扩展。当它检测到性能问题时，Aurora Serverless v1 也可以向上扩展以解决这些问题。

您可以查看 Aurora Serverless v1 中 AWS Management Console 集群的扩展事件。在自动扩展期间，Aurora Serverless v1 会重置 `EngineUptime` 指标。重置指标值的值并不意味着无缝扩展存在问题，也不意味着 Aurora Serverless v1 连接中断。这只是新容量下正常运行时间的起点。了解有关指标的更多信息，请参阅 监控 Amazon Aurora 集群中的指标。

当您的 Aurora Serverless v1 数据库集群没有活动连接时，它可以缩减到零容量（0 ACU）。要了解更多信息，请参阅“暂停和恢复 Aurora Serverless v1”。

当它确实需要执行扩展操作时，Aurora Serverless v1 首先尝试识别扩展点，此时没有处理任何查询。Aurora Serverless v1 可能会由于以下原因而无法找到扩展点：

- 长时间运行的查询
- 进行中的交易
- 临时表或表锁定

在查找扩展点时，为了提高 Aurora Serverless v1 数据库集群的成功率，建议您避免长时间运行的查询和长时间运行的事务。要了解有关阻止扩展的操作以及如何避免这些操作的更多信息，请参阅使用 Aurora Serverless v1 的最佳实践。

预设情况下，Aurora Serverless v1 尝试在 5 分钟（300 秒）内找到扩展点。您可以在创建或修改集群时指定不同的超时期限。超时期限可以介于 60 秒至 10 分钟 (600 秒) 之间。如果 Aurora Serverless v1 在指定期限内找不到扩展点，自动扩展操作将超时。

默认情况下，如果自动扩展在超时之前找不到扩展点，那么 Aurora Serverless v1 会将集群保持为当前容量。当您通过选择强制执行容量更改选项创建或修改 Aurora Serverless v1 数据库集群时，可以更改此默认行为。有关更多信息，请参阅 容量更改超时操作。

容量更改超时操作

如果自动扩展在没有找到扩展点的情况下超时，预设情况下 Aurora 将保持当前容量。您可以通过启用 Force the capacity change（强制执行容量更改）选项来选择让 Aurora 强制执行更改。创建集群时，此选项在 Create database（创建数据库）页面的 Autoscaling timeout and action（自动扩展超时和操作）部分中可供选择。

默认情况下，Force the capacity change（强制执行容量更改）选项处于取消选中状态。如果扩展操作已超时而未找到扩展点，则不要选中此选项以使 Aurora Serverless v1 数据库集群的容量保持不变。

选择此选项会导致 Aurora Serverless v1 数据库集群强制执行容量更改，即使没有扩展点也是如此。在选择此选项之前，请注意此选择带来的后果：

- 所有进行中的事务都会中断，并显示以下错误消息。

  Aurora MySQL 版本 2 – 错误 1105（HY000）：由于无缝扩缩，最后一个事务已中止。请重试。

  您可以在 Aurora Serverless v1 数据库集群可用后立即重新提交事务。
- 与临时表和表锁定的连接会中断。

  建议您仅在应用程序可以从中断的连接或未完成的事务中恢复时，才选择 Force the capacity change（强制容量更改）选项。

创建 Aurora Serverless v1 数据库集群时在 AWS Management Console 中所做的选择存储在 `ScalingConfigurationInfo` 对象的 `SecondsBeforeTimeout` 和 `TimeoutAction` 属性中。创建集群时，`TimeoutAction` 属性的值设置为以下值之一：

- `RollbackCapacityChange` - 当您选择 Roll back the capacity change（回滚容量更改）选项时设置此值。这是默认行为。
- `ForceApplyCapacityChange` – 当您选择 Force the capacity change（强制容量更改）选项时设置此值。

您可以通过使用 describe-db-clusters AWS CLI 命令在现有 Aurora Serverless v1 数据库集群上获取此属性的值，如下所示。

对于 Linux、macOS 或 Unix：

```
aws rds describe-db-clusters --region region \
  --db-cluster-identifier your-cluster-name \
  --query '*[].{ScalingConfigurationInfo:ScalingConfigurationInfo}'
```

对于 Windows：

```
aws rds describe-db-clusters --region region ^
  --db-cluster-identifier your-cluster-name ^
  --query "*[].{ScalingConfigurationInfo:ScalingConfigurationInfo}"
```

例如，下面显示了美国西部（加利福尼亚北部）区域中名为 `west-coast-sles` 的 Aurora Serverless v1 数据库集群的查询和响应。

```
$ aws rds describe-db-clusters --region us-west-1 --db-cluster-identifier west-coast-
sles
--query '*[].{ScalingConfigurationInfo:ScalingConfigurationInfo}'

[
    {
        "ScalingConfigurationInfo": {
            "MinCapacity": 1,
            "MaxCapacity": 64,
            "AutoPause": false,
            "SecondsBeforeTimeout": 300,
            "SecondsUntilAutoPause": 300,
            "TimeoutAction": "RollbackCapacityChange"
        }
    }
]
```

如响应所示，此 Aurora Serverless v1 数据库集群使用默认设置。

有关更多信息，请参阅“创建 Aurora Serverless v1 数据库集群”。在创建您的 Aurora Serverless v1 之后，您还可以随时修改超时操作和其他容量设置。要了解如何操作，请参阅修改 Aurora Serverless v1 数据库集群。

暂停和恢复 Aurora Serverless v1

您可以选择在无任何活动的指定时间段内暂停 Aurora Serverless v1 数据库集群。在暂停数据库集群之前指定无任何活动的时间长度。选择此选项后，默认的非活动时间为 5 分钟，但您可以更改此值。此选项为可选设置。

暂停数据库集群后，不会发生任何计算或内存活动，而且您只需支付存储费用。如果暂停 Aurora Serverless v1 数据库集群时请求数据库连接，数据库集群将自动恢复并处理连接请求。

当数据库集群恢复活动时，它的容量与 Aurora 暂停集群时的容量相同。ACU 的数量取决于在暂停集群之前 Aurora 扩展或缩减的程度。

> **Note**
>
> 如果暂停数据库集群超过 7 天，则可能会使用快照对数据库集群进行备份。在这种情况下，当有连接到快照的请求时，Aurora 将从快照还原数据库集群。

确定 Aurora Serverless v1 的最大数据库连接数

以下示例用于与 MySQL 5.7 兼容的 Aurora Serverless v1 数据库集群。如果已配置对 MySQL 客户端或查询编辑器的访问权限，则可以使用它们。有关更多信息，请参阅 在查询编辑器中运行查询。

查找最大数据库连接数

1. 使用 AWS CLI 查找 Aurora Serverless v1 数据库集群的容量范围。

```
aws rds describe-db-clusters \
    --db-cluster-identifier my-serverless-57-cluster \
    --query 'DBClusters[*].ScalingConfigurationInfo|[0]'
```

结果显示，其容量范围为 1-4 ACU。

```
{
    "MinCapacity": 1,
    "AutoPause": true,
    "MaxCapacity": 4,
    "TimeoutAction": "RollbackCapacityChange",
    "SecondsUntilAutoPause": 3600
}
```

2. 运行以下 SQL 查询以查找最大连接数。

```
select @@max_connections;
```

显示的结果适用于集群的最小容量 1 ACU。

```
@@max_connections
90
```

3. 将集群扩展到 8–32 ACU。

有关扩展的更多信息，请参阅修改 Aurora Serverless v1 数据库集群。

4. 确认容量范围。

```
{
    "MinCapacity": 8,
    "AutoPause": true,
    "MaxCapacity": 32,
    "TimeoutAction": "RollbackCapacityChange",
    "SecondsUntilAutoPause": 3600
}
```

5. 查找最大连接数。

```
select @@max_connections;
```

显示的结果适用于集群的最小容量 8 ACU。

```
@@max_connections
1000
```

6. 将集群扩展到最大可能的 256–256 ACU。

7. 确认容量范围。

```
{
    "MinCapacity": 256,
    "AutoPause": true,
    "MaxCapacity": 256,
    "TimeoutAction": "RollbackCapacityChange",
    "SecondsUntilAutoPause": 3600
```

```
}
```

8. 查找最大连接数。

```
select @@max_connections;
```

显示的结果适用于 256 ACU。

```
@@max_connections
6000
```

> **Note**
>
> max_connections 值不会随 ACU 的数量线性扩展。

9. 将集群向下缩减至 1-4 ACU。

```
{
    "MinCapacity": 1,
    "AutoPause": true,
    "MaxCapacity": 4,
    "TimeoutAction": "RollbackCapacityChange",
    "SecondsUntilAutoPause": 3600
}
```

这次，max_connections 值适用于 4 ACU。

```
@@max_connections
270
```

10. 让集群缩减至 2 ACU。

```
@@max_connections
180
```

如果您已将集群配置为在空闲一定时间后暂停，则它将缩小到 0 ACU。但是，max_connections 不会降至低于 1 ACU 的值。

```
@@max_connections
```

```
90
```

Aurora Serverless v1 的参数组

当您创建您的 Aurora Serverless v1 数据库集群时，您可以选择特定 Aurora 数据库引擎和关联的数据库集群参数组。与预置的 Aurora 数据库集群不同，Aurora Serverless v1 数据库集群有一个只配置了数据库集群参数组的读/写数据库实例—它没有单独的数据库参数组。在自动扩展期间，Aurora Serverless v1 需要能够更改集群的参数，以便在容量增加或减少时发挥最佳作用。因此，对于 Aurora Serverless v1 数据库集群，您对特定数据库引擎类型的参数所做的一些更改可能不适用。

例如，Aurora PostgreSQL–基于 Aurora Serverless v1 的数据库集群不能使用 `apg_plan_mgmt.capture_plan_baselines` 和其他参数，这些参数可能会用于预置的 Aurora PostgreSQL 数据库集群以进行查询计划管理。

通过使用 describe-engine-default-cluster-parameters CLI 命令并查询 AWS 区域，您可以获得各种 Aurora 数据库引擎的默认参数组的默认值列表。以下是可用于 `--db-parameter-group-family` 选项的值。

Aurora MySQL 版本 2	`aurora-mysql5.7`
Aurora PostgreSQL 版本 11	`aurora-postgresql11`
Aurora PostgreSQL 版本 13	`aurora-postgresql13`

建议您使用 AWS 访问密钥 ID 和 AWS 秘密访问密钥配置 AWS CLI，并在使用 AWS CLI 命令之前设置您的 AWS 区域。为 CLI 配置提供区域，于是您无需在运行命令时输入 `--region` 参数。要了解有关配置 AWS CLI 的更多信息，请参阅《AWS Command Line Interface 用户指南》中的配置基础知识。

以下示例从 Aurora MySQL 版本 2 的默认数据库集群组中获取参数列表。

对于 Linux、macOS 或 Unix：

```
aws rds describe-engine-default-cluster-parameters \
  --db-parameter-group-family aurora-mysql5.7 --query \
  'EngineDefaults.Parameters[*].
{ParameterName:ParameterName,SupportedEngineModes:SupportedEngineModes} | [?
contains(SupportedEngineModes, `serverless`) == `true`] | [*].{param:ParameterName}' \
```

```
  --output text
```

对于 Windows：

```
aws rds describe-engine-default-cluster-parameters ^
   --db-parameter-group-family aurora-mysql5.7 --query ^
   "EngineDefaults.Parameters[*].
{ParameterName:ParameterName,SupportedEngineModes:SupportedEngineModes} | [?
contains(SupportedEngineModes, 'serverless') == `true`] | [*].{param:ParameterName}" ^
   --output text
```

为 Aurora Serverless v1 修改参数值

如 Amazon Aurora 的参数组 中所述，无论默认参数组的类型如何（数据库集群参数组、数据库参数组），都不能直接更改其值。相反，您可以根据 Aurora 数据库引擎的默认数据库集群参数组创建自定义参数组，并根据需要更改该参数组的设置。例如，您可能希望更改 Aurora Serverless v1 数据库集群的某些设置以记录查询或将数据库引擎特定日志上传到 Amazon CloudWatch。

创建自定义数据库集群参数组

1. 登录AWS Management Console并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 选择参数组。
3. 选择创建参数组，打开参数组详细信息窗格。
4. 为要用于 Aurora Serverless v1 数据库集群的数据库引擎选择适当的默认数据库集群组。请务必选择以下选项：
   a. 对于 Parameter group family（参数组系列），请为所选的数据库引擎选择合适的系列。请确保您的选择名称中有 `aurora-` 前缀。
   b. 对于 Type（类型），请选择 DB Cluster Parameter Group（数据库集群参数组）。
   c. 在组名称和描述中，为您或可能需要使用 Aurora Serverless v1 数据库集群及其参数的其他人输入有意义的名称。
   d. 选择 Create（创建）。

您的自定义数据库集群参数组将添加到您的 AWS 区域中提供的参数组列表中。现在，当您创建新的 Aurora Serverless v1 数据库集群时，您可以使用自定义数据库集群参数组。您还可以修改现有 Aurora Serverless v1 数据库集群，以使用自定义数据库集群参数组。使用自定义数据库集群参数组启

动 Aurora Serverless v1 数据库集群后，您可以使用 AWS Management Console 或 AWS CLI 更改动态参数的值。

您还可以使用控制台查看自定义数据库集群参数组中的值与默认数据库集群参数组的值并排比较，如以下屏幕截图所示。



当您更改活动数据库集群上的参数值时，Aurora Serverless v1 将启动无缝扩展以应用参数更改。如果您的 Aurora Serverless v1 数据库集群处于已暂停状态，它会恢复并开始扩展，以便进行更改。参数组更改的扩展操作始终为 forces a capacity change（强制容量更改），因此请注意，如果在扩展期间找不到扩展点，修改参数可能会导致连接中断。

Aurora Serverless v1 的日志记录

默认情况下，Aurora Serverless v1 的错误日志已启用并自动上传到 Amazon CloudWatch。您也可以让 Aurora Serverless v1 数据库集群将 Aurora 数据库引擎特定的日志上传到 CloudWatch。为此，请在自定义数据库集群参数组中启用配置参数。然后，您的 Aurora Serverless v1 数据库集群会将所有可用日志上传到 Amazon CloudWatch。此时，您可以使用 CloudWatch 来分析日志数据、创建警报和查看指标。

对于 Aurora MySQL，下表显示了您可以启用的日志。启用后，它们自动从 Aurora Serverless v1 数据库集群上传到 Amazon CloudWatch。

Aurora MySQL 日志	描述
general_log	创建常规日志。设置为 1 以开启。默认为关闭 (0)。
log_queries_not_using_indexes	将任何查询记录到不使用索引的慢速查询日志中。默认为关闭 (0)。设置为 1 以开启此日志。
long_query_time	防止快速运行的查询记录在慢速查询日志中。可以设置为 0 到 3,1536,000 之间的浮动值。默认值为 0（不活动）。
server_audit_events	要在日志中捕获的事件列表。支持的值有 CONNECT、QUERY、QUERY_DCL、QUERY_DDL、QUERY_DML 和 TABLE。
server_audit_logging	设置为 1 以打开服务器审核日志记录。如果启用此选项，则可以通过在 server_audit_events 参数中列出审核事件来指定要发送到 CloudWatch 的审计事件。
slow_query_log	创建慢速查询日志。设置为 1 以打开慢速查询日志。默认为关闭 (0)。

有关更多信息，请参阅 在 Amazon Aurora MySQL 数据库集群中使用高级审计。

对于 Aurora PostgreSQL，下表显示了您可以启用的日志。启用后，它们自动从 Aurora Serverless v1 数据库集群上传到 Amazon CloudWatch，并附带常规的错误日志。

Aurora PostgreSQL 日志	描述
log_connections	默认情况下为已启用，无法更改。它会记录所有新客户端连接的详细信息。

Aurora PostgreSQL 日志	描述
log_disconnections	默认情况下为已启用，无法更改。记录所有客户端断开连接。
log_hostname	默认情况下处于关闭状态，无法更改。不记录主机名。
log_lock_waits	默认值为 0（关闭）。设置为 1 以记录锁定等待。
log_min_duration_statement	语句在记录前运行的最短持续时间（以毫秒为单位）。
log_min_messages	设置记录的消息级别。支持的值为 debug5、debug4、debug3、debug2、debug1、inf 将性能数据记录到 postgres 日志，将值设置为 debug1。
log_temp_files	记录指定千字节 (kB) 以上的临时文件的使用情况。
log_statement	控制被记录的特定 SQL 语句。支持的值有 none、ddl、mod 和 all。默认为 none。

为 Aurora Serverless v1 数据库集群的 Aurora MySQL 或 Aurora PostgreSQL 启用日志后，可以在 CloudWatch 中查看日志。

通过 Amazon CloudWatch 查看 Aurora Serverless v1 日志

Aurora Serverless v1 自动上传（“发布”）至自定义数据库集群参数组中启用的 Amazon CloudWatch 所有日志。您无需选择或指定日志类型。启用日志配置参数后，将立即开始上传日志。如果您以后禁用日志参数，则将停止进一步上传。但是，所有已发布到 CloudWatch 的日志都将保留，直到您删除它们。

有关将 CloudWatch 与 Aurora MySQL 日志配合使用的更多信息，请参阅 在 Amazon CloudWatch 中监控日志事件。

有关 CloudWatch 和 Aurora PostgreSQL 的更多信息，请参阅将 Aurora PostgreSQL 日志发布到 Amazon CloudWatch Logs。

要查看 Aurora Serverless v1 数据库集群的日志

1. 通过以下网址打开 CloudWatch 控制台：https://console.aws.amazon.com/cloudwatch/。
2. 选择您的 AWS 区域。
3. 选择 Log groups（日志组）。
4. 从列表中选择 Aurora Serverless v1 数据库集群日志。对于错误日志，命名模式如下。

```
/aws/rds/cluster/cluster-name/error
```

例如，在以下屏幕截图中，您可以找到为名为 `western-sles` 的 Aurora PostgreSQL Aurora Serverless v1 数据库集群发布的日志列表。您还可以找到 Aurora MySQL Aurora Serverless v1 数据库集群 `west-coast-sles` 的多个列表。选择感兴趣的日志以开始探索其内容。



Aurora Serverless v1 和维护

系统将自动为您执行 Aurora Serverless v1 数据库集群的维护（例如应用最新功能、修复程序和安全更新）。Aurora Serverless v1 有一个维护时段，您可以在 AWS Management Console中 Aurora Serverless v1 数据库集群的维护和备份中查看。您可以查找可能执行维护的日期和时间，以及 Aurora Serverless v1 数据库集群是否有任何待处理的维护，如下图所示。



您可以在创建 Aurora Serverless v1 数据库集群时设置维护时段，并可以稍后修改该时段。有关更多信息，请参阅 调整首选数据库集群维护时段。

维护时段用于定期的主要版本升级。在扩展过程中，会立即应用次要版本升级和补丁。扩展根据您的 `TimeoutAction` 设置进行：

- `ForceApplyCapacityChange` – 将立即应用更改。
- `RollbackCapacityChange` – Aurora 会在第一次尝试应用补丁 3 天后强制更新集群。

与在没有适当扩展点的情况下强制进行的任何更改一样，这可能会中断您的工作负载。

Aurora Serverless v1 会尽可能以无中断的方式执行维护。当需要维护时，Aurora Serverless v1 数据库集群会扩展容量以处理必要的操作。在进行扩展之前，Aurora Serverless v1 会查找扩展点。如有必要，此操作最多可以持续三天。

在 Aurora Serverless v1 找不到扩展点的每一天结束时，它都会创建一个集群事件。此事件会通知您有关待处理的维护以及需要进行扩展以执行维护。该通知包括 Aurora Serverless v1 可以强制数据库集群扩展的日期。

有关更多信息，请参阅 容量更改超时操作。

Aurora Serverless v1 和失效转移

如果 Aurora Serverless v1 数据库集群的数据库实例变得不可用，或者其所在的可用区（AZ）出现故障，Aurora 会在其他可用区中重新创建数据库实例。但是，Aurora Serverless v1 集群不是多可用区集群。那是因为它只包含单个可用区中的单个数据库实例。因此，这个失效转移机制需要的时间比具有预置实例或 Aurora Serverless v2 实例的 Aurora 集群更长。未定义 Aurora Serverless v1 失效转移时间，因为它取决于给定 AWS 区域中的其他可用区的需求和容量可用性。

由于 Aurora 分离计算容量和存储，因此，集群的存储卷分布在多个可用区中。即使中断影响数据库实例或关联的 AZ，您的数据也仍可用。

Aurora Serverless v1 和快照

Aurora Serverless v1 集群的集群卷始终是加密的。您可以选择加密密钥，但不能禁用加密。要复制或分享 Aurora Serverless v1 集群的快照，可使用您自己的 AWS KMS key 对快照加密。有关更多信息，请参阅 数据库集群快照复制。要了解有关加密和 Amazon Aurora 的更多信息，请参阅加密 Amazon Aurora 数据库集群。

创建 Aurora Serverless v1 数据库集群

> **Important**
>
> AWS 已宣布 Aurora Serverless v1 的生命周期终止日期为：2025 年 3 月 31 日。所有在 2025 年 3 月 31 日之前未迁移的 Aurora Serverless v1 集群都将在维护时段内迁移到 Aurora Serverless v2。如果升级失败，Amazon Aurora 会在维护时段内将 Serverless v1 集群转换为具有等效引擎版本的预置集群。如果适用，Amazon Aurora 将在 Amazon RDS 扩展支持中注册转换后的预置集群。有关更多信息，请参阅 RDS Extended Support。

以下程序创建没有任何架构对象或数据的 Aurora Serverless v1 集群。如果想要创建一个 Aurora Serverless v1 集群，它是现有预置集群或 Aurora Serverless v1 集群的副本，您可以改为执行快照还原或克隆操作。有关这些详细信息，请参阅从数据库集群快照还原和克隆 Amazon Aurora 数据库集群卷。您无法将现有预置集群转换为 Aurora Serverless v1。您也无法将现有 Aurora Serverless v1 集群转换回预置集群。

在创建 Aurora Serverless v1 数据库集群时，您可以设置集群的最小容量和最大容量。容量单位等效于特定的计算和内存配置。Aurora Serverless v1 创建 CPU 使用率、连接和可用内存阈值的扩缩规则，并根据应用程序的需求无缝地扩缩到一系列容量单位。有关更多信息，请参阅 Aurora Serverless v1 架构。

您可以为 Aurora Serverless v1 数据库集群设置以下特定值：

- Minimum Aurora capacity unit (最小 Aurora 容量单元) – Aurora Serverless v1 可以将容量减少到该容量单元。
- Maximum Aurora capacity unit (最大 Aurora 容量单元) – Aurora Serverless v1 可以将容量增加到该容量单元。

您还可以选择以下可选的扩展配置选项：

- 超时后强制将容量扩展至指定值 – 如果您希望 Aurora Serverless v1 即使在超时之前找不到扩展点也强制扩展 Aurora Serverless v1，则可以选择此设置。如果您想要 Aurora Serverless v1 在找不到扩展点的情况下取消容量更改，请不要选择此设置。有关更多信息，请参阅“容量更改超时操作”。
- 连续几分钟不活动后暂停计算容量 – 如果您希望 Aurora Serverless v1 在数据库集群没有活动达到指定的时间时缩减到零容量，则可以选择此设置。启用此设置后，Aurora Serverless v1 数据库集群会在数据库流量恢复时自动恢复处理并扩展到处理工作负载所需的容量。要了解更多信息，请参阅“暂停和恢复 Aurora Serverless v1”。

您必须有 Aurora Serverless v1 账户，才能创建 AWS 数据库集群。您还需要完成使用 Amazon Aurora 的设置任务。有关更多信息，请参阅“为 Amazon Aurora 设置环境”。您还需要完成创建任何 Aurora 数据库集群的其他准备步骤。要了解更多信息，请参阅 创建 Amazon Aurora 数据库集群。

Aurora Serverless v1 在某些 AWS 区域可用，且仅适用于特定 Aurora MySQL 和 Aurora PostgreSQL 版本。有关更多信息，请参阅 Aurora Serverless v1。

> **Note**
>
> Aurora Serverless v1 集群的集群卷始终是加密的。创建 Aurora Serverless v1 数据库集群时，无法关闭加密，但您可以选择使用自己的加密密钥。使用 Aurora Serverless v2 时，您可以选择是否加密集群卷。

您可以使用 AWS CLI 或 RDS API 创建 Aurora Serverless v1 数据库集群。

> **Note**
>
> 如果您在尝试创建集群时收到以下错误消息，则表示您的账户需要其他权限。
> `Unable to create the resource. Verify that you have permission to create service linked role. Otherwise wait and try again later.`
> 参阅 将服务相关角色用于 Amazon Aurora 了解更多信息。

您无法直接连接到 Aurora Serverless v1 数据库集群上的数据库实例。要连接到 Aurora Serverless v1 数据库集群，请使用数据库终端节点。您可以在 Aurora Serverless v1 中的集群连接和安全性选项卡中找到 AWS Management Console 数据库集群的终端节点 。有关更多信息，请参阅“连接到 Amazon Aurora 数据库集群”。

AWS CLI

要使用 Aurora Serverless v1 创建新的 AWS CLI 数据库集群，请运行 create-db-cluster 命令并为 `serverless` 选项指定 `--engine-mode`。

您可以选择指定 `--scaling-configuration` 选项来配置最小容量、最大容量以及无连接时的自动暂停。

以下命令示例通过将 `--engine-mode` 选项设置为 `serverless` 来创建新的 Serverless 数据库集群。该示例还指定 `--scaling-configuration` 选项的值。

Aurora MySQL 的示例

以下命令创建一个新的与 Aurora MySQL 兼容的无服务器数据库集群。Aurora MySQL 的有效容量值为 1、2、4、8、16、32、64、128 和 256。

对于 Linux、macOS 或 Unix：

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster \
    --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.4 \
    --engine-mode serverless \
    --scaling-configuration
 MinCapacity=4,MaxCapacity=32,SecondsUntilAutoPause=1000,AutoPause=true \
    --master-username username --master-user-password password
```

对于 Windows：

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster ^
    --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.4 ^
    --engine-mode serverless ^
    --scaling-configuration
 MinCapacity=4,MaxCapacity=32,SecondsUntilAutoPause=1000,AutoPause=true ^
    --master-username username --master-user-password password
```

Aurora PostgreSQL 的示例

以下命令创建一个新的与 PostgreSQL 13.9 兼容的无服务器数据库集群。Aurora PostgreSQL 的有效容量值为 2、4、8、16、32、64、192 和 384。

对于 Linux、macOS 或 Unix：

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster \
```

```
    --engine aurora-postgresql --engine-version 13.9 \
    --engine-mode serverless \
    --scaling-configuration
 MinCapacity=8,MaxCapacity=64,SecondsUntilAutoPause=1000,AutoPause=true \
    --master-username username --master-user-password password
```

对于 Windows：

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster ^
    --engine aurora-postgresql --engine-version 13.9 ^
    --engine-mode serverless ^
    --scaling-configuration
 MinCapacity=8,MaxCapacity=64,SecondsUntilAutoPause=1000,AutoPause=true ^
    --master-username username --master-user-password password
```

RDS API

要使用 RDS API 创建新的 Aurora Serverless v1 数据库集群，请运行 CreateDBCluster 操作并为 `serverless` 参数指定 `EngineMode`。

您可以选择指定 `ScalingConfiguration` 参数来配置最小容量、最大容量以及无连接时的自动暂停。有效的容量值包括：

- Aurora MySQL：1、2、4、8、16、32、64、128 和 256。
- Aurora PostgreSQL：2、4、8、16、32、64、192 和 384。

还原 Aurora Serverless v1 数据库集群

> **Important**
>
> AWS 已宣布 Aurora Serverless v1 的生命周期终止日期为：2025 年 3 月 31 日。所有在 2025 年 3 月 31 日之前未迁移的 Aurora Serverless v1 集群都将在维护时段内迁移到 Aurora Serverless v2。如果升级失败，Amazon Aurora 会在维护时段内将 Serverless v1 集群转换为具有等效引擎版本的预置集群。如果适用，Amazon Aurora 将在 Amazon RDS 扩展支持中注册转换后的预置集群。有关更多信息，请参阅 RDS Extended Support。

当您使用 AWS CLI 或 RDS API 还原预调配数据库集群快照时，可以配置 Aurora Serverless v1 数据库集群。

在将快照还原到 Aurora Serverless v1 数据库集群时，您可以设置以下特定值：

- Minimum Aurora capacity unit (最小 Aurora 容量单元) – Aurora Serverless v1 可以将容量减少到该容量单元。
- Maximum Aurora capacity unit (最大 Aurora 容量单元) – Aurora Serverless v1 可以将容量增加到该容量单元。
- 超时操作 – 当容量修改操作因找不到扩展点而超时时要执行的操作。Aurora Serverless v1如果设置了 Force scaling the capacity to the specified values... (强制将容量扩展至指定值…) 选项，数据库集群就可以强制数据库集群使用新的容量设置。或者，如果您不选择该选项，它将可以回滚容量更改以取消它。有关更多信息，请参阅“容量更改超时操作”。
- 不活动后暂停 – 从无数据库流量到缩减到零处理容量之间的时间长度。当数据库流量恢复后，Aurora 将自动恢复处理容量并进行扩展以处理流量。

有关从快照还原数据库集群的一般信息，请参阅从数据库集群快照还原。

AWS CLI

您可以在使用 AWS Management Console、AWS CLI 或 RDS API 还原预置数据库集群快照时配置 Aurora Serverless 数据库集群。

在将快照还原到 Aurora Serverless 数据库集群时，您可以设置以下特定值：

- Minimum Aurora capacity unit (最小 Aurora 容量单元) – Aurora Serverless 可以将容量减少到该容量单元。
- Maximum Aurora capacity unit (最大 Aurora 容量单元) – Aurora Serverless 可以将容量增加到该容量单元。
- 超时操作 – 当容量修改操作因找不到扩展点而超时时要执行的操作。Aurora Serverless v1如果设置了 Force scaling the capacity to the specified values... (强制将容量扩展至指定值…) 选项，数据库集群就可以强制数据库集群使用新的容量设置。或者，如果您不选择该选项，它将可以回滚容量更改以取消它。有关更多信息，请参阅“容量更改超时操作”。
- 不活动后暂停 – 从无数据库流量到缩减到零处理容量之间的时间长度。当数据库流量恢复后，Aurora 将自动恢复处理容量并进行扩展以处理流量。

> **Note**
>
> 数据库集群快照的版本必须与 Aurora Serverless v1 兼容。有关所支持版本的列表，请参阅 Aurora Serverless v1。

要将快照还原到与 MySQL 5.7 兼容的 Aurora Serverless v1 集群，请包括以下附加参数：

- `--engine` **`aurora-mysql`**
- `--engine-version` **`5.7`**

`--engine` 和 `--engine-version` 参数允许您从与 MySQL 5.6 兼容的 Aurora 或 Aurora Serverless v1 快照，创建与 MySQL 5.7 兼容的 Aurora Serverless v1 集群。以下示例将快照从名为 *`mydbclustersnapshot`* 的与 MySQL 5.6 兼容的集群，还原到名为 *`mynewdbcluster`* 的与 MySQL 5.7 兼容的 Aurora Serverless v1 集群。

对于 Linux、macOS 或 Unix：

```
aws rds restore-db-cluster-from-snapshot \
    --db-cluster-identifier mynewdbcluster \
    --snapshot-identifier mydbclustersnapshot \
    --engine-mode serverless \
    --engine aurora-mysql \
    --engine-version 5.7
```

对于 Windows：

```
aws rds restore-db-cluster-from-snapshot ^
    --db-instance-identifier mynewdbcluster ^
    --db-snapshot-identifier mydbclustersnapshot ^
    --engine aurora-mysql ^
    --engine-version 5.7
```

您可以选择指定 `--scaling-configuration` 选项来配置最小容量、最大容量以及无连接时的自动暂停。有效的容量值包括：

- Aurora MySQL：1、2、4、8、16、32、64、128 和 256。
- Aurora PostgreSQL：2、4、8、16、32、64、192 和 384。

在以下示例中，您将从之前创建的名为 *mydbclustersnapshot* 的数据库集群，还原到名为 *mynewdbcluster* 的新数据库集群。您可以设置 `--scaling-configuration`，以便新 Aurora Serverless v1 数据库集群可以根据需要，从 8 ACU 扩展到 64 ACU（Aurora 容量单元），以处理工作负载。处理完成且没有支持的连接 1000 秒后，集群将关闭，直到连接请求提示它重新启动。

对于 Linux、macOS 或 Unix：

```
aws rds restore-db-cluster-from-snapshot \
    --db-cluster-identifier mynewdbcluster \
    --snapshot-identifier mydbclustersnapshot \
    --engine-mode serverless --scaling-configuration
 MinCapacity=8,MaxCapacity=64,TimeoutAction='ForceApplyCapacityChange',SecondsUntilAutoPause=10
```

对于 Windows：

```
aws rds restore-db-cluster-from-snapshot ^
    --db-instance-identifier mynewdbcluster ^
    --db-snapshot-identifier mydbclustersnapshot ^
    --engine-mode serverless --scaling-configuration
 MinCapacity=8,MaxCapacity=64,TimeoutAction='ForceApplyCapacityChange',SecondsUntilAutoPause=10
```

RDS API

要在使用 RDS API 从数据库集群中还原时配置 Aurora Serverless v1 数据库集群，请运行 RestoreDBClusterFromSnapshot 操作并为 `serverless` 参数指定 `EngineMode`。

您可以选择指定 `ScalingConfiguration` 参数来配置最小容量、最大容量以及无连接时的自动暂停。有效的容量值包括：

- Aurora MySQL：1、2、4、8、16、32、64、128 和 256。
- Aurora PostgreSQL：2、4、8、16、32、64、192 和 384。

修改 Aurora Serverless v1 数据库集群

Important

AWS 已宣布 Aurora Serverless v1 的生命周期终止日期为：2025 年 3 月 31 日。所有在 2025 年 3 月 31 日之前未迁移的 Aurora Serverless v1 集群都将在维护时段内迁移到 Aurora

Serverless v2。如果升级失败，Amazon Aurora 会在维护时段内将 Serverless v1 集群转换为具有等效引擎版本的预置集群。如果适用，Amazon Aurora 将在 Amazon RDS 扩展支持中注册转换后的预置集群。有关更多信息，请参阅 RDS Extended Support。

在配置 Aurora Serverless v1 数据库集群后，您可以使用 AWS Management Console、AWS CLI 或 RDS API 修改某些属性。您可以修改的大多数属性与其他类型的 Aurora 集群相同。

以下是 Aurora Serverless v1 的最相关的更改。

- 修改扩缩配置
- 升级主要版本
- 从 Aurora Serverless v1 转换为预置

修改 Aurora Serverless v1 数据库集群的扩缩配置。

可以设置数据库集群的最小容量和最大容量。每个容量单元等效于特定的计算和内存配置。Aurora Serverless 自动创建 CPU 使用率、连接和可用内存阈值的扩展规则。您还可以设置 Aurora Serverless 在无任何活动时是否暂停数据库，然后在活动再次开始时恢复。

您可以为扩展配置设置以下特定值：

- Minimum Aurora capacity unit (最小 Aurora 容量单元) – Aurora Serverless 可以将容量减少到该容量单元。
- Maximum Aurora capacity unit (最大 Aurora 容量单元) – Aurora Serverless 可以将容量增加到该容量单元。
- 自动扩展超时和操作 - 此部分指定 Aurora Serverless 在超时之前等待多长时间以查找扩展点。它也指定当容量修改操作因找不到扩展点而超时时要执行的操作。Aurora 可以强制执行容量更改，尽快将容量设置为指定值。也可以回滚容量更改以取消此更改。有关更多信息，请参阅 容量更改超时操作。
- 不活动后暂停 - 使用可选的集群空闲时将容量扩展到 0 ACU 设置，可在数据库处于非活动状态时将数据库的处理容量扩展到零。当数据库流量恢复后，Aurora 将自动恢复处理容量并进行扩展以处理流量。

> **Note**
>
> 当您修改 Aurora Serverless 数据库集群的容量范围时，无论您是选择立即应用更改，还是在下一个计划维护时段内应用更改，更改都会立即发生。

控制台

您可以使用AWS Management Console修改 Aurora 数据库集群的扩展配置。

修改 Aurora Serverless v1 数据库集群

1. 通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Databases (数据库)。
3. 选择要修改的 Aurora Serverless v1 数据库集群。
4. 对于操作，选择修改集群。
5. 在容量设置部分，修改扩展配置。
6. 选择继续。
7. 在修改数据库集群页面上，查看您所做的修改，然后选择何时应用这些修改。
8. 选择修改集群。

AWS CLI

要使用 Aurora Serverless v1 修改 AWS CLI 数据库集群的扩展配置，请运行 modify-db-cluster AWS CLI 命令。指定 `--scaling-configuration` 选项来配置最小容量、最大容量以及无连接时的自动暂停。有效的容量值包括：

- Aurora MySQL：1、2、4、8、16、32、64、128 和 256。
- Aurora PostgreSQL：2、4、8、16、32、64、192 和 384。

在该示例中，您修改一个名为 *`sample-cluster`* 的 Aurora Serverless v1 数据库集群的扩展配置。

对于 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster \
    --db-cluster-identifier sample-cluster \
```

```
    --scaling-configuration
 MinCapacity=8,MaxCapacity=64,SecondsUntilAutoPause=500,TimeoutAction='ForceApplyCapacityChange
```

对于 Windows：

```
aws rds modify-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --scaling-configuration
 MinCapacity=8,MaxCapacity=64,SecondsUntilAutoPause=500,TimeoutAction='ForceApplyCapacityChange
```

RDS API

您可以使用 ModifyDBCluster API 操作修改 Aurora 数据库集群的扩展配置。指定 `ScalingConfiguration` 参数来配置最小容量、最大容量以及无连接时的自动暂停。有效的容量值包括：

- Aurora MySQL：1、2、4、8、16、32、64、128 和 256。
- Aurora PostgreSQL：2、4、8、16、32、64、192 和 384。

升级 Aurora Serverless v1 数据库集群的主要版本

> **Important**
>
> AWS 已宣布 Aurora Serverless v1 的生命周期终止日期为：2025 年 3 月 31 日。所有在 2025 年 3 月 31 日之前未迁移的 Aurora Serverless v1 集群都将在维护时段内迁移到 Aurora Serverless v2。如果升级失败，Amazon Aurora 会在维护时段内将 Serverless v1 集群转换为具有等效引擎版本的预置集群。如果适用，Amazon Aurora 将在 Amazon RDS 扩展支持中注册转换后的预置集群。有关更多信息，请参阅 RDS Extended Support。

您可以将与 PostgreSQL 11 兼容的 Aurora Serverless v1 数据库集群的主要版本升级到相应的 PostgreSQL 13 兼容版本。

控制台

您可以使用 AWS Management Console执行 Aurora Serverless v1 数据库集群的就地升级。

升级 Aurora Serverless v1 数据库集群

1. 通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。

2. 在导航窗格中，选择 Databases (数据库)。
3. 选择要升级的 Aurora Serverless v1 数据库集群。
4. 对于操作，选择修改集群。
5. 对于版本，请选择 Aurora PostgreSQL 版本 13 版本号。

   以下示例显示了从 Aurora PostgreSQL 11.16 到 13.9 的就地升级。



   如果您执行主要版本升级，请保持所有其他属性相同。要更改任何其他属性，请在升级完成后执行另一个修改操作。

6. 选择继续。
7. 在修改数据库集群页面上，查看您所做的修改，然后选择何时应用这些修改。
8. 选择修改集群。

AWS CLI

要执行从 PostgreSQL 11 兼容的 Aurora Serverless v1 数据库集群到 PostgreSQL 13 兼容的数据库集群的就地升级，请使用与 Aurora Serverless v1 兼容的 Aurora PostgreSQL 版本 13 版本号指定 `--engine-version` 参数。还包含 `--allow-major-version-upgrade` 参数。

在本例中，您是修改与 PostgreSQL 11 兼容的 Aurora Serverless v1 数据库集群（名为 `sample-cluster`）的主要版本。这样做可以就地升级到 PostgreSQL 13 兼容的 Aurora Serverless v1 数据库集群。

```
aws rds modify-db-cluster \
    --db-cluster-identifier sample-cluster \
    --engine-version 13.serverless_12 \
    --allow-major-version-upgrade
```

对于 Windows：

```
aws rds modify-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --engine-version 13.serverless_12 ^
    --allow-major-version-upgrade
```

RDS API

要执行从 PostgreSQL 11 兼容的 Aurora Serverless v1 数据库集群到 PostgreSQL 13 兼容的数据库集群的就地升级，请使用与 Aurora Serverless v1 兼容的 Aurora PostgreSQL 版本 13 版本号指定 `EngineVersion` 参数。还包含 `AllowMajorVersionUpgrade` 参数。

将 Aurora Serverless v1 数据库集群转换为预调配

可以将 Aurora Serverless v1 数据库集群转换为预调配的数据库集群。要执行转换，请使用 AWS CLI 或 Amazon RDS API 将数据库实例类更改为预置。使用下面的步骤修改数据库实例类。

以下示例演示如何使用 AWS CLI 将 Aurora Serverless v1 数据库集群转换为预置集群。

AWS CLI

要将 Aurora Serverless v1 数据库集群转换为预调配集群，请运行 modify-db-cluster AWS CLI 命令。

以下参数为必需参数：

- `--db-cluster-identifier` – 您要转换为预调配的 Aurora Serverless v1 数据库集群。
- `--engine-mode` – 使用值 `provisioned`。
- `--allow-engine-mode-change`
- `--db-cluster-instance-class` – 根据 Aurora Serverless v1 数据库集群的容量为预调配的数据库集群选择数据库实例类。

在此示例中，您将转换名为 `sample-cluster` 的 Aurora Serverless v1 数据库集群并使用 `db.r5.xlarge` 数据库实例类。

对于 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster \
--db-cluster-identifier sample-cluster \
--engine-mode provisioned \
--allow-engine-mode-change \
--db-cluster-instance-class db.r5.xlarge
```

对于 Windows：

```
aws rds modify-db-cluster ^
--db-cluster-identifier sample-cluster ^
--engine-mode provisioned ^
--allow-engine-mode-change ^
--db-cluster-instance-class db.r5.xlarge
```

以下示例演示如何使用 Amazon RDS API 将 Aurora Serverless v1 数据库集群转换为预置集群。

RDS API

要将 Aurora Serverless v1 数据库集群转换为预调配的集群，请使用 ModifyDBCluster API 操作。

以下参数为必需参数：

- `DBClusterIdentifier` – 您要转换为预调配的 Aurora Serverless v1 数据库集群。
- `EngineMode` – 使用值 `provisioned`。
- `AllowEngineModeChange`
- `DBClusterInstanceClass` – 根据 Aurora Serverless v1 数据库集群的容量为预调配的数据库集群选择数据库实例类。

从 Aurora Serverless v1 数据库集群转换为预置集群时的注意事项

将 Aurora Serverless v1 数据库集群转换为预置集群时，以下注意事项适用：

- 您可以将此转换用作将数据库集群从 Aurora Serverless v1 升级到 Aurora Serverless v2 的一部分。有关更多信息，请参阅 从 Aurora Serverless v1 集群升级到 Aurora Serverless v2。
- 转换过程在数据库集群中创建读取器数据库集群，将读取器实例提升为写入器实例，然后删除原始 Aurora Serverless v1 实例。转换数据库集群时，无法同时执行任何其他修改，例如更改数据库引擎版本或数据库集群参数组。转换操作将立即应用，无法撤消。
- 在转换过程中，会拍摄数据库集群的备份数据库集群快照，以防出现错误。数据库集群快照的标识符格式为 `pre-modify-engine-mode-`*`DB_cluster_identifier`*`-`*`timestamp`*。
- Aurora 为预调配的数据库集群使用当前默认的数据库次要引擎版本。
- 如果您没有为转换后的数据库集群提供数据库实例类，Aurora 会根据原始 Aurora Serverless v1 数据库集群的最大容量推荐一个数据库实例类。下表中显示了推荐的容量到实例类映射。

Serverless 最大容量（ACU）	预调配的数据库实例类
1	db.t3.small
2	db.t3.medium
4	db.t3.large
8	db.r5.large
16	db.r5.xlarge
32	db.r5.2xlarge
64	db.r5.4xlarge
128	db.r5.8xlarge
192	db.r5.12xlarge
256	db.r5.16xlarge
384	db.r5.24xlarge

> **Note**
>
> 根据您选择的数据库实例类和数据库使用情况，您可能看到预调配数据库集群的成本与 Aurora Serverless v1 相比不同。
> 如果您将 Aurora Serverless v1 数据库集群转换为可突增（db.t* ）数据库实例类，则使用数据库集群可能会产生额外费用。有关更多信息，请参阅 数据库实例类类型。

手动扩展 Aurora Serverless v1 数据库集群容量

> **Important**
>
> AWS 已宣布 Aurora Serverless v1 的生命周期终止日期为：2025 年 3 月 31 日。所有在 2025 年 3 月 31 日之前未迁移的 Aurora Serverless v1 集群都将在维护时段内迁移到 Aurora Serverless v2。如果升级失败，Amazon Aurora 会在维护时段内将 Serverless v1 集群转换为具有等效引擎版本的预置集群。如果适用，Amazon Aurora 将在 Amazon RDS 扩展支持中注册转换后的预置集群。有关更多信息，请参阅 RDS Extended Support。

通常，Aurora Serverless v1 数据库集群可根据工作负载无缝扩展。但是，容量的扩展速度可能并不总是足以应对突然的极端情况，例如事务的指数级增长。在这种情况下，您可以通过设置新的容量值手动启动扩展操作。在显式设置容量后，Aurora Serverless v1 会自动扩展数据库集群。它会根据纵向缩减的冷却时间执行此操作。

您可以使用 AWS Management Console、AWS CLI 或 RDS API 将 Aurora Serverless v1 数据库集群的容量显式设置为特定的值。

控制台

您可以使用AWS Management Console设置 Aurora 数据库集群的容量。

修改 Aurora Serverless v1 数据库集群

1. 通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Databases (数据库)。
3. 选择要修改的 Aurora Serverless v1 数据库集群。
4. 有关 Actions (操作)，选择 Set capacity (设置容量)。

5. 在扩展数据库容量窗口中，选择以下选项：

   a. 针对将数据库集群扩展到下拉选择器，选择您希望为数据库集群使用的新容量。

   b. 针对如果找不到无缝扩缩点复选框，为 Aurora Serverless v1 数据库集群的 `TimeoutAction` 设置选择所需的行为，如下所示：

      - 如果您希望 Aurora Serverless v1 在超时之前找不到扩展点的情况下容量依然保持不变，请清除此选项。
      - 如果您希望强制 Aurora Serverless v1 数据库集群更改容量（即使其在超时之前找不到扩展点），请选择此选项。此选项可能会导致 Aurora Serverless v1 中断连接，使其无法找到扩展点。

   c. 在秒字段中，输入您希望允许 Aurora Serverless v1 数据库集群在超时之前查找扩展点的时间量。您可以指定从 10 秒到 600 秒（10 分钟）的任意时间量。默认值为 5 分钟（300 秒）。以下示例强制 Aurora Serverless v1 数据库集群向下扩展到 2 ACU，即使它在五分钟内找不到扩展点也是如此。



6. 选择 Apply。

要了解有关扩展点、`TimeoutAction` 和冷却时间的更多信息，请参阅 Aurora Serverless v1 的自动扩展。

AWS CLI

要使用 Aurora Serverless v1 设置 AWS CLI 数据库集群的容量，请运行 modify-current-db-cluster-capacity AWS CLI 命令并指定 `--capacity` 选项。有效的容量值包括：

- Aurora MySQL：1、2、4、8、16、32、64、128 和 256。
- Aurora PostgreSQL：2、4、8、16、32、64、192 和 384。

在该示例中，您将一个名为 *`sample-cluster`* 的 Aurora Serverless v1 数据库集群的容量设置为 *`64`*。

```
aws rds modify-current-db-cluster-capacity --db-cluster-identifier sample-cluster --
capacity 64
```

RDS API

您可以使用 ModifyCurrentDBClusterCapacity API 操作设置 Aurora 数据库集群的容量。指定 `Capacity` 参数。有效的容量值包括：

- Aurora MySQL：1、2、4、8、16、32、64、128 和 256。
- Aurora PostgreSQL：2、4、8、16、32、64、192 和 384。

查看 Aurora Serverless v1 数据库集群

> **Important**
>
> AWS 已宣布 Aurora Serverless v1 的生命周期终止日期为：2025 年 3 月 31 日。所有在 2025 年 3 月 31 日之前未迁移的 Aurora Serverless v1 集群都将在维护时段内迁移到 Aurora Serverless v2。如果升级失败，Amazon Aurora 会在维护时段内将 Serverless v1 集群转换为具有等效引擎版本的预置集群。如果适用，Amazon Aurora 将在 Amazon RDS 扩展支持中注册转换后的预置集群。有关更多信息，请参阅 RDS Extended Support。

在创建一个或多个 Aurora Serverless v1 数据库集群后，您可以查看无服务器和实例类型的数据库集群。您还可以查看每个 Aurora Serverless v1 数据库集群正在使用的 Aurora 容量单元（ACU）的当前数量。每个 ACU 是处理器（CPU）和内存（RAM）容量的组合。

查看 Aurora Serverless v1 数据库集群

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在 AWS Management Console 的右上角，选择在其中创建了 Aurora Serverless v1 数据库集群的 AWS 区域。
3. 在导航窗格中，选择 Databases (数据库)。

   对于每个数据库集群，数据库集群类型显示在 Role (角色) 下方。Aurora Serverless v1 数据库集群显示无服务器作为类型。您可以在大小下查看 Aurora Serverless v1 数据库集群的当前容量。



4. 选择 Aurora Serverless v1 数据库集群的名称以显示其详细信息。

   在连接和安全性选项卡上，记下数据库终端节点。使用此终端节点连接到 Aurora Serverless v1 数据库集群。



选择配置选项卡以查看容量设置。



扩展事件 在数据库集群扩展、缩减、暂停或恢复时生成。选择 Logs & events (日志和事件) 选项卡可查看近期事件。下图显示了这些事件的示例。



监控 Aurora Serverless v1 数据库集群的容量和扩展事件

您可以在 CloudWatch 中查看 Aurora Serverless v1 数据库集群，以使用 `ServerlessDatabaseCapacity` 指标监控分配给数据库集群的容量。此外，您还可以监控所有标准 Aurora CloudWatch 指标，比如 `CPUUtilization`、`DatabaseConnections`、`Queries` 等。

您可使用 Aurora 将某些或所有数据库日志发布到 CloudWatch。通过在与 集群关联的数据库集群参数组中启用 general_log 和 slow_query_log 等配置参数Aurora Serverless v1，可选择要发布的日志。与预置的集群不同，Aurora Serverless v1 集群无需您在数据库集群设置中指定要将哪些日志类型上传到 CloudWatch。Aurora Serverless v1 集群会自动上传所有可用的日志。在禁用日志配置参数时，将停止向 CloudWatch 发布日志。如果不再需要使用 CloudWatch 中的日志，您也可以删除这些日志。

要开始为您的 Aurora Serverless v1 数据库集群使用 Amazon CloudWatch，请参见通过 Amazon CloudWatch 查看 Aurora Serverless v1 日志。要详细了解如何通过 CloudWatch 监控 Aurora 数据库集群 ，请参阅 在 Amazon CloudWatch 中监控日志事件。

要连接到 Aurora Serverless v1 数据库集群，请使用数据库终端节点。有关更多信息，请参阅“连接到 Amazon Aurora 数据库集群”。

> **Note**
>
> 您无法直接连接到 Aurora Serverless v1 数据库集群中的特定数据库实例。

删除 Aurora Serverless v1 数据库集群

> **Important**
>
> AWS 已宣布 Aurora Serverless v1 的生命周期终止日期为：2025 年 3 月 31 日。所有在 2025 年 3 月 31 日之前未迁移的 Aurora Serverless v1 集群都将在维护时段内迁移到 Aurora Serverless v2。如果升级失败，Amazon Aurora 会在维护时段内将 Serverless v1 集群转换为具有等效引擎版本的预置集群。如果适用，Amazon Aurora 将在 Amazon RDS 扩展支持中注册转换后的预置集群。有关更多信息，请参阅 RDS Extended Support。

根据您创建 Aurora Serverless v1 数据库集群的方式，默认情况下可能会开启删除保护。您不能立即删除启用了删除保护的 Aurora Serverless v1 数据库集群。要使用 Aurora Serverless v1 删除具有删除保护的 AWS Management Console 数据库集群，首先需要修改集群以删除此保护。有关针对该任务使用 AWS CLI 的信息，请参阅 AWS CLI。

使用 AWS Management Console 禁用删除保护

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。

2. 在导航窗格中，选择数据库集群。
3. 从列表中选择 Aurora Serverless v1 数据库集群。
4. 选择修改以打开数据库集群的配置。“修改数据库集群”页面将打开 Aurora Serverless v1 数据库集群的设置、容量设置和其他配置详细信息。删除保护位于 Additional configuration（其他配置）部分。
5. 清除 Additional configuration（其他配置）属性卡中的 Enable deletion protection（启用删除保护）复选框。
6. 选择 Continue (继续)。此时将显示修改摘要。
7. 选择修改集群以接受修改摘要。您还可以选择上一步以修改更改，或选择取消以放弃更改。

待删除保护不再处于活动状态后，您可以使用AWS Management Console删除 Aurora Serverless v1 数据库集群。

控制台

删除 Aurora Serverless v1 数据库集群

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在资源部分，选择数据库集群。
3. 选择要删除的 Aurora Serverless v1 数据库集群。
4. 对于 Actions，选择 Delete。系统会提示您确认是否要删除 Aurora Serverless v1 数据库集群。
5. 建议您保留预先选择的选项：
   - 对是否创建最终快照？选择是
   - 您的 Aurora Serverless v1 数据库集群名称加上 `-final-snapshot`，即为最终快照名称。但是，您可以在此字段中更改最终快照的名称。



如果针对是否创建最终快照？选择否，则您无法使用快照或时间点恢复来还原数据库集群。

6. 选择删除数据库集群。

Aurora Serverless v1 会删除您的数据库集群。如果您选择保留最终快照，则在删除 Aurora Serverless v1 数据库集群之前，您会看到其状态更改为“正在备份”且不再显示在列表中。

AWS CLI

开始之前，请使用 AWS CLI 访问密钥 ID、AWS 秘密访问密钥和您的 AWS 数据库集群所在的 AWS 区域来配置 Aurora Serverless v1。有关更多信息，请参阅 AWS Command Line Interface 用户指南中的配置基础知识。

只有在首次禁用使用此选项配置的集群删除保护后，您才能删除 Aurora Serverless v1 数据库集群。如果您尝试删除启用了此保护选项的集群，则会看到以下错误消息。

```
An error occurred (InvalidParameterCombination) when calling the DeleteDBCluster
  operation: Cannot delete protected Cluster, please disable deletion protection and
 try again.
```

您可以使用 modify-db-cluster Aurora Serverless v1 命令更改 AWS CLI 数据库集群的删除保护设置，如下所示：

```
aws rds modify-db-cluster --db-cluster-identifier your-cluster-name --no-deletion-
protection
```

此命令返回指定数据库集群的修订属性。您现在可以删除您的 Aurora Serverless v1 数据库集群。

建议您在删除 Aurora Serverless v1 数据库集群时始终创建最终快照。以下使用 AWS CLI delete-db-cluster 命令的示例向您展示了创建快照的操作方法。您可以提供数据库集群的名称和快照的名称。

对于 Linux、macOS 或 Unix：

```
aws rds delete-db-cluster --db-cluster-identifier \
  your-cluster-name --no-skip-final-snapshot \
  --final-db-snapshot-identifier name-your-snapshot
```

对于 Windows：

```
aws rds delete-db-cluster --db-cluster-identifier ^
  your-cluster-name --no-skip-final-snapshot ^
  --final-db-snapshot-identifier name-your-snapshot
```

Aurora Serverless v1 和 Aurora 数据库引擎版本

> **Important**
>
> AWS 已宣布 Aurora Serverless v1 的生命周期终止日期为：2025 年 3 月 31 日。所有在 2025 年 3 月 31 日之前未迁移的 Aurora Serverless v1 集群都将在维护时段内迁移到 Aurora Serverless v2。如果升级失败，Amazon Aurora 会在维护时段内将 Serverless v1 集群转换为具有等效引擎版本的预置集群。如果适用，Amazon Aurora 将在 Amazon RDS 扩展支持中注册转换后的预置集群。有关更多信息，请参阅 RDS Extended Support。

Aurora Serverless v1 在某些 AWS 区域可用，且仅适用于特定 Aurora MySQL 和 Aurora PostgreSQL 版本。有关支持 Aurora Serverless v1 的 AWS 区域的当前列表，以及每个区域中可用的特定 Aurora MySQL 和 Aurora PostgreSQL 版本，请参阅 Aurora Serverless v1。

Aurora Serverless v1 使用其关联 Aurora 数据库引擎来标识每个受支持的数据库引擎的特定受支持版本，如下所示：

- Aurora MySQL Serverless

- Aurora PostgreSQL Serverless

当数据库引擎的次要版本对 Aurora Serverless v1 可用时，它们将自动应用于 Aurora Serverless v1 可用的各个 AWS 区域。换句话说，在集群的数据库引擎对 Aurora Serverless v1 可用时，您无需升级 Aurora Serverless v1 数据库集群即可获得新的次要版本。

Aurora MySQL Serverless

Aurora Serverless v1 在某些 AWS 区域中可用，且仅适用于特定 Aurora MySQL 版本。有关支持 Aurora Serverless v1 的 AWS 区域的当前列表，以及每个区域中可用的特定 Aurora MySQL 版本，请参阅适用于 Aurora MySQL 的 Aurora Serverless v1。

要了解 Aurora MySQL 版本 2 的增强功能和错误修复，请参阅《Aurora MySQL 发布说明》中的 Amazon Aurora MySQL 版本 2 的数据库引擎更新。

要使用更高版本的 Aurora MySQL，可以使用 Aurora Serverless v2。有关可与 Aurora Serverless v2 结合使用的 AWS 区域和 Aurora MySQL 版本，请参阅适用于 Aurora MySQL 的 Aurora Serverless v2。有关 Aurora Serverless v2 的使用信息，请参阅使用 Aurora Serverless v2。

Aurora PostgreSQL Serverless

Aurora Serverless v1 在某些 AWS 区域中可用，且仅适用于特定 Aurora PostgreSQL 版本。有关支持 Aurora Serverless v1 的 AWS 区域的当前列表，以及每个区域中可用的特定 Aurora PostgreSQL 版本，请参阅适用于 Aurora PostgreSQL 的 Aurora Serverless v1。

如果您要对 Aurora Serverless v1 数据库集群使用 Aurora PostgreSQL，则可以使用 Aurora PostgreSQL 13 兼容版本。Aurora PostgreSQL 兼容版的次要版本仅包括向后兼容的更改。当 Aurora PostgreSQL 次要版本在您的 AWS 区域中变得对 Aurora Serverless v1 可用时，您的 Aurora Serverless v1 数据库集群将以透明方式升级。

要使用更高版本的 Aurora PostgreSQL，可以使用 Aurora Serverless v2。有关可与 Aurora Serverless v2 结合使用的 AWS 区域和 Aurora PostgreSQL 版本，请参阅适用于 Aurora PostgreSQL 的 Aurora Serverless v2。有关 Aurora Serverless v2 的使用信息，请参阅使用 Aurora Serverless v2。

Aurora Serverless v1 的自动次要版本升级

当数据库引擎的次要版本对 Aurora Serverless v1 可用时，它们将自动应用于 Aurora Serverless v1 可用的各个 AWS 区域。换句话说，在集群的数据库引擎对 Aurora Serverless v1 可用时，您无需升级 Aurora Serverless v1 数据库集群即可获得新的次要版本。

使用 Amazon RDS 数据 API

通过 RDS 数据 API（数据 API），您可以将 Web 服务接口用于 Aurora 数据库集群。数据 API 不需要与数据库集群的持久连接。相反，它提供了安全 HTTP 端点以及与 AWS 开发工具包的集成。您可以使用端点运行 SQL 语句，而无需管理连接。

用户无需在调用数据 API 时传递凭证，因为数据 API 会使用存储在 AWS Secrets Manager 中的数据库凭证。要在 Secrets Manager 中存储凭证，用户必须获得使用 Secrets Manager 以及数据 API 的适当权限。有关向用户授权的更多信息，请参阅 授予对 Amazon RDS 数据 API 的访问权限。

您还可以使用数据 API 将 Amazon Aurora 与其他 AWS 应用程序（如 AWS Lambda、AWS AppSync 和 AWS Cloud9）集成。数据 API 提供了一种更安全的方式来使用 AWS Lambda。通过此方式，您无需配置 Lambda 函数来访问虚拟私有云（VPC）中的资源，即可访问数据库集群。有关更多信息，请参阅 AWS Lambda、AWS AppSync 和 AWS Cloud9。

您可以在创建 Aurora 数据库集群时启用数据 API。您也可以稍后修改配置。有关更多信息，请参阅 启用 Amazon RDS 数据 API。

启用数据 API 后，还可以使用查询编辑器运行即席查询，而无需配置查询工具在 VPC 中访问 Aurora。有关更多信息，请参阅 使用 Aurora 查询编辑器。

主题

- Amazon RDS 数据 API 的区域和版本可用性
- Amazon RDS 数据 API 的限制
- Aurora Serverless v2、预置集群与 Aurora Serverless v1 集群的 Amazon RDS 数据 API 行为比较
- 授予对 Amazon RDS 数据 API 的访问权限
- 启用 Amazon RDS 数据 API
- 为 Amazon RDS 数据 API 创建 Amazon VPC 端点（AWS PrivateLink）
- 调用 Amazon RDS 数据 API
- 使用适用于 RDS 数据 API 的 Java 客户端库
- 处理 JSON 格式的 Amazon RDS 数据 API 查询结果
- Amazon RDS 数据 API 故障排除
- 使用 AWS CloudTrail 记录 Amazon RDS 数据 API 调用
- 使用性能详情监控 RDS 数据 API 查询

Amazon RDS 数据 API 的区域和版本可用性

有关适用于数据 API 的区域和引擎版本的信息，请参阅以下部分。

集群类型	区域和版本可用性	
Aurora PostgreSQL 预调配和 Serverless v2	适用于 Aurora PostgreSQL Serverless v2 和预调配的数据 API	
Aurora MySQL 预调配和 Serverless v2	???	
Aurora PostgreSQL Serverless v1	适用于 Aurora PostgreSQL Serverless v1 的数据 API	
Aurora MySQL Serverless v1	适用于 Aurora MySQL Serverless v1 的数据 API	

如果在通过命令行界面或 API 访问数据 API 时需要经过 FIPS 140-2 验证的加密模块，请使用 FIPS 端点。有关可用的 FIPS 端点的更多信息，请参阅《美国联邦信息处理标准（FIPS）第 140-2 版》。

Amazon RDS 数据 API 的限制

RDS 数据 API 存在以下限制：

- 您只能对数据库集群中的写入器实例执行数据 API 查询。但是，写入器实例可以接受写入和读取查询。
- 使用 Aurora Global Database，您可以在主数据库集群和辅助数据库集群上均启用数据 API。但是，辅助集群在提升为主集群之前，没有写入器实例。数据 API 需要访问写入器实例才能进行查询处理，即使是读取查询也是如此。因此，当辅助集群缺少写入器实例时，发送到辅助集群的读取和写入查询会失败。在辅助集群得到提升并具有可用的写入器实例后，对该数据库实例的数据 API 查询会成功。
- T 数据库实例类不支持数据 API。
- 对于 Aurora Serverless v2 和预调配数据库集群，RDS 数据 API 不支持某些数据类型。有关受支持类型的列表，请参阅the section called “Aurora Serverless v2 与 Aurora Serverless v1 的数据 API 比较”。

- 对于 Aurora PostgreSQL 版本 14 和更高版本的数据库，数据 API 仅支持使用 `scram-sha-256` 进行密码加密。
- 响应大小限制为 1MiB。如果调用返回的响应数据超过 1MiB，则调用将终止。
- 对于 Aurora Serverless v1，每秒最大请求数为 1000。对于所有其它受支持的数据库，没有限制。
- 数据库返回的结果集中的 Data API 大小限制为每行 64 KB。确保结果集中的每一行都小于或等于 64 KB。

Aurora Serverless v2、预置集群与 Aurora Serverless v1 集群的 Amazon RDS 数据 API 行为比较

Amazon RDS 数据 API 的最新增强功能，使得数据 API 可用于使用最新版本 PostgreSQL 或 MySQL 引擎的集群。这些集群可以配置为使用 Aurora Serverless v2 或预置实例类，例如 `db.r6g` 或 `db.r6i`。

以下部分描述了在 Aurora Serverless v2、预置数据库集群和 Aurora Serverless v1 数据库集群上 Amazon RDS 数据 API 的区别。Aurora Serverless v1数据库集群使用 `serverless` 引擎模式。预置数据库集群使用 `provisioned` 引擎模式。Aurora Serverless v2 数据库集群还使用 `provisioned` 引擎模式，并包含一个或多个使用 `db.serverless` 实例类的 Aurora Serverless v2 数据库实例。

每秒最大请求数

Aurora Serverless v1

数据 API 每秒最多可以发出 1000 个请求。

Aurora Serverless v2

数据 API 每秒可以发出的请求数量没有限制。

在现有数据库上启用或禁用 Amazon RDS 数据 API

Aurora Serverless v1

- 使用 Amazon RDS API– 使用 `ModifyCluster` 操作，并为 `EnableHttpEndpoint` 参数指定 `True` 或 `False`（如果适用）。
- 使用 AWS CLI – 使用带有 `--enable-http-endpoint` 或 `--no-enable-http-endpoint` 选项的 `modify-db-cluster` 操作（如果适用）。

Aurora Serverless v2

- 使用 Amazon RDS API – 使用 `EnableHttpEndpoint` 和 `DisableHttpEndpoint` 操作。
- 使用 AWS CLI：使用 `enable-http-endpoint` 和 `disable-http-endpoint` 操作。

CloudTrail 事件

Aurora Serverless v1

来自数据 API 调用的事件是管理事件。默认情况下，这些事件会自动包含在跟踪记录中。有关更多信息，请参阅 the section called “从 CloudTrail 跟踪记录中排除数据 API 事件（仅限 Aurora Serverless v1）”。

Aurora Serverless v2

来自数据 API 调用的事件是数据事件。默认情况下，这些事件会自动排除在跟踪记录之外。有关更多信息，请参阅 the section called “在 CloudTrail 跟踪记录中包含数据 API 事件”。

多语句支持

Aurora Serverless v1

- 对于 Aurora MySQL，不支持多语句。
- 对于 Aurora PostgreSQL，多语句仅返回第一个查询响应。

Aurora Serverless v2

不支持多语句。尝试在单个 API 调用中执行多条语句会返回 “`An error occurred (ValidationException) when calling the ExecuteStatement operation: Multistatements aren't supported.`”。要执行多条语句，请使用单独的 `ExecuteStatement` API 调用或使用 `BatchExecuteStatement` API 进行批处理。

以下示例显示了 API 调用在尝试执行多条语句时产生的错误消息。

```
aws rds-data execute-statement \
   --resource-arn "arn:aws:rds:region:account:cluster:cluster-name" \
   --secret-arn "arn:aws:secretsmanager:region:account:secret:secret-name" \
   --database "your_database" \
   --sql "SELECT * FROM your_table; Select * FROM next_table;
```

```
                                    "An error occurred (ValidationException) when calling
 the ExecuteStatement operation: Multistatements aren't supported.
```

以下示例使用单独 ExecuteStatement API 调用执行多条语句。

```
aws rds-data execute-statement \
    --resource-arn "arn:aws:rds:region:account:cluster:cluster-name" \
    --secret-arn "arn:aws:secretsmanager:region:account:secret:secret-name" \
    --database "your_database" \
    --sql "SELECT * FROM your_table;"

aws rds-data execute-statement \
    --resource-arn "arn:aws:rds:region:account:cluster:cluster-name" \
    --secret-arn "arn:aws:secretsmanager:region:account:secret:secret-name" \
    --database "your_database" \
    --sql "SELECT * FROM next_table;"
```

对于同一事务 ID 的并发请求

Aurora Serverless v1

后续请求将等到当前请求完成。如果等待时间过长，应用程序需要处理超时错误。

Aurora Serverless v2

数据 API 收到多个具有相同事务 ID 的请求时，会立即返回以下错误：

`DatabaseErrorException: Transaction is still running a query`

此错误会在两种情况下发生：

- 您的应用程序使用相同的事务 ID 发出异步请求（如 JavaScript promise）。
- 之前使用相同事务 ID 的请求仍在处理中。

以下示例显示与 `promise.all()` 并行执行的所有请求。

```
const api_calls = [];
for (let i = 0; i < 10; i++) {
api_calls.push(
    client.send(
    new ExecuteStatementCommand({
        ...params,
```

```
        sql: `insert into table_name values (i);`,
        transactionId
    })
    )
);
}
await Promise.all(api_calls);
```

要解决此错误，请等待当前请求完成，然后再发送具有相同事务 ID 的另一个请求，或者删除事务 ID 以允许并行请求。

以下示例显示的 API 调用使用了具有相同事务 ID 的顺序执行。

```
 for (let i = 0; i < 10; i++) {
    await client.send(
    new ExecuteStatementCommand({
        ...params,
        sql: `insert into table_name values (i);`,
        transactionId
    })
    ).promise()
);
}
```

BatchExecuteStatement 行为

有关 `BatchExecuteStatement` 的更多信息，请参阅 BatchExecuteStatement。

Aurora Serverless v1

更新结果中生成的字段对象包括插入的值。

Aurora Serverless v2

- 对于 Aurora MySQL，更新结果中生成的字段对象包括插入的值。
- 对于 Aurora PostgreSQL，生成的字段对象为空。

ExecuteSQL 行为

有关 `ExecuteSQL` 的更多信息，请参阅 ExecuteSQL。

Aurora Serverless v1

ExecuteSQL 操作已弃用。

Aurora Serverless v2

不支持 ExecuteSQL 操作。

ExecuteStatement 行为

有关 ExecuteStatement 的更多信息，请参阅 ExecuteStatement。

Aurora Serverless v1

ExecuteStatement 参数支持检索多维数组列和所有高级数据类型。

Aurora Serverless v2

ExecuteStatement 参数不支持多维数组列。它也不支持某些 PostgreSQL 数据类型，包括几何类型和货币类型。数据 API 在遇到不支持的数据类型时，会返回以下错误：UnsupportedResultException: The result contains the unsupported data type data_type。

要解决此问题，请将不支持的数据类型强制转换为 TEXT。以下示例将不支持的数据类型转换为 TEXT。

```
SELECT custom_type::TEXT FROM my_table;--
ORSELECT CAST(custom_type AS TEXT) FROM my_table;
```

有关各个 Aurora 数据库引擎支持的数据类型的列表，请参阅数据 API 操作参考。

Schema 参数行为

Aurora Serverless v1

不支持 Schema 参数。当您在 API 调用中时包含 Schema 参数时，数据 API 会忽略该参数。

Aurora Serverless v2

Schema 参数已弃用。当您在 API 调用中时包含 Schema 参数时，数据 API 会返回以下错误：ValidationException: The schema parameter isn't supported。以下示例显示返回 ValidationException 错误的数据 API 调用。

```
aws rds-data execute-statement \
--resource-arn "arn:aws:rds:region:account:cluster:cluster-name" \
```

```
--secret-arn "arn:aws:secretsmanager:region:account:secret:secret-name" \
--database "your_database" \
--schema "your_schema" \
--sql "SELECT * FROM your_table LIMIT 10"
```

要解决此问题，请删除 API 调用中的 Schema 参数。

以下示例显示删除了 Schema 参数的数据 API 调用。

```
aws rds-data execute-statement \
--resource-arn "arn:aws:rds:region:account:cluster:cluster-name" \
--secret-arn "arn:aws:secretsmanager:region:account:secret:secret-name" \
--database "your_database" \
--sql "SELECT * FROM your_table LIMIT 10"
```

授予对 Amazon RDS 数据 API 的访问权限

用户只有在获得授权的情况下才能调用 Amazon RDS 数据 API（数据 API）操作。您可以通过附加定义用户权限的 AWS Identity and Access Management（IAM）策略，授予用户使用数据 API 的权限。如果您使用的是 IAM 角色，您还可以将策略附加到角色。AWS 托管式策略 AmazonRDSDataFullAccess 包含数据 API 的权限。

AmazonRDSDataFullAccess 策略还包含允许用户从 AWS Secrets Manager 获取密钥值的权限。用户需要使用 Secrets Manager 来存储他们在调用数据 API 时可以使用的密钥。使用密钥意味着用户不需要在调用数据 API 时提供其目标资源的数据库凭证。数据 API 会透明地调用 Secrets Manager，后者会允许（或拒绝）用户的密钥请求。有关如何设置与数据 API 一起使用的密钥的信息，请参阅在 AWS Secrets Manager 中存储数据库凭证。

AmazonRDSDataFullAccess 策略提供对资源的完全访问权限（通过数据 API 进行访问）。您可以通过定义指定资源的 Amazon 资源名称（ARN）的策略来缩小范围。

例如，以下策略显示了用户访问数据库集群（由其 ARN 标识）的数据 API 所需的最低权限的示例。该策略包含用户访问 Secrets Manager 和获取数据库实例授权所需的权限。

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Sid": "SecretsManagerDbCredentialsAccess",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue"
            ],
            "Resource": "arn:aws:secretsmanager:*:*:secret:rds-db-credentials/*"
        },
        {
            "Sid": "RDSDataServiceAccess",
            "Effect": "Allow",
            "Action": [
                "rds-data:BatchExecuteStatement",
                "rds-data:BeginTransaction",
                "rds-data:CommitTransaction",
                "rds-data:ExecuteStatement",
                "rds-data:RollbackTransaction"
            ],
            "Resource": "arn:aws:rds:us-east-2:111122223333:cluster:prod"
        }
    ]
}
```

建议您为策略语句中的 "Resources" 元素使用具体的 ARN（如示例所示），而不是通配符 (*)。

使用基于标签的授权

RDS 数据 API（数据 API）和 Secrets Manager 都支持基于标签的授权。标签是用附加的字符串值标记资源（如 RDS 集群）的键值对，例如：

- `environment:production`
- `environment:development`

您可以出于成本分配、操作支持、访问控制及许多其他原因，为资源应用标签。（如果您的资源上还没有标签，并且您想要应用标签，您可以在为 Amazon RDS 资源添加标签中了解更多信息。）您可以在策略语句中使用标签来限制对使用这些标签进行标记的 RDS 集群的访问。例如，Aurora 数据库集群可能包含将其环境标识为生产环境或开发环境的标签。

以下示例说明如何在策略语句中使用标签。这条语句要求在 Data API 请求中传递的集群和密钥都包含 `environment:production` 标签。

策略的应用方式如下：当用户使用数据 API 进行调用时，请求被发送到服务。数据 API 首先验证在请求中传递的集群 ARN 是否包含 `environment:production` 标签。然后，它会调用 Secrets Manager 以在请求中检索用户密钥的值。Secrets Manager 还会验证用户的密钥是否已使用 `environment:production` 进行标记。如果包含，Data API 将使用检索到的值作为用户的数据库密码。最后，如果此密码正确，则会为用户成功调用 Data API 请求。

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "SecretsManagerDbCredentialsAccess",
            "Effect": "Allow",
            "Action": [
                 "secretsmanager:GetSecretValue"
               ],
            "Resource": "arn:aws:secretsmanager:*:*:secret:rds-db-credentials/*",
            "Condition": {
                    "StringEquals": {
                        "aws:ResourceTag/environment": [
                                        "production"
                                       ]
                     }
             }
        },
        {
            "Sid": "RDSDataServiceAccess",
            "Effect": "Allow",
            "Action": [
                  "rds-data:*"
               ],
            "Resource": "arn:aws:rds:us-east-2:111122223333:cluster:*",
            "Condition": {
                    "StringEquals": {
                        "aws:ResourceTag/environment": [
                                        "production"
                                       ]
                     }
             }
         }
     ]
```

```
}
```

该示例分别显示了针对数据 API 和 Secrets Manager 的 `rds-data` 和 `secretsmanager` 操作。不过，您可以通过多种不同的方式组合操作和定义标签条件，以支持您的特定使用案例。有关更多信息，请参阅为 Secrets Manager 使用基于身份的策略（IAM 策略）。

在策略的 "Condition" 元素中，您可以从以下选项中选择标签键：

- `aws:TagKeys`
- `aws:ResourceTag/${TagKey}`

要了解有关资源标签以及如何使用 `aws:TagKeys` 的更多信息，请参阅使用资源标签控制对 AWS 资源的访问。

> **Note**
>
> 数据 API 和 AWS Secrets Manager 都会对用户进行授权。如果您不具有策略中定义的所有操作的权限，则会收到 `AccessDeniedException` 错误。

在 AWS Secrets Manager 中存储数据库凭证

调用 Amazon RDS 数据 API（数据 API）时，您可以使用 Secrets Manager 中的密钥传递 Aurora 数据库集群的凭证。要通过此方式传递凭证，您需要指定密钥的名称或密钥的 Amazon 资源名称（ARN）。

在密钥中存储数据库集群凭证

1. 使用 Secrets Manager 创建包含 Aurora 数据库集群凭证的密钥。

   有关说明，请参阅《AWS Secrets Manager 用户指南》中的创建数据库密钥。
2. 使用 Secrets Manager 控制台查看您创建的密钥的详细信息，或运行 `aws secretsmanager describe-secret` AWS CLI 命令。

   记下密钥的名称和 ARN。您可以将其用于对数据 API 的调用中。

有关使用 Secrets Manager 的更多信息，请参阅 AWS Secrets Manager 用户指南。

要了解 Amazon Aurora 如何管理身份和访问管理，请参阅 Amazon Aurora 如何使用 IAM。

有关创建 IAM 策略的更多信息，请参阅 IAM 用户指南中的创建 IAM 策略。有关将 IAM 策略添加到用户的信息，请参阅《IAM 用户指南》 中的添加和删除 IAM 身份权限。

启用 Amazon RDS 数据 API

要使用 Amazon RDS 数据 API（数据 API），请为 Aurora 数据库集群启用它。您可以在创建或修改数据库集群时启用数据 API。

> **Note**
>
> 数据 API 是否适用于您的集群取决于您的 Aurora 版本、数据库引擎和 AWS 区域。对于较早的 Aurora 版本，数据 API 仅适用于 Aurora Serverless v1 集群。对于较新的 Aurora 版本，数据 API 适用于同时使用预调配和 Aurora Serverless v2 实例的集群。要检查您的集群是否可以使用数据 API，请参阅支持 RDS 数据 API 的区域和 Aurora 数据库引擎。

主题

- 在创建数据库时启用 RDS 数据 API
- 在现有数据库上启用 RDS 数据 API

在创建数据库时启用 RDS 数据 API

在创建支持 RDS 数据 API（数据 API）的数据库时，您可以启用此功能。以下过程描述了在使用 AWS Management Console、AWS CLI 或 RDS API 时如何执行此操作。

控制台

要在创建数据库集群时启用数据 API，请在创建数据库页面的连接部分中选中启用 RDS 数据 API 复选框，如以下屏幕截图所示。



有关如何创建可使用 RDS 数据 API 的 Aurora 数据库集群的说明，请参阅以下内容：

- 对于 Aurora Serverless v2 和预调配集群：创建 Amazon Aurora 数据库集群
- 对于 Aurora Serverless v1 – 创建 Aurora Serverless v1 数据库集群

AWS CLI

要在创建 Aurora 数据库集群时启用数据 API，请运行带 `--enable-http-endpoint` 选项的 create-db-cluster AWS CLI 命令。

以下示例创建一个启用了数据 API 的 Aurora PostgreSQL 数据库集群。

对于 Linux、macOS 或 Unix：

```
aws rds create-db-cluster \
    --db-cluster-identifier my_pg_cluster \
    --engine aurora-postgresql \
    --enable-http-endpoint
```

对于 Windows：

```
aws rds create-db-cluster ^
    --db-cluster-identifier my_pg_cluster ^
    --engine aurora-postgresql ^
    --enable-http-endpoint
```

RDS API

要在创建 Aurora 数据库集群时启用数据 API，请使用 CreateDBCluster 操作，并将 `EnableHttpEndpoint` 参数的值设置为 `true`。

在现有数据库上启用 RDS 数据 API

您可以修改支持 RDS 数据 API（数据 API）的数据库集群以启用或禁用此功能。

主题

- 启用或禁用数据 API（Aurora Serverless v2 和预调配）
- 启用或禁用数据 API（仅限 Aurora Serverless v1）

启用或禁用数据 API（Aurora Serverless v2 和预调配）

使用以下过程可在 Aurora Serverless v2 和预调配数据库上启用或禁用数据 API。要在 Aurora Serverless v1 数据库上启用或禁用数据 API，请使用the section called “启用或禁用数据 API（仅限 Aurora Serverless v1）”中的过程。

控制台

您可以使用 RDS 控制台为支持此功能的数据库集群启用或禁用数据 API。为此，请打开要启用或禁用数据 API 的数据库的集群详细信息页面上，然后在连接和安全选项卡上，转到 RDS 数据 API 部分。此部分显示数据 API 的状态，并允许您启用或禁用它。

以下屏幕截图显示 RDS 数据 API 未启用。



AWS CLI

要在现有数据库上启用或禁用数据 API，请运行 enable-http-endpoint 或 disable-http-endpoint AWS CLI 命令，然后指定数据库集群的 ARN。

以下示例启用数据 API。

对于 Linux、macOS 或 Unix：

```
aws rds enable-http-endpoint \
    --resource-arn cluster_arn
```

对于 Windows：

```
aws rds enable-http-endpoint ^
    --resource-arn cluster_arn
```

RDS API

要在现有数据库上启用或禁用数据 API，请使用 EnableHttpEndpoint 和 DisableHttpEndpoint 操作。

启用或禁用数据 API（仅限 Aurora Serverless v1）

使用以下步骤在现有 Aurora Serverless v1 数据库上启用或禁用数据 API。要在 Aurora Serverless v2 和预调配数据库上启用或禁用数据 API，请使用the section called “启用或禁用数据 API”中的过程。

控制台

在修改 Aurora Serverless v1 数据库集群时，您可以在 RDS 控制台的连接部分中启用数据 API。

以下屏幕截图显示了在修改 Aurora 数据库集群时启用的数据 API。



有关如何修改 Aurora Serverless v1 数据库集群的说明，请参阅修改 Aurora Serverless v1 数据库集群。

AWS CLI

要启用或禁用数据 API，请运行 modify-db-cluster AWS CLI 命令，并使用 `--enable-http-endpoint` 或 `--no-enable-http-endpoint`（如果适用）。

以下示例在 `sample-cluster` 上启用数据 API。

对于 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster \
    --db-cluster-identifier sample-cluster \
```

```
    --enable-http-endpoint
```

对于 Windows：

```
aws rds modify-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --enable-http-endpoint
```

RDS API

要启用数据 API，请使用 ModifyDBCluster 操作，并将 `EnableHttpEndpoint` 的值设置为 `true` 或 `false`（如果适用）。

为 Amazon RDS 数据 API 创建 Amazon VPC 端点（AWS PrivateLink）

借助 Amazon VPC，您可以在 Virtual Private Cloud (VPC) 中启动AWS资源（例如 Aurora 数据库集群和应用程序）。AWS PrivateLink 在亚马逊网络上提供了 VPC 和AWS服务之间的私有连接，这种连接具有高度的安全性。通过使用 AWS PrivateLink，您可以创建 Amazon VPC 端点，这可让您根据 Amazon VPC 跨不同的账号和 VPC 连接到服务。有关 AWS PrivateLink 的更多信息，请参阅《Amazon Virtual Private Cloud 用户指南》中的 VPC 端点服务（AWS PrivateLink）。

您可以使用 Amazon VPC 端点调用 RDS 数据 API（数据 API）。使用 Amazon VPC 端点可保持 Amazon VPC 中的应用程序与 AWS 网络中数据 API 间的流量，而无需使用公有 IP 地址。Amazon VPC 端点可帮助您遵守与管理公共互联网连接有关的合规性和法规要求。例如，如果您使用 Amazon VPC 端点，则可保持 Amazon EC2 实例上运行的应用程序和包含端点的 VPC 中的数据 API 之间的流量。

创建 Amazon VPC 端点后，便能开始使用此端点，而无需在应用程序中进行任何代码或配置更改。

为数据 API 创建 Amazon VPC 端点

1. 登录到AWS Management Console并打开 Amazon VPC 控制台，网址：https://console.aws.amazon.com/vpc/。
2. 选择端点，然后选择创建端点。
3. 在 Create Endpoint (创建端点) 页面上，为 Service category (服务类别) 选择 AWS services（服务)。对于 Service Name (服务名称)，请选择 rds-data。



4. 对于 VPC，选择要在其中创建端点的 VPC。

   选择包含用于进行 Data API 调用的应用程序的 VPC。

5. 对于 Subnets（子网），请为运行应用程序的 AWS 服务所使用的每个可用区 (AZ) 选择子网。



   要创建 Amazon VPC 端点，请指定端点可在其中访问的私有 IP 地址范围。为此，请为每个可用区选择子网。执行此操作会将 VPC 端点限制为特定于每个可用区的私有 IP 地址范围，并且还会在每个可用区中创建一个 Amazon VPC 端点。

6. 对于 Enable DNS Name (启用 DNS 名称)，选择 Enable for this endpoint (为此端点启用)。



私有 DNS 会将标准 Data API DNS 主机名 (`https://rds-data.region.amazonaws.com`) 解析为与特定于 Amazon VPC 端点的 DNS 主机名关联的私有 IP 地址。因此，您可以使用 AWS CLI 或 AWS SDK 访问数据 API VPC 端点，而无需进行任何代码或配置更改来更新数据 API 的端点 URL。

7. 对于安全组，选择要与 Amazon VPC 端点关联的安全组。

   选择允许访问运行应用程序的 AWS 服务的安全组。例如，如果 Amazon EC2 实例正在运行您的应用程序，请选择允许访问 Amazon EC2 实例的安全组。利用安全组，您可以控制从 VPC 中的资源发送到 Amazon VPC 端点的流量。

8. 对于 Policy (策略)，选择 Full Access (完全访问) 以允许 Amazon VPC 中的任何人通过此端点访问 Data API。或者，选择 Custom (自定义) 以指定限制访问的策略。

   如果选择 Custom (自定义)，请在策略创建工具中输入策略。

9. 选择Create endpoint。

创建端点后，选择 AWS Management Console中的链接以查看端点详细信息。



端点 Details (详细信息) 选项卡将显示在创建 Amazon VPC 端点时生成的 DNS 主机名。



您可以使用标准端点 (`rds-data.`*`region`*`.amazonaws.com`) 或特定于 VPC 的端点之一来调用 Amazon VPC 中的 Data API。标准 Data API 端点将自动路由到 Amazon VPC 端点。发生此路由的原因是，在创建 Amazon VPC 端点时启用了私有 DNS 主机名。

在数据 API 调用中使用 Amazon VPC 端点时，应用程序和数据 API 之间的所有流量将保留在包含它们的 Amazon VPC 中。可以将 Amazon VPC 端点用于任何类型的 Data API 调用。有关调用数据 API 的信息，请参阅调用 Amazon RDS 数据 API。

调用 Amazon RDS 数据 API

在 Aurora 数据库集群上启用 Amazon RDS 数据 API（数据 API）后，您可以使用数据 API 或 AWS CLI 在 Aurora 数据库集群上运行 SQL 语句。数据 API 支持 AWS SDK 所支持的编程语言。有关更多信息，请参阅用于在AWS上进行构建的工具。

主题

- Amazon RDS 数据 API 操作参考
- 使用 AWS CLI 调用 Amazon RDS 数据 API
- 从 Python 应用程序调用 Amazon RDS 数据 API
- 从 Java 应用程序调用 Amazon RDS 数据 API
- 控制数据 API 超时行为

Amazon RDS 数据 API 操作参考

Amazon RDS 数据 API 提供了以下操作来执行 SQL 语句。

Data API 操作	AWS CLI command	描述
ExecuteStatement	aws rds-data execute-statement	对数据库运行 SQL 语句。
BatchExecuteStatement	aws rds-data batch-execute-statement	对数据数组运行批处理 SQL 语句，以执行批量更新和插入操作。您可以使用参数集数组运行数据操作语言 (DML) 语句。相比单个插入和更新语句，批处理 SQL 语句可提供显著的性能改进。

您可以使用任一操作来运行单独的 SQL 语句或运行事务。针对事务，数据 API 提供了以下操作。

Data API 操作	AWS CLI command	描述
BeginTransaction	aws rds-data begin-transaction	开始 SQL 事务。
CommitTransaction	aws rds-data commit-transaction	结束 SQL 事务并提交更改。
RollbackTransaction	aws rds-data rollback-transaction	执行事务回滚。

执行 SQL 语句和支持事务的操作具有以下常用 Data API 参数和 AWS CLI 选项。某些操作支持其他参数或选项。

Data API 操作参数	AWS CLI 命令选项	必需	描述
resourceArn	--resource-arn	是	Aurora 数据库集群的 Amazon 资源名称（ARN）。集群必须与调用数据 API 的 IAM 角色或用户位于相同的 AWS 账户

Data API 操作参数	AWS CLI 命令选项	必需	描述
			中。要使用其它账户访问集群，请代入该账户中的角色。
secretArn	--secret-arn	是	允许访问数据库集群的密钥的名称或 ARN。

RDS 数据 API 对于 Aurora MySQL 支持以下数据类型：

- TINYINT(1), BOOLEAN, BOOL
- TINYINT
- SMALLINT [SIGNED | UNSIGNED]
- MEDIUMINT [SIGNED | UNSIGNED]
- INT [SIGNED | UNSIGNED]
- BIGINT [SIGNED | UNSIGNED]
- FLOAT
- DOUBLE
- VARCHAR, CHAR, TEXT, ENUM
- VARBINARY, BINARY, BLOB
- DATE, TIME, DATETIME, TIMESTAMP
- DECIMAL
- JSON
- BIT, BIT(N)

RDS 数据 API 支持以下 Aurora PostgreSQL 标量类型：

- BOOL
- BYTEA
- DATE
- CIDR

- DECIMAL, NUMERIC
- ENUM
- FLOAT8, DOUBLE PRECISION
- INET
- INT, INT4, SERIAL
- INT2, SMALLINT, SMALLSERIAL
- INT8, BIGINT, BIGSERIAL
- JSONB, JSON
- REAL, FLOAT
- TEXT, CHAR(N), VARCHAR, NAME
- TIME
- TIMESTAMP
- UUID
- VECTOR

RDS 数据 API 支持以下 Aurora PostgreSQL 数组类型：

- BOOL[], BIT[]
- DATE[]
- DECIMAL[], NUMERIC[]
- FLOAT8[], DOUBLE PRECISION[]
- INT[], INT4[]
- INT2[]
- INT8[], BIGINT[]
- JSON[]
- REAL[], FLOAT[]
- TEXT[], CHAR(N)[], VARCHAR[], NAME[]
- TIME[]
- TIMESTAMP[]

- UUID[]

您可以在对 ExecuteStatement 和 BatchExecuteStatement 的 Data API 调用中使用参数，或在运行 AWS CLI 命令 execute-statement 和 batch-execute-statement 时使用。要使用参数，请在 SqlParameter 数据类型中指定名称/值对。您可以使用 Field 数据类型指定值。下表将 Java 数据库连接 (JDBC) 数据类型映射到您在 Data API 调用中指定的数据类型。

JDBC 数据类型	Data API 数据类型
INTEGER, TINYINT, SMALLINT, BIGINT	LONG – 、 或 STRING
FLOAT, REAL, DOUBLE	DOUBLE
DECIMAL	STRING
BOOLEAN, BIT	BOOLEAN
BLOB, BINARY, LONGVARBINARY, VARBINARY	BLOB
CLOB	STRING
其他类型（包括与日期和时间有关的类型）	STRING

> **Note**
>
> 您可以在 Data API 调用中为数据库返回的 LONG 值指定 LONG 或 STRING 数据类型。建议您这样操作，以避免在使用 JavaScript 时发生超大数字失去精准性的情况。

某些类型（例如 DECIMAL 和 TIME）需要提示，以便数据 API 将 String 值作为正确的类型传递给数据库。要使用提示，typeHint 数据类型中需要包含 SqlParameter 的值。typeHint 的可能值如下所示：

- DATE – 相应的 String 参数值作为 DATE 类型的对象发送到数据库。接受的格式为 YYYY-MM-DD。
- DECIMAL – 相应的 String 参数值作为 DECIMAL 类型的对象发送到数据库。
- JSON – 相应的 String 参数值作为 JSON 类型的对象发送到数据库。

- `TIME` – 相应的 `String` 参数值作为 `TIME` 类型的对象发送到数据库。接受的格式为 `HH:MM:SS[.FFF]`。
- `TIMESTAMP` – 相应的 `String` 参数值作为 `TIMESTAMP` 类型的对象发送到数据库。接受的格式为 `YYYY-MM-DD HH:MM:SS[.FFF]`。
- `UUID` – 相应的 `String` 参数值作为 `UUID` 类型的对象发送到数据库。

> **Note**
>
> 目前，数据 API 不支持通用唯一标识符（UUID）数组。

> **Note**
>
> 对于 Amazon Aurora PostgreSQL，数据 API 始终以 UTC 时区返回 Aurora PostgreSQL 数据类型 `TIMESTAMPTZ`。

使用 AWS CLI 调用 Amazon RDS 数据 API

您可以使用 AWS CLI 调用 RDS 数据 API（数据 API）。

以下示例将 AWS CLI 用于数据 API。有关更多信息，请参阅 Data API 的 AWS CLI 参考。

在每个示例中，将数据库集群的 Amazon 资源名称（ARN）替换为 Aurora 数据库集群的 ARN。另外，将密钥 ARN 替换为 Secrets Manager 中允许访问该数据库集群的密钥的 ARN。

> **Note**
>
> AWS CLI 可以使用 JSON 设置响应格式。

主题

- 启动 SQL 事务
- 运行 SQL 语句
- 对数据数组运行批处理 SQL 语句
- 提交 SQL 事务
- 回滚 SQL 事务

启动 SQL 事务

您可以使用 `aws rds-data begin-transaction` CLI 命令启动 SQL 事务。调用会返回事务标识符。

> **Important**
>
> 在数据 API 中，如果三分钟之内没有任何调用使用某个事务的 ID，该事务将超时。如果事务在提交之前超时，数据 API 会自动进行回滚。
> 事务内的 MySQL 数据定义语言（DDL，Data Definition Language）语句会导致隐式提交。建议您在单独的 `execute-statement` 命令中，采用 `--continue-after-timeout` 选项运行每个 MySQL DDL 语句。

除了常用选项之外，请指定提供数据库名称的 `--database` 选项。

例如，以下 CLI 命令开始 SQL 事务。

对于 Linux、macOS 或 Unix：

```
aws rds-data begin-transaction --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" \
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret"
```

对于 Windows：

```
aws rds-data begin-transaction --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" ^
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret"
```

以下为响应示例。

```
{
    "transactionId": "ABC1234567890xyz"
}
```

运行 SQL 语句

您可以使用 `aws rds-data execute-statement` CLI 命令运行 SQL 语句。

您可以采用 --transaction-id 选项指定事务标识符，从而在事务中运行 SQL 语句。您可以使用 aws rds-data begin-transaction CLI 命令开始事务。您可以使用 aws rds-data commit-transaction CLI 命令结束并提交事务。

> **Important**
>
> 如果未指定 --transaction-id 选项，则调用产生的更改将自动提交。

除常用选项之外，还可指定以下选项：

- --sql（必需）– 在数据库集群上运行的 SQL 语句。
- --transaction-id（可选）– 使用 begin-transaction CLI 命令开始的事务的标识符。指定您希望包含 SQL 语句的事务的事务 ID。
- --parameters（可选）– SQL 语句的参数。
- --include-result-metadata | --no-include-result-metadata（可选）– 指定是否在结果中包含元数据的值。默认为 --no-include-result-metadata。
- --database（可选）– 数据库的名称。

  当您在先前的请求中运行 --sql "use *database_name*;" 之后运行 SQL 语句时，--database 选项可能不起作用。建议您使用 --database 选项，而不是运行 --sql "use *database_name*;" 语句。
- --continue-after-timeout | --no-continue-after-timeout（可选）– 一个值，指定当调用超过了数据 API 超时间隔（45 秒）后，是否继续运行语句。默认为 --no-continue-after-timeout。

  对于数据定义语言（DDL）语句，我们建议在调用超时后继续运行语句，以避免错误以及数据结构损坏的可能性。
- --format-records-as "JSON"|"NONE" – 一个可选值，它指定是否将结果集格式化为 JSON 字符串。默认值为 "NONE"。有关处理 JSON 结果集的使用信息，请参阅处理 JSON 格式的 Amazon RDS 数据 API 查询结果。

数据库集群为调用返回响应。

> **Note**
>
> 响应大小限制为 1MiB。如果调用返回的响应数据超过 1MiB，则调用将终止。

对于 Aurora Serverless v1，每秒最大请求数为 1000。对于所有其它受支持的数据库，没有限制。

例如，以下 CLI 命令运行单个 SQL 语句，并在结果中省略元数据（默认设置）。

对于 Linux、macOS 或 Unix：

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" \
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" \
--sql "select * from mytable"
```

对于 Windows：

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" ^
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" ^
--sql "select * from mytable"
```

以下为响应示例。

```
{
    "numberOfRecordsUpdated": 0,
    "records": [
        [
            {
                "longValue": 1
            },
            {
                "stringValue": "ValueOne"
            }
        ],
        [
            {
                "longValue": 2
            },
            {
                "stringValue": "ValueTwo"
            }
```

```
        ],
        [
            {
                "longValue": 3
            },
            {
                "stringValue": "ValueThree"
            }
        ]
    ]
}
```

以下 CLI 命令通过指定 `--transaction-id` 选项，在事务中运行单个 SQL 语句。

对于 Linux、macOS 或 Unix：

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" \
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" \
--sql "update mytable set quantity=5 where id=201" --transaction-id "ABC1234567890xyz"
```

对于 Windows：

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" ^
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" ^
--sql "update mytable set quantity=5 where id=201" --transaction-id "ABC1234567890xyz"
```

以下为响应示例。

```
{
    "numberOfRecordsUpdated": 1
}
```

以下 CLI 命令使用参数运行单个 SQL 语句。

对于 Linux、macOS 或 Unix：

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" \
```

```
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" \
--sql "insert into mytable values (:id, :val)" --parameters "[{\"name\": \"id\",
 \"value\": {\"longValue\": 1}},{\"name\": \"val\", \"value\": {\"stringValue\":
 \"value1\"}}]"
```

对于 Windows：

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" ^
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" ^
--sql "insert into mytable values (:id, :val)" --parameters "[{\"name\": \"id\",
 \"value\": {\"longValue\": 1}},{\"name\": \"val\", \"value\": {\"stringValue\":
 \"value1\"}}]"
```

以下为响应示例。

```
{
    "numberOfRecordsUpdated": 1
}
```

以下 CLI 命令运行数据定义语言 (DDL) SQL 语句。DDL 语句将列 `job` 重命名为列 `role`。

> **Important**
>
> 对于 DDL 语句，我们建议在调用超时后继续运行语句。如果 DDL 语句在结束运行之前终止，则可能导致错误以及数据结构损坏。要在调用超过 RDS 数据 API 超时间隔（45 秒）后继续运行语句，请指定 `--continue-after-timeout` 选项。

对于 Linux、macOS 或 Unix：

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" \
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" \
--sql "alter table mytable change column job role varchar(100)" --continue-after-
timeout
```

对于 Windows：

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" ^
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" ^
--sql "alter table mytable change column job role varchar(100)" --continue-after-
timeout
```

以下为响应示例。

```
{
    "generatedFields": [],
    "numberOfRecordsUpdated": 0
}
```

> **Note**
>
> Aurora PostgreSQL 不支持 generatedFields 数据。若要获取生成字段的值，请使用 RETURNING 子句。有关更多信息，请参阅 PostgreSQL 文档中的从已修改的行返回数据。

对数据数组运行批处理 SQL 语句

您可以通过使用 aws rds-data batch-execute-statement CLI 命令，对数据数组运行批处理 SQL 语句。您可以使用该命令执行批量导入或更新操作。

您可以采用 --transaction-id 选项指定事务标识符，从而在事务中运行 SQL 语句。您可以使用 aws rds-data begin-transaction CLI 命令开始事务。您可以使用 aws rds-data commit-transaction CLI 命令结束并提交事务。

> **Important**
>
> 如果未指定 --transaction-id 选项，则调用产生的更改将自动提交。

除常用选项之外，还可指定以下选项：

- --sql（必需）– 在数据库集群上运行的 SQL 语句。

> **Tip**
>
> 对于与 MySQL 兼容的语句，不要在 `--sql` 参数末尾包含分号。尾随的分号可能会导致语法错误。

- `--transaction-id`（可选）– 使用 `begin-transaction` CLI 命令开始的事务的标识符。指定您希望包含 SQL 语句的事务的事务 ID。
- `--parameter-set`（可选）– 批处理操作的参数集。
- `--database`（可选）– 数据库的名称。

数据库集群返回调用的响应。

> **Note**
>
> 参数集数量没有固定的上限。但是，通过数据 API 提交的 HTTP 请求的最大大小为 4MiB。如果请求超出此限制，数据 API 将返回错误并且不处理请求。此 4MiB 限制包括 HTTP 标头的大小和请求中的 JSON 符号。因此，可以包含的参数集数取决于因素的组合，例如 SQL 语句的大小和每个参数集的大小。
>
> 响应大小限制为 1MiB。如果调用返回的响应数据超过 1MiB，则调用将终止。
>
> 对于 Aurora Serverless v1，每秒最大请求数为 1000。对于所有其它受支持的数据库，没有限制。

例如，以下 CLI 命令采用参数集对数据数组运行批处理 SQL 语句。

对于 Linux、macOS 或 Unix：

```
aws rds-data batch-execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" \
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" \
--sql "insert into mytable values (:id, :val)" \
--parameter-sets "[[{\"name\": \"id\", \"value\": {\"longValue\": 1}},{\"name\":
 \"val\", \"value\": {\"stringValue\": \"ValueOne\"}}],
[{\"name\": \"id\", \"value\": {\"longValue\": 2}},{\"name\": \"val\", \"value\":
 {\"stringValue\": \"ValueTwo\"}}],
[{\"name\": \"id\", \"value\": {\"longValue\": 3}},{\"name\": \"val\", \"value\":
 {\"stringValue\": \"ValueThree\"}}]]"
```

对于 Windows：

```
aws rds-data batch-execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" ^
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" ^
--sql "insert into mytable values (:id, :val)" ^
--parameter-sets "[[{\"name\": \"id\", \"value\": {\"longValue\": 1}},{\"name\":
 \"val\", \"value\": {\"stringValue\": \"ValueOne\"}}],
[{\"name\": \"id\", \"value\": {\"longValue\": 2}},{\"name\": \"val\", \"value\":
 {\"stringValue\": \"ValueTwo\"}}],
[{\"name\": \"id\", \"value\": {\"longValue\": 3}},{\"name\": \"val\", \"value\":
 {\"stringValue\": \"ValueThree\"}}]]"
```

> **Note**
>
> `--parameter-sets` 选项中请勿包括换行符。

提交 SQL 事务

通过使用 `aws rds-data commit-transaction` CLI 命令，您可以结束使用 `aws rds-data begin-transaction` 开始的 SQL 事务并提交更改。

除常用选项之外，还可指定以下选项：

- `--transaction-id`（必需）– 使用 `begin-transaction` CLI 命令开始的事务的标识符。指定您希望结束并提交的事务的事务 ID。

例如，以下 CLI 命令结束 SQL 事务并提交更改。

对于 Linux、macOS 或 Unix：

```
aws rds-data commit-transaction --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" \
--secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret" \
--transaction-id "ABC1234567890xyz"
```

对于 Windows：

```
aws rds-data commit-transaction --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" ^
--secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret" ^
--transaction-id "ABC1234567890xyz"
```

以下为响应示例。

```
{
    "transactionStatus": "Transaction Committed"
}
```

回滚 SQL 事务

通过使用 `aws rds-data rollback-transaction` CLI 命令，您可以回滚使用 `aws rds-data begin-transaction` 开始的 SQL 事务。回滚事务会取消其更改。

> **Important**
>
> 如果事务 ID 已过期，事务将自动回滚。在此情况下，指定已过期事务 ID 的 `aws rds-data rollback-transaction` 命令将返回错误。

除常用选项之外，还可指定以下选项：

- `--transaction-id`（必需）– 使用 `begin-transaction` CLI 命令开始的事务的标识符。指定您希望回滚的事务的事务 ID。

例如，以下 AWS CLI 命令回滚 SQL 事务。

对于 Linux、macOS 或 Unix：

```
aws rds-data rollback-transaction --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" \
--secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret" \
--transaction-id "ABC1234567890xyz"
```

对于 Windows：

```
aws rds-data rollback-transaction --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" ^
```

```
--secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret" ^
--transaction-id "ABC1234567890xyz"
```

以下为响应示例。

```
{
    "transactionStatus": "Rollback Complete"
    }
```

从 Python 应用程序调用 Amazon RDS 数据 API

您可以从 Python 应用程序调用 Amazon RDS 数据 API（数据 API）。

以下示例使用适用于 Python 的 AWS 开发工具包（Boto）。有关 Boto 的更多信息，请参阅适用于 Python 的 AWS 开发工具包（Boto 3）文档。

在每个示例中，将数据库集群的 Amazon 资源名称（ARN）替换为您的 Aurora 数据库集群的 ARN。另外，将密钥 ARN 替换为 Secrets Manager 中允许访问该数据库集群的密钥的 ARN。

主题

- 运行 SQL 查询
- 运行 DML SQL 语句
- 运行 SQL 事务

运行 SQL 查询

您可以运行 SELECT 语句并使用 Python 应用程序提取结果。

以下示例运行 SQL 查询。

```
import boto3

rdsData = boto3.client('rds-data')

cluster_arn = 'arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster'
secret_arn = 'arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret'

response1 = rdsData.execute_statement(
            resourceArn = cluster_arn,
            secretArn = secret_arn,
```

```
            database = 'mydb',
            sql = 'select * from employees limit 3')

print (response1['records'])
[
    [
        {
            'longValue': 1
        },
        {
            'stringValue': 'ROSALEZ'
        },
        {
            'stringValue': 'ALEJANDRO'
        },
        {
            'stringValue': '2016-02-15 04:34:33.0'
        }
    ],
    [
        {
            'longValue': 1
        },
        {
            'stringValue': 'DOE'
        },
        {
            'stringValue': 'JANE'
        },
        {
            'stringValue': '2014-05-09 04:34:33.0'
        }
    ],
    [
        {
            'longValue': 1
        },
        {
            'stringValue': 'STILES'
        },
        {
            'stringValue': 'JOHN'
        },
        {
```

```
            'stringValue': '2017-09-20 04:34:33.0'
        }
    ]
]
```

运行 DML SQL 语句

您可以运行数据操作语言 (DML) 语句，在数据库中插入、更新或删除数据。在 DML 语句中也可以使用参数。

> **Important**
>
> 如果调用由于未包含 `transactionID` 参数而不属于事务的一部分，则调用产生的更改将自动提交。

以下示例运行插入 SQL 语句并使用参数。

```
import boto3

cluster_arn = 'arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster'
secret_arn = 'arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret'

rdsData = boto3.client('rds-data')


param1 = {'name':'firstname', 'value':{'stringValue': 'JACKSON'}}
param2 = {'name':'lastname', 'value':{'stringValue': 'MATEO'}}
paramSet = [param1, param2]

response2 = rdsData.execute_statement(resourceArn=cluster_arn,
                                      secretArn=secret_arn,
                                      database='mydb',
                                      sql='insert into employees(first_name, last_name)
 VALUES(:firstname, :lastname)',
                                      parameters = paramSet)

print (response2["numberOfRecordsUpdated"])
```

运行 SQL 事务

您可以通过 Python 应用程序开始 SQL 事务、运行一个或多个 SQL 语句，然后提交更改。

> **Important**
>
> 如果三分钟之内没有任何调用使用其事务 ID，事务将超时。如果事务在提交之前超时，则会自动回滚。
> 如果未指定事务 ID，则调用产生的更改将自动提交。

以下示例运行 SQL 事务在表中插入行。

```
import boto3

rdsData = boto3.client('rds-data')

cluster_arn = 'arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster'
secret_arn = 'arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret'

tr = rdsData.begin_transaction(
     resourceArn = cluster_arn,
     secretArn = secret_arn,
     database = 'mydb')

response3 = rdsData.execute_statement(
     resourceArn = cluster_arn,
     secretArn = secret_arn,
     database = 'mydb',
     sql = 'insert into employees(first_name, last_name) values('XIULAN', 'WANG')',
     transactionId = tr['transactionId'])

cr = rdsData.commit_transaction(
     resourceArn = cluster_arn,
     secretArn = secret_arn,
     transactionId = tr['transactionId'])

cr['transactionStatus']
'Transaction Committed'

response3['numberOfRecordsUpdated']
1
```

> **Note**
>
> 如果运行数据定义语言 (DDL) 语句，我们建议在调用超时后继续运行语句。如果 DDL 语句在结束运行之前终止，则可能导致错误以及数据结构损坏。要在调用超过 RDS 数据 API 超时间隔（45 秒）后继续运行语句，请将 `continueAfterTimeout` 参数设置为 `true`。

从 Java 应用程序调用 Amazon RDS 数据 API

您可以从 Java 应用程序调用 Amazon RDS 数据 API（数据 API）。

以下示例使用适用于 Java 的 AWS 开发工具包。有关更多信息，请参阅 适用于 Java 的 AWS SDK 开发人员指南。

在每个示例中，将数据库集群的 Amazon 资源名称（ARN）替换为您的 Aurora 数据库集群的 ARN。另外，将密钥 ARN 替换为 Secrets Manager 中允许访问该数据库集群的密钥的 ARN。

主题

- 运行 SQL 查询
- 运行 SQL 事务
- 运行批处理 SQL 操作

运行 SQL 查询

您可以运行 `SELECT` 语句并使用 Java 应用程序提取结果。

以下示例运行 SQL 查询。

```
package com.amazonaws.rdsdata.examples;

import com.amazonaws.services.rdsdata.AWSRDSData;
import com.amazonaws.services.rdsdata.AWSRDSDataClient;
import com.amazonaws.services.rdsdata.model.ExecuteStatementRequest;
import com.amazonaws.services.rdsdata.model.ExecuteStatementResult;
import com.amazonaws.services.rdsdata.model.Field;

import java.util.List;

public class FetchResultsExample {
```

```
  public static final String RESOURCE_ARN = "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster";
  public static final String SECRET_ARN = "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret";

  public static void main(String[] args) {
    AWSRDSData rdsData = AWSRDSDataClient.builder().build();

    ExecuteStatementRequest request = new ExecuteStatementRequest()
            .withResourceArn(RESOURCE_ARN)
            .withSecretArn(SECRET_ARN)
            .withDatabase("mydb")
            .withSql("select * from mytable");

    ExecuteStatementResult result = rdsData.executeStatement(request);

    for (List<Field> fields: result.getRecords()) {
      String stringValue = fields.get(0).getStringValue();
      long numberValue = fields.get(1).getLongValue();

      System.out.println(String.format("Fetched row: string = %s, number = %d",
 stringValue, numberValue));
    }
  }
}
```

运行 SQL 事务

您可以通过 Java 应用程序开始 SQL 事务、运行一个或多个 SQL 语句，然后提交更改。

> **Important**
>
> 如果三分钟之内没有任何调用使用其事务 ID，事务将超时。如果事务在提交之前超时，则会自动回滚。
>
> 如果未指定事务 ID，则调用产生的更改将自动提交。

以下示例运行 SQL 事务。

```
package com.amazonaws.rdsdata.examples;

import com.amazonaws.services.rdsdata.AWSRDSData;
```

```
import com.amazonaws.services.rdsdata.AWSRDSDataClient;
import com.amazonaws.services.rdsdata.model.BeginTransactionRequest;
import com.amazonaws.services.rdsdata.model.BeginTransactionResult;
import com.amazonaws.services.rdsdata.model.CommitTransactionRequest;
import com.amazonaws.services.rdsdata.model.ExecuteStatementRequest;

public class TransactionExample {
  public static final String RESOURCE_ARN = "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster";
  public static final String SECRET_ARN = "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret";

  public static void main(String[] args) {
    AWSRDSData rdsData = AWSRDSDataClient.builder().build();

    BeginTransactionRequest beginTransactionRequest = new BeginTransactionRequest()
            .withResourceArn(RESOURCE_ARN)
            .withSecretArn(SECRET_ARN)
            .withDatabase("mydb");
    BeginTransactionResult beginTransactionResult =
 rdsData.beginTransaction(beginTransactionRequest);
    String transactionId = beginTransactionResult.getTransactionId();

    ExecuteStatementRequest executeStatementRequest = new ExecuteStatementRequest()
            .withTransactionId(transactionId)
            .withResourceArn(RESOURCE_ARN)
            .withSecretArn(SECRET_ARN)
            .withSql("INSERT INTO test_table VALUES ('hello world!')");
    rdsData.executeStatement(executeStatementRequest);

    CommitTransactionRequest commitTransactionRequest = new CommitTransactionRequest()
            .withTransactionId(transactionId)
            .withResourceArn(RESOURCE_ARN)
            .withSecretArn(SECRET_ARN);
    rdsData.commitTransaction(commitTransactionRequest);
  }
}
```

> **Note**
>
> 如果运行数据定义语言 (DDL) 语句，我们建议在调用超时后继续运行语句。如果 DDL 语句在结束运行之前终止，则可能导致错误以及数据结构损坏。要在调用超过 RDS 数据 API 超时间隔（45 秒）后继续运行语句，请将 `continueAfterTimeout` 参数设置为 `true`。

运行批处理 SQL 操作

您可以使用 Java 应用程序，对数据数组运行批量插入和更新操作。您可以使用参数集数组运行 DML 语句。

> **Important**
>
> 如果未指定事务 ID，则调用产生的更改将自动提交。

以下示例运行批处理插入操作。

```
package com.amazonaws.rdsdata.examples;

import com.amazonaws.services.rdsdata.AWSRDSData;
import com.amazonaws.services.rdsdata.AWSRDSDataClient;
import com.amazonaws.services.rdsdata.model.BatchExecuteStatementRequest;
import com.amazonaws.services.rdsdata.model.Field;
import com.amazonaws.services.rdsdata.model.SqlParameter;

import java.util.Arrays;

public class BatchExecuteExample {
  public static final String RESOURCE_ARN = "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster";
  public static final String SECRET_ARN = "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret";

  public static void main(String[] args) {
      AWSRDSData rdsData = AWSRDSDataClient.builder().build();

    BatchExecuteStatementRequest request = new BatchExecuteStatementRequest()
            .withDatabase("test")
            .withResourceArn(RESOURCE_ARN)
```

```
            .withSecretArn(SECRET_ARN)
            .withSql("INSERT INTO test_table2 VALUES (:string, :number)")
            .withParameterSets(Arrays.asList(
                    Arrays.asList(
                            new SqlParameter().withName("string").withValue(new
 Field().withStringValue("Hello")),
                            new SqlParameter().withName("number").withValue(new
 Field().withLongValue(1L))
                    ),
                    Arrays.asList(
                            new SqlParameter().withName("string").withValue(new
 Field().withStringValue("World")),
                            new SqlParameter().withName("number").withValue(new
 Field().withLongValue(2L))
                    )
            ));

    rdsData.batchExecuteStatement(request);
  }
}
```

控制数据 API 超时行为

对 Data API 的所有调用都是同步的。假设您执行了一个数据 API 操作，该操作运行 SQL 语句，例如 `INSERT` 或 `CREATE TABLE`。如果数据 API 调用成功返回，则当调用返回时，SQL 处理即告完成。

默认情况下，如果操作未在 45 秒内完成处理，则数据 API 会取消该操作并返回超时错误。在这种情况下不会插入数据，也不会创建表，依此类推。

您可以使用数据 API 执行无法在 45 秒内完成的长时间运行的操作。如果您预计对大型表执行的操作（如批量 `INSERT` 或 DDL 操作）所花费的时间超过 45 秒，则可以为 `ExecuteStatement` 操作指定 `continueAfterTimeout` 参数。您的应用程序仍会收到超时错误。但是，操作会继续运行而不会被取消。有关示例，请参阅运行 SQL 事务。

如果在您使用的编程语言中，AWS SDK 对 API 调用或 HTTP 套接字连接有自己的超时时间，请确保所有这些超时时间都超过 45 秒。对于某些 SDK，超时时间默认为小于 45 秒。我们建议将任何特定于 SDK 或特定于客户端的超时时间设置为至少 1 分钟。采用这种方法，您可以避免虽然数据 API 操作成功完成，但应用程序仍可能收到超时错误。这样您就可以确定是否要重试操作。

例如，假设 SDK 向您的应用程序返回超时错误，但是数据 API 操作仍在数据 API 超时间隔内完成。在这种情况下，重试该操作可能会插入重复的数据或以其他方式生成不正确的结果。SDK 可能会自动重试该操作，从而导致数据不正确，虽然您的应用程序没有执行任何操作。

对于 Java 2 SDK 来说，超时间隔尤其重要。在该 SDK 中，API 调用超时和 HTTP 套接字超时默认均为 30 秒。在以下示例中，将这些超时值设置为更高值：

```
public RdsDataClient createRdsDataClient() {
    return RdsDataClient.builder()
        .region(Region.US_EAST_1) // Change this to your desired Region
        .overrideConfiguration(createOverrideConfiguration())
        .httpClientBuilder(createHttpClientBuilder())
        .credentialsProvider(defaultCredentialsProvider()) // Change this to your
 desired credentials provider
        .build();
}

private static ClientOverrideConfiguration createOverrideConfiguration() {
    return ClientOverrideConfiguration.builder()
        .apiCallTimeout(Duration.ofSeconds(60))
        .build();
}

private HttpClientBuilder createHttpClientBuilder() {
    return ApacheHttpClient.builder() // Change this to your desired HttpClient
        .socketTimeout(Duration.ofSeconds(60));
}
```

以下是使用异步数据客户端的等效示例：

```
public static RdsDataAsyncClient createRdsDataAsyncClient() {
    return RdsDataAsyncClient.builder()
        .region(Region.US_EAST_1) // Change this to your desired Region
        .overrideConfiguration(createOverrideConfiguration())
        .credentialsProvider(defaultCredentialsProvider())  // Change this to your
 desired credentials provider
        .build();
}

private static ClientOverrideConfiguration createOverrideConfiguration() {
    return ClientOverrideConfiguration.builder()
        .apiCallAttemptTimeout(Duration.ofSeconds(60))
        .build();
}

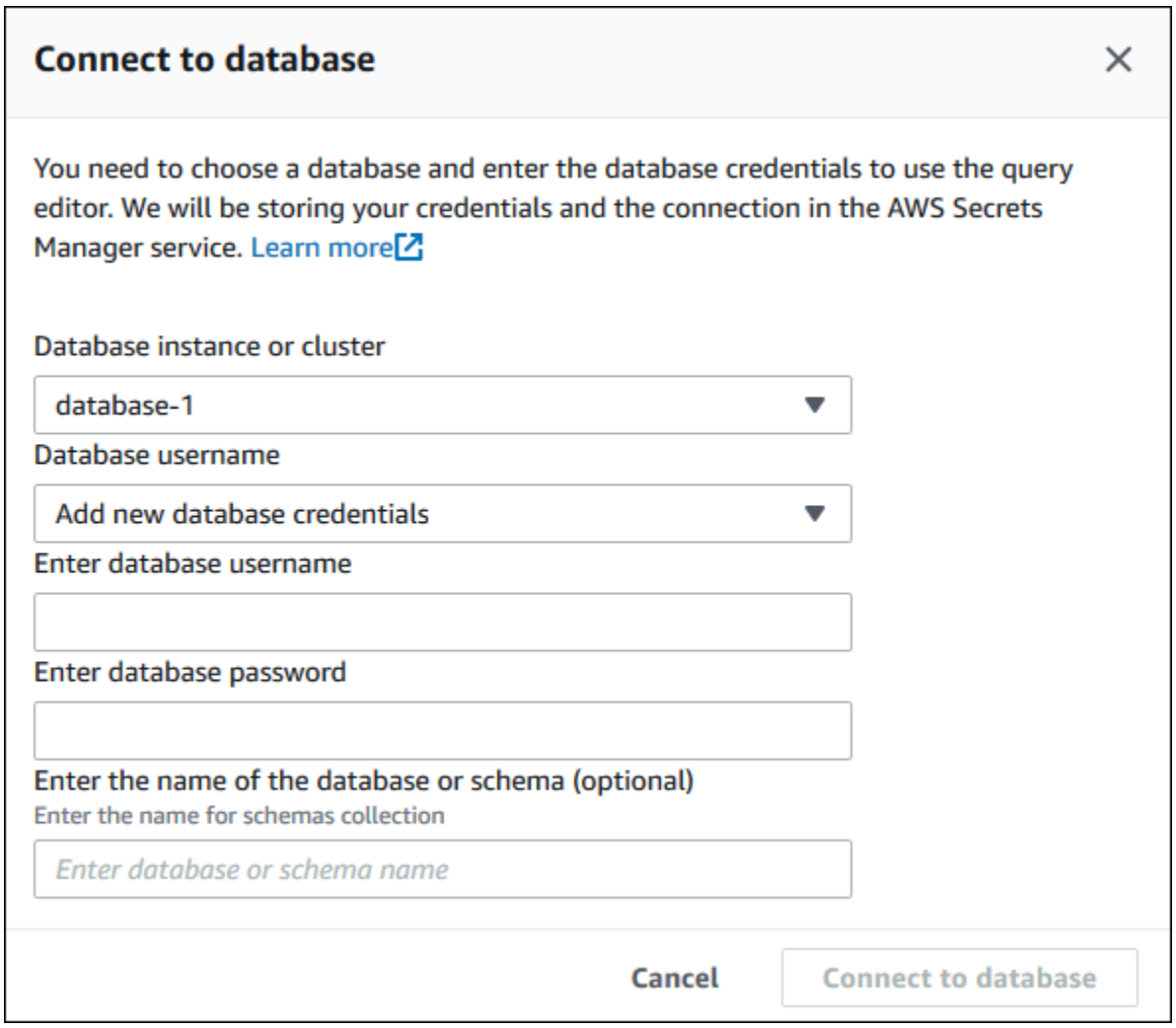
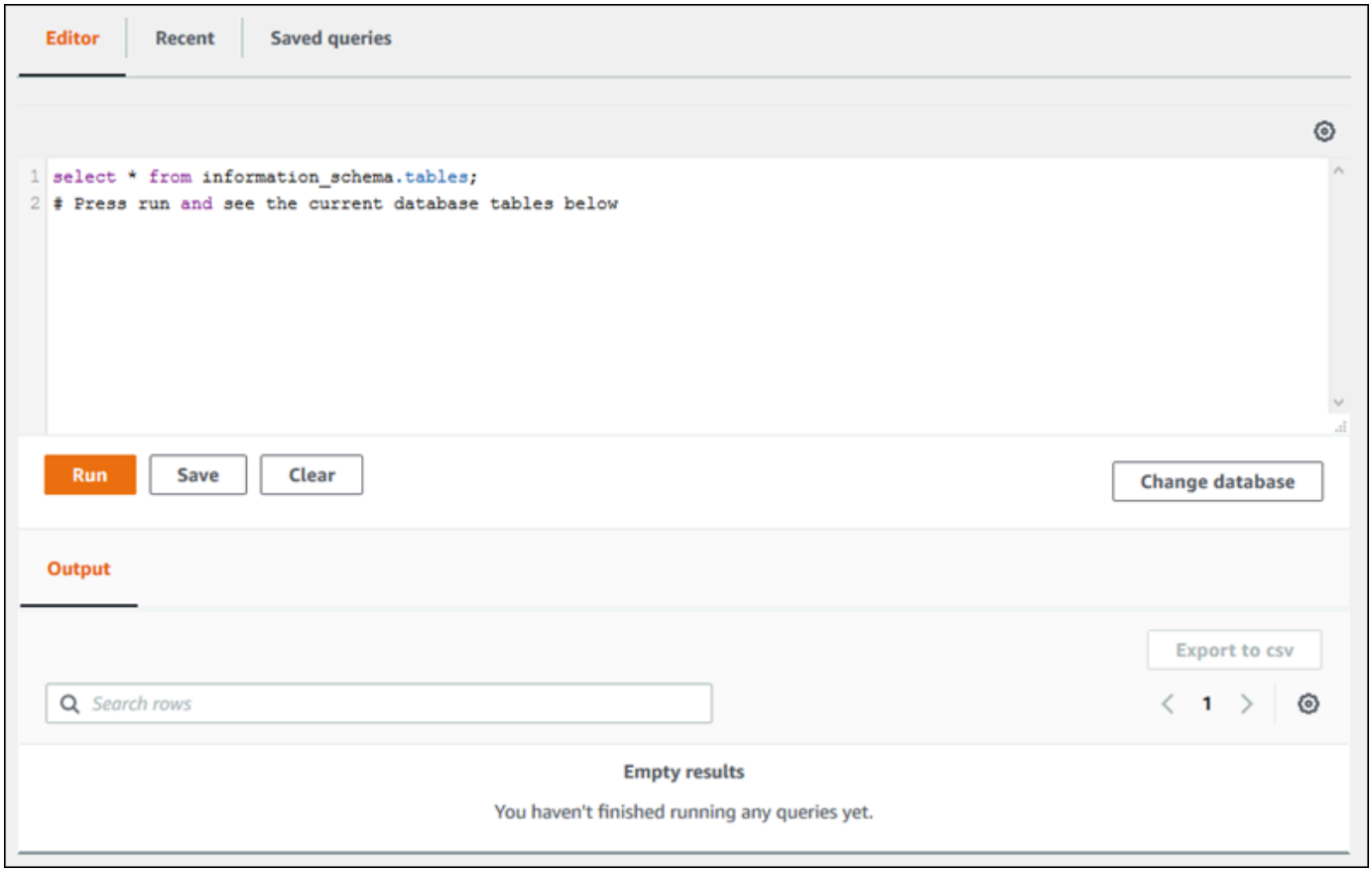
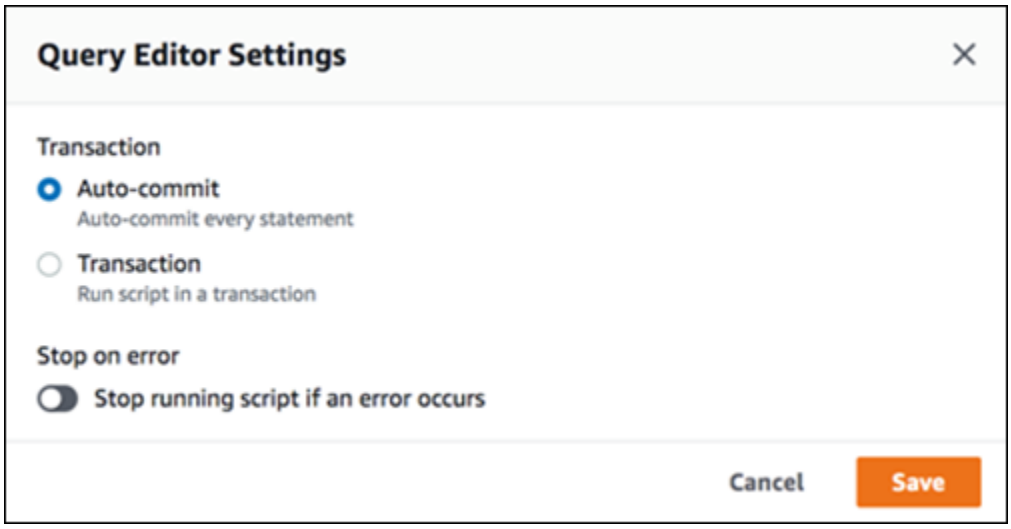
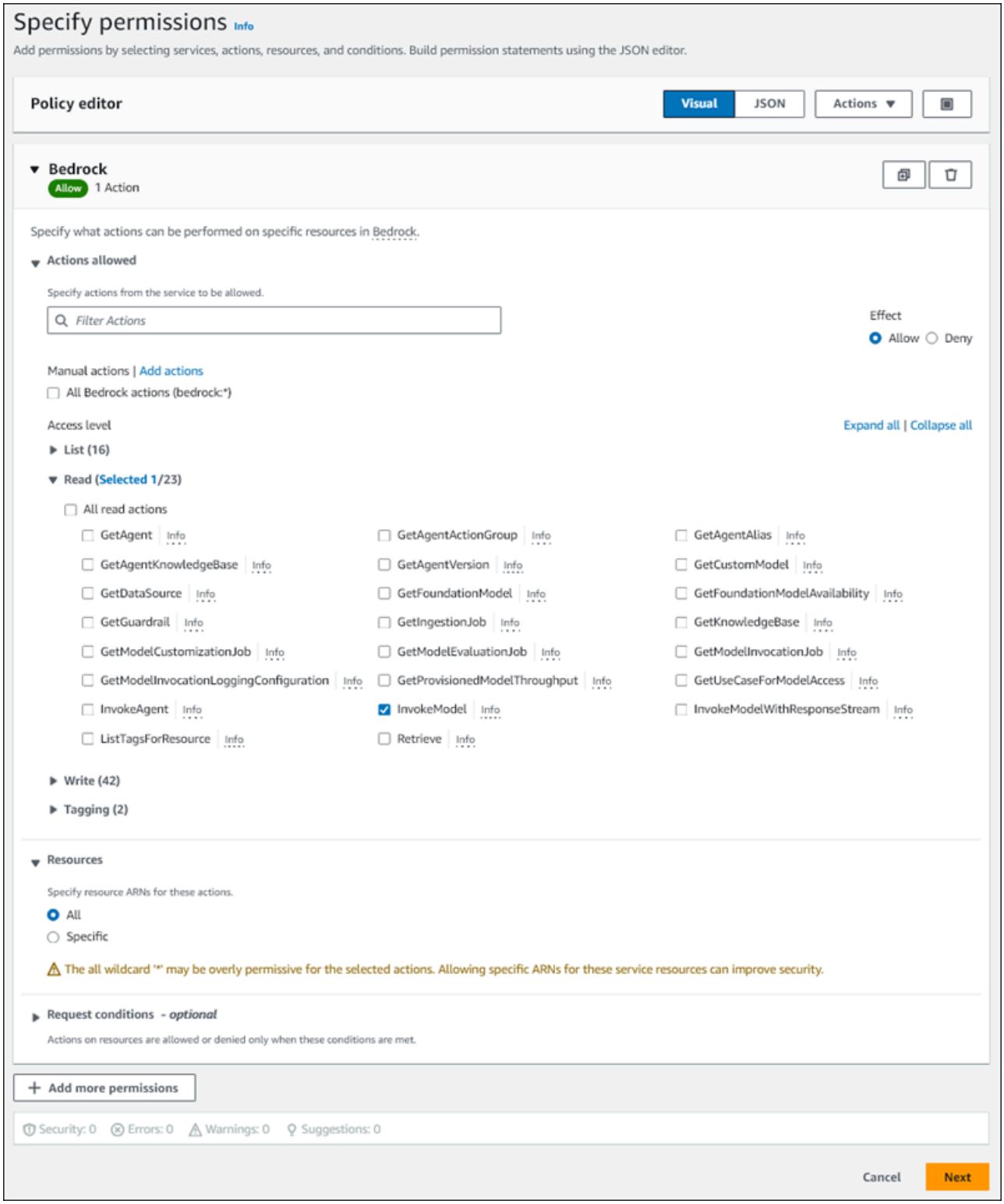
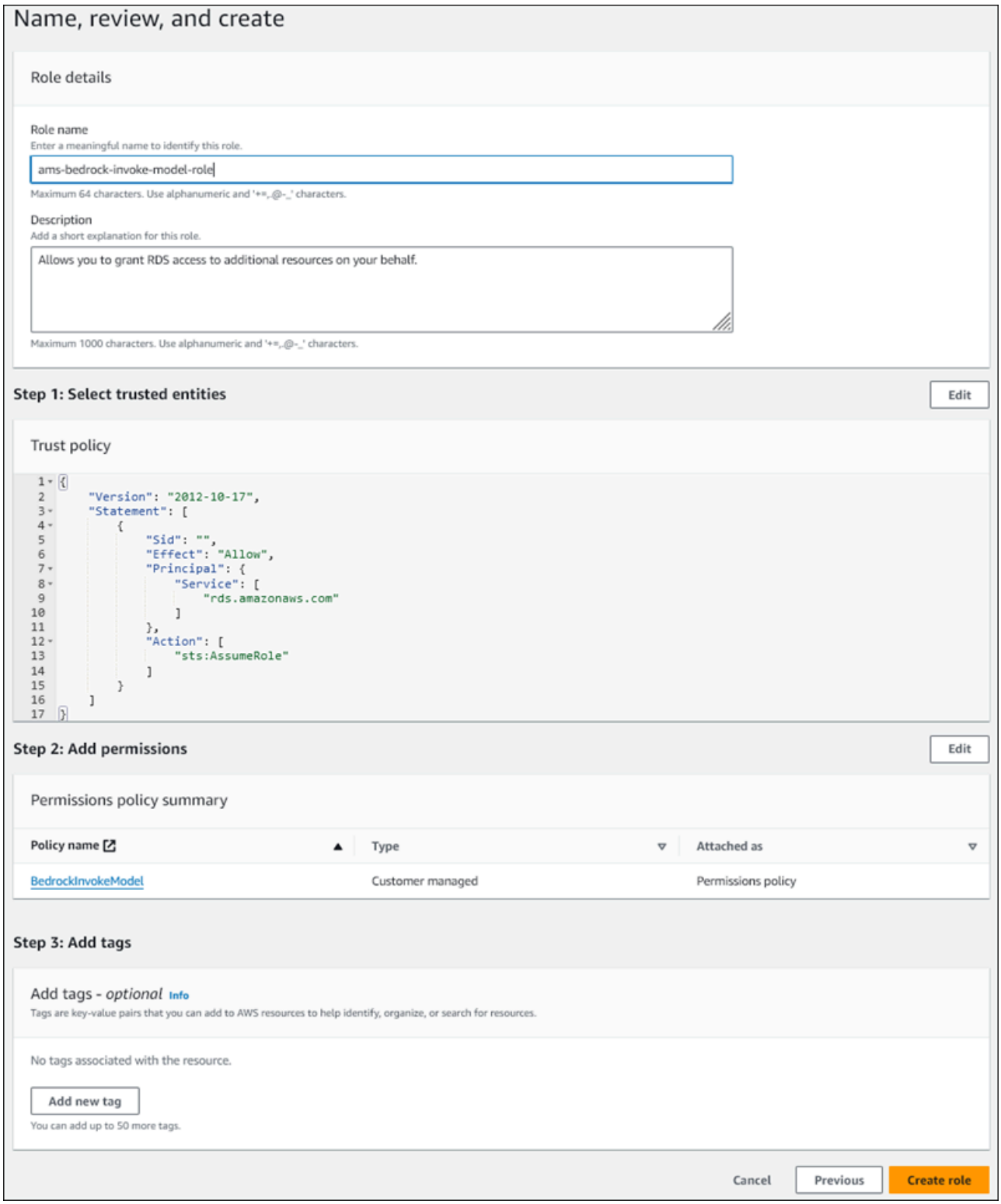
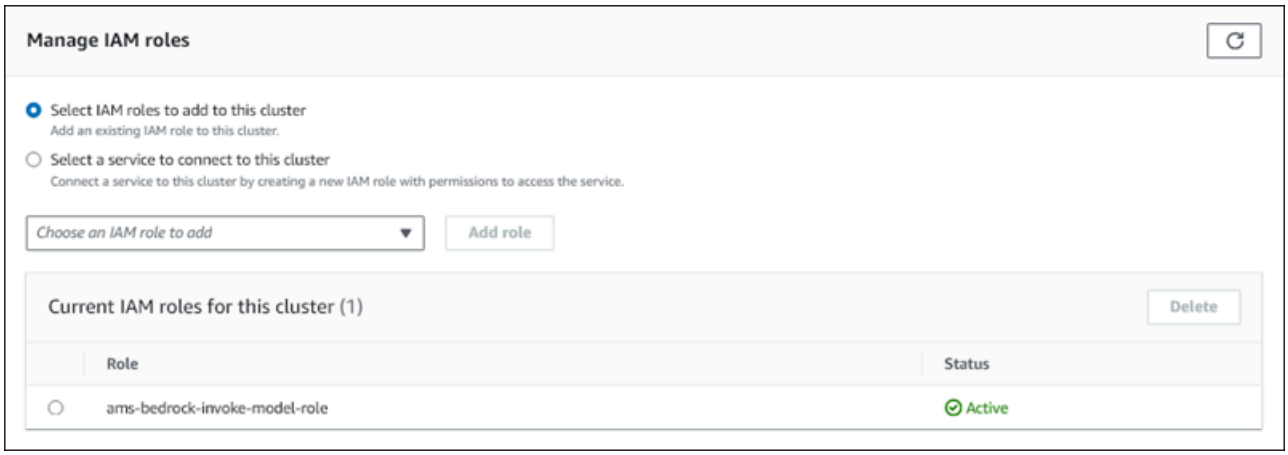
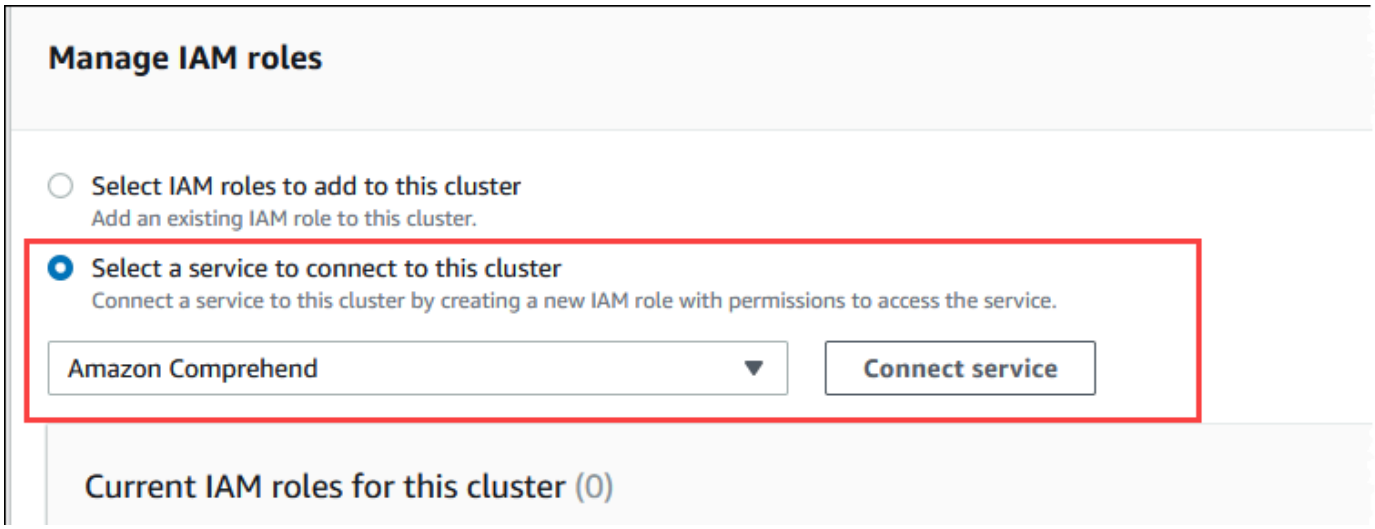
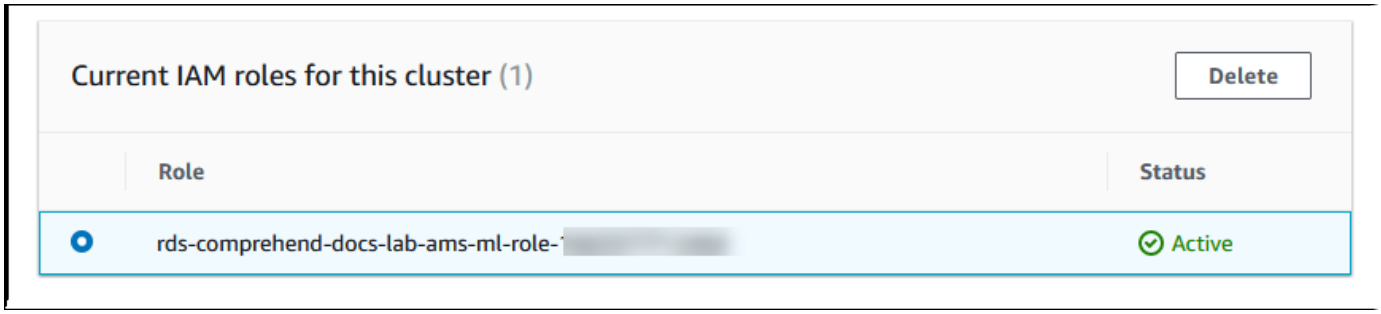
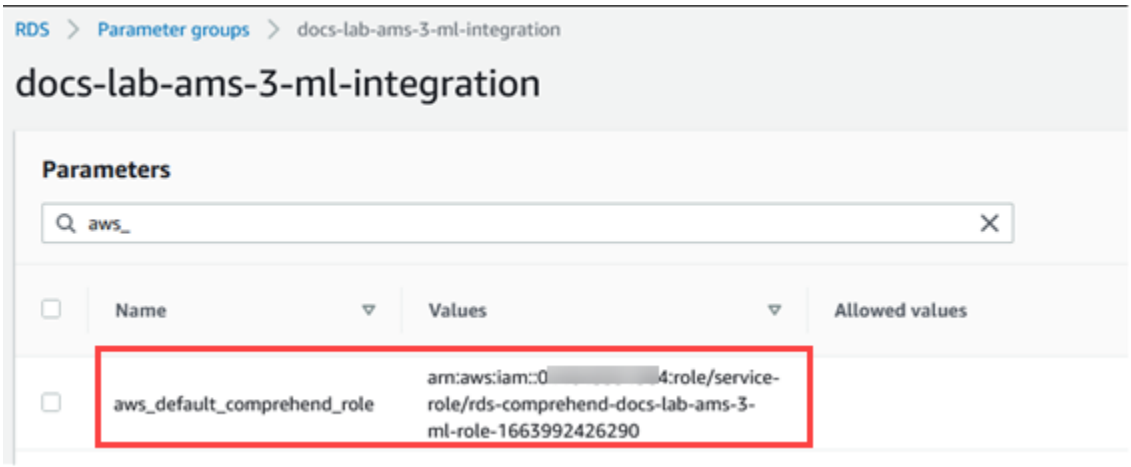
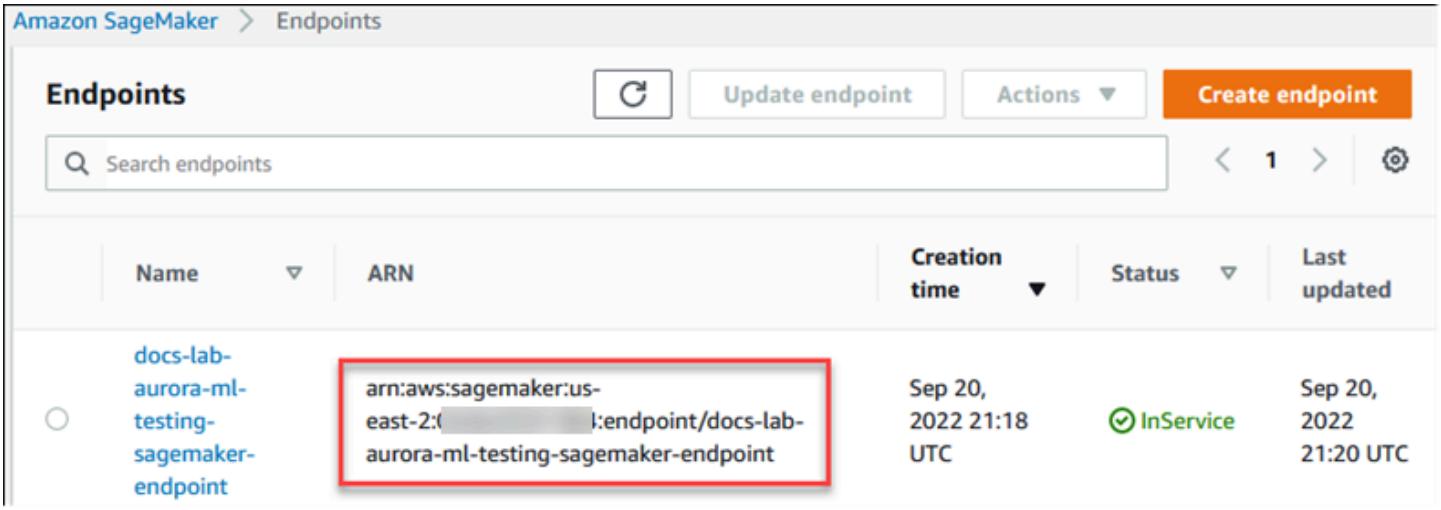
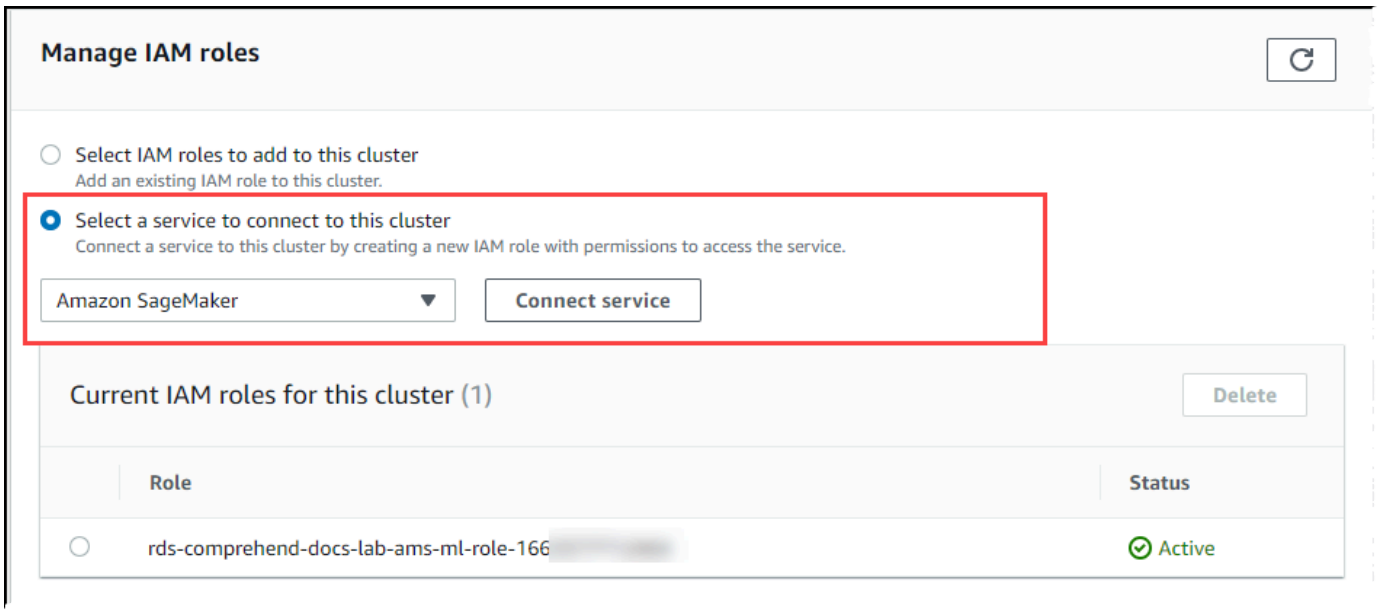
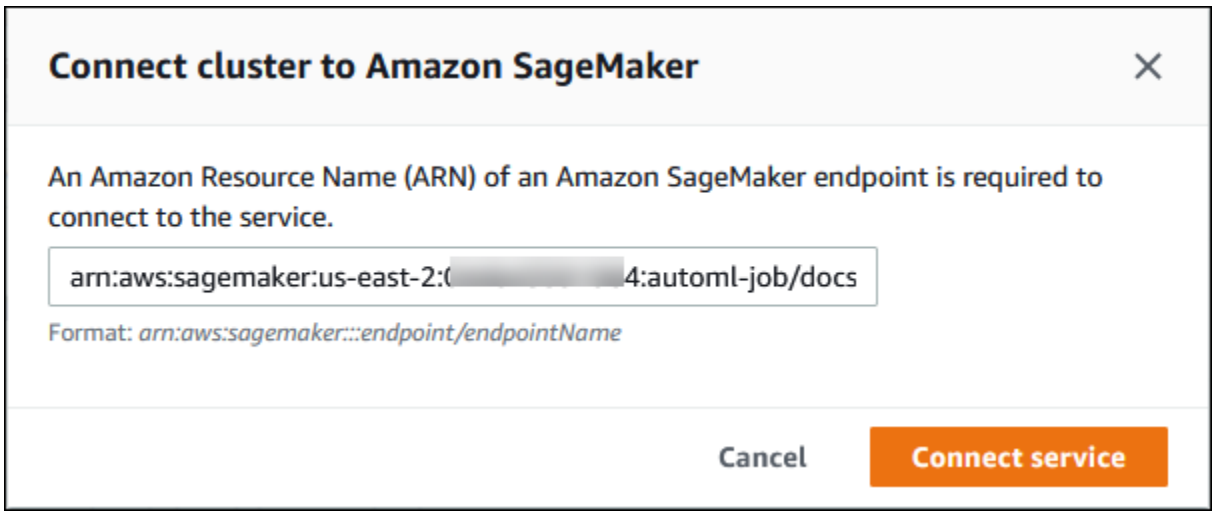
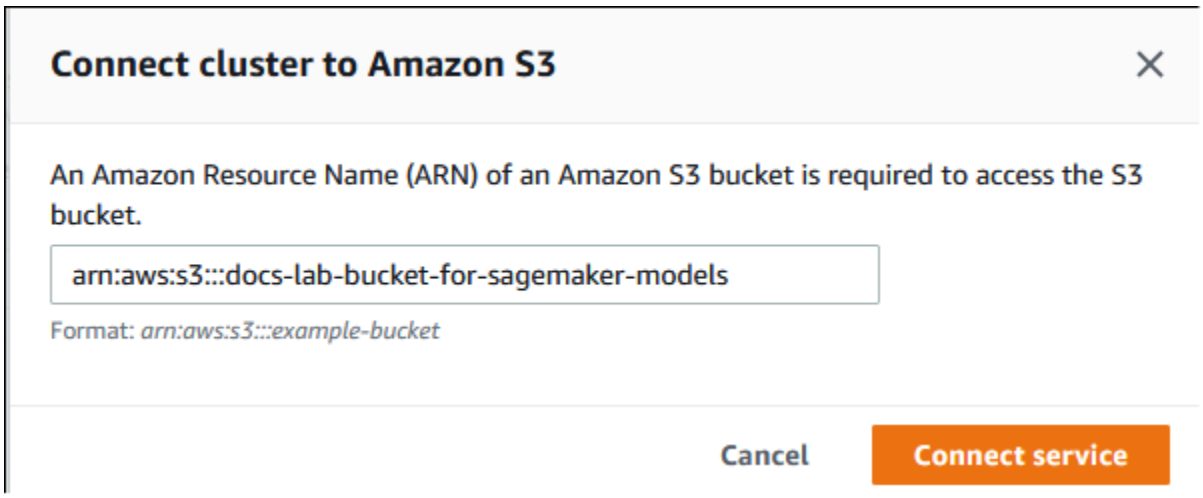
private HttpClientBuilder createHttpClientBuilder() {
```

```
    return NettyNioAsyncHttpClient.builder() // Change this to your desired
 AsyncHttpClient
        .readTimeout(Duration.ofSeconds(60));
}
```

使用适用于 RDS 数据 API 的 Java 客户端库

您可以下载并使用适用于 RDS 数据 API（数据 API）的 Java 客户端库。此 Java 客户端库提供了使用数据 API 的另一种方法。使用该库，您可以将客户端类映射到数据 API 的请求和响应。这种映射支持可以简化与某些特定 Java 类型（如 `Date`、`Time` 和 `BigDecimal`）的集成。

下载适用于 Data API 的 Java 客户端库

Data API Java 客户端库是 GitHub 中是开源的，位于以下位置：

https://github.com/awslabs/rds-data-api-client-library-java

您可以从源文件手动构建该库，但最佳实践是使用 Apache Maven 依赖项管理来使用该库。将以下依赖项添加到 Maven POM 文件中。

对于与 AWS SDK 2.x 兼容的版本 2.x，请使用以下命令：

```
<dependency>
   <groupId>software.amazon.rdsdata</groupId>
   <artifactId>rds-data-api-client-library-java</artifactId>
   <version>2.0.0</version>
</dependency>
```

对于与 AWS SDK 1.x 兼容的版本 1.x，请使用以下命令：

```
<dependency>
    <groupId>software.amazon.rdsdata</groupId>
    <artifactId>rds-data-api-client-library-java</artifactId>
    <version>1.0.8</version>
</dependency>
```

Java 客户端库示例

在下面，您可以找到一些使用 Data API Java 客户端库的常见示例。这些示例假设您有一个包含两列（`accountId` 和 `name`）的表 `accounts`。您还拥有以下数据传输对象 (DTO)。

```
public class Account {
    int accountId;
    String name;
    // getters and setters omitted
}
```

该客户端库使您可以将 DTO 作为输入参数进行传递。以下示例演示了如何将客户 DTO 映射到输入参数集。

```
var account1 = new Account(1, "John");
var account2 = new Account(2, "Mary");
client.forSql("INSERT INTO accounts(accountId, name) VALUES(:accountId, :name)")
         .withParamSets(account1, account2)
         .execute();
```

在某些情况下，使用简单值作为输入参数会更容易。可以使用以下语法来做到这一点。

```
client.forSql("INSERT INTO accounts(accountId, name) VALUES(:accountId, :name)")
         .withParameter("accountId", 3)
         .withParameter("name", "Zhang")
         .execute();
```

下面是另一个使用简单值作为输入参数的示例。

```

client.forSql("INSERT INTO accounts(accountId, name) VALUES(?, ?)", 4, "Carlos")
         .execute();

```

当返回结果时，客户端库提供到 DTO 的自动映射。以下示例演示了如何将结果映射到您的 DTO。

```
List<Account> result = client.forSql("SELECT * FROM accounts")
          .execute()
          .mapToList(Account.class);

Account result = client.forSql("SELECT * FROM accounts WHERE account_id = 1")
          .execute()
          .mapToSingle(Account.class);
```

在很多情况下，数据库结果集只包含一个值。为了简化检索此类结果的过程，客户端库提供了以下 API：

```
int numberOfAccounts = client.forSql("SELECT COUNT(*) FROM accounts")
          .execute()
          .singleValue(Integer.class);
```

> **Note**
>
> mapToList 函数将 SQL 结果集转换为用户定义的对象列表。我们不支持在对 Java 客户端库的 ExecuteStatement 调用中使用 .withFormatRecordsAs(RecordsFormatType.JSON) 语句，因为它具有相同的目的。有关更多信息，请参阅 处理 JSON 格式的 Amazon RDS 数据 API 查询结果。

处理 JSON 格式的 Amazon RDS 数据 API 查询结果

当您调用 ExecuteStatement 操作时，可以选择将查询结果作为 JSON 格式的字符串返回。这样，您就可以使用编程语言的 JSON 解析功能来解释和重新格式化结果集。这样做有助于避免编写额外的代码来循环浏览结果集并解释每个列值。

要请求 JSON 格式的结果集，您需要传递值为 JSON 的可选 formatRecordsAs 参数。JSON 格式的结果集将在 ExecuteStatementResponse 结构的 formattedRecords 字段中返回。

BatchExecuteStatement 操作不返回结果集。因此，JSON 选项不适用于该操作。

要自定义 JSON 哈希结构中的键，请在结果集中定义列别名。为此，您可以使用 SQL 查询的列列表中的 AS 子句。

您可以使用 JSON 功能使结果集更易于阅读，并将其内容映射到特定于语言的框架。由于 ASCII 编码的结果集的卷大于默认表示形式，因此如果查询返回大量的行或较大的列值，占用的内存超过应用程序可用的内存，则您可以选择默认表示形式。

主题

- 检索 JSON 格式的查询结果
- 数据类型映射
- 故障排除
- 示例

检索 JSON 格式的查询结果

要以 JSON 字符串的形式接收结果集，请在 ExecuteStatement 调用中包括 .withFormatRecordsAs(RecordsFormatType.JSON)。返回值在 formattedRecords 字段中以 JSON 字符串的形式返回。在本例中，columnMetadata 为 null。列标签是表示每行的对象的键。这些列名称在结果集中的每一行都会重复出现。列值是带引号的字符串、数值或表示 true、false 或 null 的特殊值。JSON 响应中不会保留列元数据，例如长度约束以及数字和字符串的精确类型。

如果您省略 .withFormatRecordsAs() 调用或指定 NONE 的参数，系统将使用 Records 和 columnMetadata 字段以二进制格式返回结果集。

数据类型映射

结果集中的 SQL 值映射到一组较小的 JSON 类型。这些值在 JSON 中表示为字符串、数字和一些特殊常量，例如 true、false 和null。您可以根据编程语言的不同，使用强类型或弱类型将这些值转换为应用程序中的变量。

JDBC 数据类型	JSON 数据类型
INTEGER, TINYINT, SMALLINT, BIGINT	默认情况下为数字。如果 LongReturnType 选项设置为 STRING，则为字符串。
FLOAT, REAL, DOUBLE	数字
DECIMAL	默认情况下为字符串。如果 DecimalReturnType 选项设置为 DOUBLE_OR_LONG ，则为数字。
STRING	字符串
BOOLEAN, BIT	布尔值
BLOB, BINARY, VARBINARY ，LONGVARBINARY	base64 编码的字符串。
CLOB	字符串
ARRAY	数组

JDBC 数据类型	JSON 数据类型
NULL	null
其他类型（包括与日期和时间有关的类型）	字符串

故障排除

JSON 响应限制为 10 兆字节。如果响应超过此限制，则您的程序会收到 BadRequestException 错误。在这种情况下，您可以使用以下方法之一解决该错误：

- 减少结果集中的行数。为此，请添加 LIMIT 子句。您可以通过提交多个带有 LIMIT 和 OFFSET 子句的查询，将一个大型结果集拆分为多个较小的结果集。

  如果结果集中包含被应用程序逻辑筛选掉的行，您可以通过在 WHERE 子句中添加更多条件，从结果集中删除这些行。
- 减少结果集中的列数。为此，请从查询的选择列表中删除项目。
- 通过在查询中使用列别名来缩短列标签。对于结果集中的每一行，每个列名都会在 JSON 字符串中重复出现。因此，具有长列名和许多行的查询结果可能会超过大小限制。特别是，对复杂的表达式使用列别名，以避免在 JSON 字符串中重复整个表达式。
- 虽然在 SQL 中可以使用列别名来生成具有多个同名列的结果集，但在 JSON 中不允许有重复的键名。如果您请求 JSON 格式的结果集，并且多个列具有相同名称，则 RDS Data API 将返回错误。因此，请确保所有列标签都具有唯一的名称。

示例

以下 Java 示例演示了如何调用 ExecuteStatement 并将响应作为 JSON 格式的字符串，然后解释结果集。用适当的值替换 *databaseName*、*secretStoreArn* 和 *clusterArn* 参数。

以下 Java 示例演示了一个在结果集中返回十进制数值的查询。assertThat 调用会测试响应的字段是否具有基于 JSON 结果集规则的预期属性。

该示例适用于以下架构和示例数据：

```
create table test_simplified_json (a float);
insert into test_simplified_json values(10.0);
```

```
public void JSON_result_set_demo() {
    var sql = "select * from test_simplified_json";
    var request = new ExecuteStatementRequest()
      .withDatabase(databaseName)
      .withSecretArn(secretStoreArn)
      .withResourceArn(clusterArn)
      .withSql(sql)
      .withFormatRecordsAs(RecordsFormatType.JSON);
    var result = rdsdataClient.executeStatement(request);
}
```

之前程序中的 formattedRecords 字段的值为：

```
[{"a":10.0}]
```

由于存在 JSON 结果集，响应中的 Records 和 ColumnMetadata 字段均为 null。

以下 Java 示例演示了一个在结果集中返回整数数值的查询。该示例调用 getFormattedRecords 以仅返回 JSON 格式的字符串，并忽略其他空白或为 null 的响应字段。该示例将结果反序列化为一个表示记录列表的结构。每条记录都有一些字段，字段名称对应于结果集中的列别名。此技术简化了解析结果集的代码。您的应用程序不必循环浏览结果集的行和列并将每个值转换为适当类型。

该示例适用于以下架构和示例数据：

```
create table test_simplified_json (a int);
insert into test_simplified_json values(17);
```

```
public void JSON_deserialization_demo() {
    var sql = "select * from test_simplified_json";
    var request = new ExecuteStatementRequest()
      .withDatabase(databaseName)
      .withSecretArn(secretStoreArn)
      .withResourceArn(clusterArn)
      .withSql(sql)
      .withFormatRecordsAs(RecordsFormatType.JSON);
    var result = rdsdataClient.executeStatement(request)
      .getFormattedRecords();

/* Turn the result set into a Java object, a list of records.
   Each record has a field 'a' corresponding to the column
   labelled 'a' in the result set. */
```

```
    private static class Record { public int a; }
    var recordsList = new ObjectMapper().readValue(
        response, new TypeReference<List<Record>>() {
        });
}
```

之前程序中的 formattedRecords 字段的值为：

```
[{"a":17}]
```

要检索结果行 0 的 a 列，应用程序将引用 recordsList.get(0).a。

相比之下，以下 Java 示例显示了在不使用 JSON 格式的情况下，构建保存结果集的数据结构所需的代码类型。在这种情况下，结果集的每一行都包含一些字段，其中包含有关单个用户的信息。构建表示结果集的数据结构需要循环浏览这些行。对于每一行，代码会检索每个字段的值，执行适当的类型转换，并将结果分配给表示该行的对象中的相应字段。然后，代码将表示每个用户的对象添加到表示整个结果集的数据结构中。如果将查询更改为重新排序、添加或删除结果集中的字段，则也必须更改应用程序代码。

```
/* Verbose result-parsing code that doesn't use the JSON result set format */
for (var row: response.getRecords()) {
    var user = User.builder()
      .userId(row.get(0).getLongValue())
      .firstName(row.get(1).getStringValue())
      .lastName(row.get(2).getStringValue())
      .dob(Instant.parse(row.get(3).getStringValue()))
      .build();
    result.add(user);
  }
```

以下示例值显示了具有不同列数、列别名和列数据类型的结果集的 formattedRecords 字段的值。

如果结果集包含多行，则每一行都表示为一个作为数组元素的对象。结果集中的每一列都成为对象中的一个键。这些键在结果集中的每一行都会重复出现。因此，对于包含许多行和列的结果集，您可能需要定义短列别名，以避免超出整个响应的长度限制。

该示例适用于以下架构和示例数据：

```
create table sample_names (id int, name varchar(128));
insert into sample_names values (0, "Jane"), (1, "Mohan"), (2, "Maria"), (3, "Bruce"),
 (4, "Jasmine");
```

```
[{"id":0,"name":"Jane"},{"id":1,"name":"Mohan"},
{"id":2,"name":"Maria"},{"id":3,"name":"Bruce"},{"id":4,"name":"Jasmine"}]
```

如果结果集中的某一列被定义为表达式，则该表达式的文本将成为 JSON 键。因此，为查询的选择列表中的每个表达式定义一个描述性列别名通常很方便。例如，以下查询在其选择列表中包含诸如函数调用和算术运算之类的表达式。

```
select count(*), max(id), 4+7 from sample_names;
```

这些表达式将作为键传递到 JSON 结果集。

```
[{"count(*)":5,"max(id)":4,"4+7":11}]
```

添加带有描述性标签的 AS 列可以简化 JSON 结果集中键的解释。

```
select count(*) as rows, max(id) as largest_id, 4+7 as addition_result from
 sample_names;
```

在修订后的 SQL 查询中，使用由 AS 子句定义的列标签作为键名。

```
[{"rows":5,"largest_id":4,"addition_result":11}]
```

JSON 字符串中每个键值对的值可以是带引号的字符串。该字符串可能包含 Unicode 字符。如果该字符串包含转义序列或者 " 或 \ 字符，则这些字符前面必须有反斜杠转义字符。以下 JSON 字符串示例演示了这些可能性。例如，string_with_escape_sequences 结果包含特殊字符：退格符、换行符、回车符、制表符、换页符和 \。

```
[{"quoted_string":"hello"}]
[{"unicode_string":"####"}]
[{"string_with_escape_sequences":"\b \n \r \t \f \\ '"}]
```

JSON 字符串中每个键值对的值也可以表示一个数字。该数字可能是整数、浮点值、负值或以指数表示法表示的值。以下 JSON 字符串示例演示了这些可能性。

```
[{"integer_value":17}]
[{"float_value":10.0}]
[{"negative_value":-9223372036854775808,"positive_value":9223372036854775807}]
[{"very_small_floating_point_value":4.9E-324,"very_large_floating_point_value":1.79769313486231
```

布尔值和 null 值用不带引号的特殊关键字 `true`、`false` 和 `null` 表示。以下 JSON 字符串示例演示了这些可能性。

```
[{"boolean_value_1":true,"boolean_value_2":false}]
[{"unknown_value":null}]
```

如果选择 BLOB 类型的值，则结果将在 JSON 字符串中表示为 base64 编码值。要将值转换回原始表示形式，可以使用应用程序语言中的相应解码函数。例如，在 Java 中调用函数 `Base64.getDecoder().decode()`。以下示例输出显示了选择 `hello world` 的 BLOB 值并将结果集作为 JSON 字符串返回的结果。

```
[{"blob_column":"aGVsbG8gd29ybGQ="}]
```

以下 Python 示例展示了如何访问调用 Python `execute_statement` 函数的结果中的值。结果集是字段 `response['formattedRecords']` 中的字符串值。该代码通过调用 `json.loads` 函数，将 JSON 字符串转换为数据结构。然后，结果集的每一行都是数据结构中的一个列表元素，在每一行中，您可以按名称引用结果集的每个字段。

```
import json

result = json.loads(response['formattedRecords'])
print (result[0]["id"])
```

以下 JavaScript 示例展示了如何访问调用 JavaScript `executeStatement` 函数的结果中的值。结果集是字段 `response.formattedRecords` 中的字符串值。该代码通过调用 `JSON.parse` 函数，将 JSON 字符串转换为数据结构。然后，结果集的每一行都是数据结构中的一个数组元素，在每一行中，您可以按名称引用结果集的每个字段。

```
<script>
    const result = JSON.parse(response.formattedRecords);
    document.getElementById("display").innerHTML = result[0].id;
</script>
```

Amazon RDS 数据 API 故障排除

使用以下部分（标题为常见错误消息）帮助解决 Amazon RDS 数据 API（数据 API）问题。

主题

- 未找到事务 <transaction_ID>
- 用于查询的包太大
- 数据库响应超出大小限制
- 没有为集群 <cluster_ID> 启用 HttpEndpoint
- DatabaseErrorException：事务仍在运行查询
- 不支持的结果异常
- 不支持多语句
- 不支持架构参数

未找到事务 <transaction_ID>

在此情况下，找不到 Data API 调用中指定的事务 ID。错误消息提供了导致此问题的原因，是以下原因之一：

- 事务可能已过期。

  确保每个事务调用在上一个事务调用之后的 3 分钟内运行。

  也可能是指定的事务 ID 不是由 BeginTransaction 调用创建的。确保您的调用具有有效的事务 ID。
- 之前的一次调用导致您的事务终止。

  交易已经由您的`CommitTransaction`或者`RollbackTransaction`调用。
- 由于之前的调用发生错误，事务已中止。

  检查您之前的电话是否引发了任何异常。

有关运行事务的信息，请参阅调用 Amazon RDS 数据 API。

用于查询的包太大

在此情况下，为行返回的结果集太大。数据库返回的结果集中的 Data API 大小限制为每行 64 KB。

要解决此问题，请确保结果集中的每一行都小于或等于 64 KB。

数据库响应超出大小限制

在此情况下，数据库返回的结果集太大。数据库返回的结果集中的 Data API 限制为 1MiB。

要解决此问题，请确保对数据 API 的调用返回 1MiB 或更少数据。如果需要返回 1MiB 以上的数据，您可以在查询中将多个 `ExecuteStatement` 调用与 `LIMIT` 子句结合使用。

有关 `LIMIT` 子句的更多信息，请参阅 MySQL 文档中的 SELECT 语法。

没有为集群 <cluster_ID> 启用 HttpEndpoint

请查看以下可能导致该问题的原因：

- Aurora 数据库集群不支持数据 API。有关 RDS 数据 API 支持的数据库集群类型的信息，请参阅the section called “Amazon RDS 数据 API 的区域和版本可用性”。
- 未为 Aurora 数据库集群启用数据 API。要将数据 API 与 Aurora 数据库集群结合使用，必须为该数据库集群启用数据 API。有关启用数据 API 的信息，请参阅启用 Amazon RDS 数据 API。
- 在为数据库集群启用数据 API 之后，已重命名该数据库集群。在这种情况下，请关闭该集群的数据 API，然后再次启用它。
- 您指定的 ARN 与集群的 ARN 不完全匹配。请检查从另一个源返回的 ARN 或由程序逻辑构造的 ARN 是否与集群的 ARN 完全匹配。例如，请确保您使用的 ARN 的所有字母字符的大小写正确。

DatabaseErrorException：事务仍在运行查询

如果您的应用程序发送了带有事务 ID 的请求，而该事务当前正在处理另一个请求，则 Data API 会立即将此错误返回给您的应用程序。如果您的应用程序使用诸如 Javascript 中的“promises”之类的机制发出异步请求，则可能会出现这种情况。

要解决此问题，请等到上一个请求完成，然后重试该请求。您可以继续重试，直到错误不再出现，或者应用程序收到其他类型的错误。

这种情况可能发生在 Aurora Serverless v2 的 Data API 和预配置实例中。在 Aurora Serverless v1 的 Data API 中，对同一事务 ID 的后续请求会自动等待上一个请求完成。但是，由于之前的请求花费了太长时间，这种过时的行为可能会出现超时。如果您要移植发出并发请求的过时 Data API 应用程序，请修改异常处理逻辑以处理这种新错误。

不支持的结果异常

数据 API 并不支持全部的数据类型。当您执行返回不支持的数据类型的查询时，就会发生此错误。

要解决此问题，请将不支持的数据类型强制转换为 `TEXT`。例如：

```
SELECT custom_type::TEXT FROM my_table;
```

```
-- OR
SELECT CAST(custom_type AS TEXT) FROM my_table;
```

不支持多语句

Aurora Serverless v2 和预置集群的数据 API 不支持多语句。尝试在单个 API 调用中执行多条语句会导致此错误。

要执行多条语句，请使用单独的 `ExecuteStatement` API 调用或使用 `BatchExecuteStatement` API 进行批处理。

不支持架构参数

Aurora Serverless v1 会静默忽略架构参数。但是，Aurora Serverless v2 和预置集群会显式拒绝包含架构参数的 API 调用。

要解决此问题，请在使用 Aurora Serverless v2 或预置集群时，从对数据 API 的所有调用中移除架构参数。

使用 AWS CloudTrail 记录 Amazon RDS 数据 API 调用

RDS 数据 API（数据 API）已与 AWS CloudTrail 集成，后者是一项提供数据 API 中由用户、角色或 AWS 服务所采取操作的记录的服务。CloudTrail 将对数据 API 的所有 API 调用作为事件捕获，包括来自 Amazon RDS 控制台的调用和对数据 API 操作的代码调用。如果您创建跟踪，可以使 CloudTrail 事件持续传送到 Amazon S3 存储桶（包括数据 API 的事件）。通过使用 CloudTrail 收集的数据，您可以确定大量的信息。此信息包括向数据 API 发出了什么请求、发出请求的 IP 地址、何人发出的请求、请求的发出时间以及其他详细信息。

要了解有关 CloudTrail 的更多信息，请参阅 AWS CloudTrail 用户指南。

在 CloudTrail 中使用数据 API 信息

在您创建 AWS 账户时，将在该账户上启用 CloudTrail。当数据 API 中发生受支持的活动（管理事件）时，该活动将记录在 CloudTrail 事件中，并与其它 AWS 服务事件一同保存在事件历史记录中。您可以在 AWS 账户中查看、搜索和下载最新管理事件。有关更多信息，请参见《AWS CloudTrail 用户指南》的 使用 CloudTrail 事件历史记录。

要持续记录 AWS 账户中的事件（包括数据 API 的事件），请创建跟踪。通过跟踪，CloudTrail 可将日志文件传送至 Amazon S3 存储桶。预设情况下，在控制台中创建跟踪时，此跟踪应用于所有 AWS 区域。此跟踪记录来自 AWS 分区中的所有 AWS 区域的事件，并将日志文件传送至您指定的 Amazon

S3 存储桶。此外，您可以配置其他 AWS 服务，进一步分析在 CloudTrail 日志中收集的事件数据并采取行动。有关更多信息，请参阅《AWS CloudTrail 用户指南》中的以下主题：

- 创建跟踪记录概述
- CloudTrail 支持的服务和集成
- 为 CloudTrail 配置 Amazon SNS 通知
- 从多个区域接收 CloudTrail 日志文件和从多个账户中接收 CloudTrail 日志文件

所有数据 API 操作都由 CloudTrail 记录并记载在 Amazon RDS 数据服务 API 参考 中。例如，对 `BatchExecuteStatement`、`BeginTransaction`、`CommitTransaction` 和 `ExecuteStatement` 操作的调用将在 CloudTrail 日志文件中生成条目。

每个事件或日志条目都包含有关生成请求的人员信息。身份信息有助于您确定以下内容：

- 请求是使用根凭证还是用户凭证发出的。
- 请求是使用角色还是联合用户的临时安全凭证发出的。
- 请求是否由其他 AWS 服务发出。

有关更多信息，请参阅 CloudTrail userIdentity 元素。

在 AWS CloudTrail 跟踪记录中包含和排除数据 API 事件

大多数的 Data API 用户依赖 AWS CloudTrail 跟踪记录中的事件来提供 Data API 操作的记录。事件数据不会在对数据 API 的请求中显示数据库名称、架构名称或 SQL 语句。但是，了解哪位用户在给定时间对特定数据库集群进行了某种类型的调用，可以帮助检测异常访问模式。

在 AWS CloudTrail 跟踪记录中包含数据 API 事件

对于 Aurora PostgreSQL Serverless v2 和预调配数据库，以下数据 API 操作将作为数据事件记录到 AWS CloudTrail 中。数据事件是大容量数据面板 API 操作，CloudTrail 默认情况下不记录这些操作。记录数据事件将收取额外费用。有关 CloudTrail 定价的信息，请参阅 AWS CloudTrail 定价。

- BatchExecuteStatement
- BeginTransaction
- CommitTransaction
- ExecuteStatement
- RollbackTransaction

您可以使用 CloudTrail 控制台、AWS CLI 或 CloudTrail API 操作来记录这些数据 API 操作。在 CloudTrail 控制台中，为数据事件类型选择 RDS 数据 API - 数据库集群。有关更多信息，请参阅《AWS CloudTrail 用户指南》中的使用 AWS Management Console记录数据事件。

使用 AWS CLI，运行 `aws cloudtrail put-event-selectors` 命令来记录跟踪记录的这些数据 API 操作。要记录数据库集群上的所有数据 API 事件，请为资源类型指定 `AWS::RDS::DBCluster`。以下示例记录了数据库集群上的所有数据 API 事件。有关更多信息，请参阅《AWS CloudTrail 用户指南》中的使用 AWS Command Line Interface 记录数据事件。

```
aws cloudtrail put-event-selectors --trail-name trail_name --advanced-event-selectors \
'{
   "Name": "RDS Data API Selector",
   "FieldSelectors": [
      {
         "Field": "eventCategory",
         "Equals": [
            "Data"
         ]
      },
      {
         "Field": "resources.type",
         "Equals": [
            "AWS::RDS::DBCluster"
         ]
      }
   ]
}'
```

您可以将高级事件选择器配置为对 `readOnly`、`eventName`，和 `resources.ARN` 字段进行额外筛选。有关这些字段的更多信息，请参阅 AdvancedFieldSelector。

从 AWS CloudTrail 跟踪记录中排除数据 API 事件（仅限 Aurora Serverless v1）

对于 Aurora Serverless v1，数据 API 事件是管理事件。默认情况下，所有数据 API 事件都包含在 AWS CloudTrail 跟踪记录中。但是，由于数据 API 可能生成大量事件，因此您可以从 CloudTrail 跟踪记录中排除这些事件。排除 Amazon RDS 数据 API 事件设置将所有数据 API 事件排除在跟踪记录之外。您无法排除特定的数据 API 事件。

要从跟踪记录中排除 Data API 事件，请执行以下操作：

- 创建跟踪记录或者更新跟踪记录时，在 CloudTrail 控制台中选择 Exclude Amazon RDS Data API events (排除 Amazon RDS Data API 事件) 设置。

- 在 CloudTrail API 中，使用 PutEventSelectors 操作。如果您使用的是高级事件选择器，则可以通过将 eventSource 字段设置为不等于 rdsdata.amazonaws.com 来排除 Data API 事件。如果您使用的是基本事件选择器，则可以通过将 ExcludeManagementEventSources 属性的值设置为 rdsdata.amazonaws.com 来排除 Data API 事件。有关更多信息，请参阅《AWS CloudTrail 用户指南》中的使用 AWS Command Line Interface 记录事件。

> **Warning**
>
> 从 CloudTrail 日志中排除 Data API 事件可能会掩盖 Data API 操作。请谨慎赋予委托人执行此操作所需的 cloudtrail:PutEventSelectors 权限。

您可以随时更改跟踪记录的控制台设置或事件选择器，以关闭此排除。随后，跟踪记录将开始记录 Data API 事件。但是，它无法恢复在排除生效期间发生的 Data API 事件。

使用控制台或 API 排除 Data API 事件时，生成的 CloudTrail PutEventSelectors API 操作也会记录在 CloudTrail 日志中。如果 Data API 事件未显示在 CloudTrail 日志中，请查找 PutEventSelectors 属性设置为 ExcludeManagementEventSources 的 rdsdata.amazonaws.com 事件。

有关更多信息，请参阅 AWS CloudTrail 用户指南中的记录管理事件以便进行跟踪记录。

了解数据 API 日志文件条目

跟踪是一种配置，可用于将事件作为日志文件传送到您指定的 Amazon S3 存储桶。CloudTrail 日志文件包含一个或多个日记账条目。一个事件表示来自任何源的一个请求，包括有关所请求的操作、操作的日期和时间、请求参数等方面的信息。CloudTrail 日志文件不是公用 API 调用的有序堆栈跟踪，因此它们不会按任何特定顺序显示。

Aurora PostgreSQL Serverless v2 和预调配

以下示例显示一个 CloudTrail 日志条目，该条目演示 Aurora PostgreSQL Serverless v2 和预调配数据库的 ExecuteStatement 操作。对于这些数据库，所有数据 API 事件都是数据事件，其中事件源为 rdsdataapi.amazonaws.com，事件类型为 Rds 数据服务。

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
```

```
        "principalId": "AKIAIOSFODNN7EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/johndoe",
        "accountId": "123456789012",
        "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
        "userName": "johndoe"
    },
    "eventTime": "2019-12-18T00:49:34Z",
    "eventSource": "rdsdataapi.amazonaws.com",
    "eventName": "ExecuteStatement",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "aws-cli/1.16.102 Python/3.7.2 Windows/10 botocore/1.12.92",
    "requestParameters": {
        "continueAfterTimeout": false,
        "database": "**********",
        "includeResultMetadata": false,
        "parameters": [],
        "resourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:my-database-1",
        "schema": "**********",
        "secretArn": "arn:aws:secretsmanager:us-
east-1:123456789012:secret:dataapisecret-ABC123",
        "sql": "**********"
    },
    "responseElements": null,
    "requestID": "6ba9a36e-b3aa-4ca8-9a2e-15a9eada988e",
    "eventID": "a2c7a357-ee8e-4755-a0d0-aed11ed4253a",
    "eventType": "Rds Data Service",
    "recipientAccountId": "123456789012"
}
```

Aurora Serverless v1

以下示例显示了前面的示例 CloudTrail 日志条目对于 Aurora Serverless v1 的显示方式。对于 Aurora Serverless v1，所有事件都是管理事件，其中事件源为 rdsdata.amazonaws.com，事件类型为 AwsApiCall。

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AKIAIOSFODNN7EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/johndoe",
```

```
        "accountId": "123456789012",
        "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
        "userName": "johndoe"
    },
    "eventTime": "2019-12-18T00:49:34Z",
    "eventSource": "rdsdata.amazonaws.com",
    "eventName": "ExecuteStatement",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "aws-cli/1.16.102 Python/3.7.2 Windows/10 botocore/1.12.92",
    "requestParameters": {
        "continueAfterTimeout": false,
        "database": "**********",
        "includeResultMetadata": false,
        "parameters": [],
        "resourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:my-database-1",
        "schema": "**********",
        "secretArn": "arn:aws:secretsmanager:us-
east-1:123456789012:secret:dataapisecret-ABC123",
        "sql": "**********"
    },
    "responseElements": null,
    "requestID": "6ba9a36e-b3aa-4ca8-9a2e-15a9eada988e",
    "eventID": "a2c7a357-ee8e-4755-a0d0-aed11ed4253a",
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```


使用性能详情监控 RDS 数据 API 查询

如果 Aurora 集群正在运行 Aurora Serverless v2 或预调配实例，则可以将性能详情与 RDS 数据 API 结合使用。

有关如何将性能详情与 Aurora 结合使用的更多信息，请参阅在 Amazon Aurora 上使用性能详情监控数据库负载。

如何在性能详情中表示 RDS 数据 API 查询

借助数据 API，Aurora 集群会根据您从应用程序中提交的数据 API 调用来处理查询。作为其内部工作的一部分，数据库 API 还会执行一些 SQL 语句，例如取消超过超时阈值的查询。这两种 SQL 操作都显示在性能详情统计数据和图表中。

- 对于您提交到 Aurora 集群的数据 API 查询，API 控制面板中的主机字段标记为 RDS 数据 API。对于 Aurora PostgreSQL，application_name 字段的值为 `rds-data-api`。使用热门主机或热门应用程序作为维度来分析数据库负载时，请查找这些标签。
- 数据 API 为管理数据库的各个方面（例如连接池和查询超时）而运行的所有内部查询都标有前缀 RDS 数据 API。示例：`/* RDS Data API */ select * from my_table;` 当您将热门 SQL 作为维度来分析数据库负载时，请查找这些前缀。语句标有 SQL 注释 `/* RDS Data API */`。

使用 Aurora 查询编辑器

利用 Aurora 查询编辑器，您可以通过 AWS Management Console在 Aurora 数据库集群上运行 SQL 语句。您可以运行 SQL 查询、数据操作（DML）语句和数据定义（DDL）语句。使用控制台界面，您可以执行数据库维护、生成报告和进行 SQL 试验。您可以避免将网络配置设置为从单独的客户端系统（例如 EC2 实例或笔记本电脑）连接到数据库集群。

查询编辑器要求 Aurora 数据库集群已启用 RDS 数据 API（数据 API）。有关支持数据 API 的数据库集群以及如何启用它的信息，请参阅使用 Amazon RDS 数据 API。您可以运行的 SQL 受数据 API 限制的约束。有关更多信息，请参阅 the section called “限制”。

查询编辑器的可用性

查询编辑器可用于 Aurora 数据库集群，这些集群使用支持数据 API 的 Aurora MySQL 和 Aurora PostgreSQL 引擎版本，并位于提供数据 API 的 AWS 区域中。有关更多信息，请参阅 支持 RDS 数据 API 的区域和 Aurora 数据库引擎。

授予对查询编辑器的访问权限

用户必须获得在查询编辑器中运行查询的授权。您可以通过向用户添加 `AmazonRDSDataFullAccess` 策略（预定义的 AWS Identity and Access Management（IAM）策略）来授予该用户在查询编辑器中运行查询的权限。

> **Note**
>
> 请确保您在创建 IAM 用户时使用的用户名和密码，与为数据库用户所使用的用户名和密码相同，例如管理用户名和密码。有关更多信息，请参阅《AWS Identity and Access Management 用户指南》中的在您的 AWS 账户中创建 IAM 用户。

您还可以创建 IAM 策略，授予对查询编辑器的访问权限。在创建策略后，将该策略附加到需要对查询编辑器的访问权限的每个用户。

以下策略提供用户访问查询编辑器所需的最低权限。

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "QueryEditor0",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutResourcePolicy",
                "secretsmanager:PutSecretValue",
                "secretsmanager:DeleteSecret",
                "secretsmanager:DescribeSecret",
                "secretsmanager:TagResource"
            ],
            "Resource": "arn:aws:secretsmanager:*:*:secret:rds-db-credentials/*"
        },
        {
            "Sid": "QueryEditor1",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetRandomPassword",
                "tag:GetResources",
                "secretsmanager:CreateSecret",
                "secretsmanager:ListSecrets",
                "dbqms:CreateFavoriteQuery",
                "dbqms:DescribeFavoriteQueries",
                "dbqms:UpdateFavoriteQuery",
                "dbqms:DeleteFavoriteQueries",
                "dbqms:GetQueryString",
                "dbqms:CreateQueryHistory",
                "dbqms:UpdateQueryHistory",
                "dbqms:DeleteQueryHistory",
                "dbqms:DescribeQueryHistory",
                "rds-data:BatchExecuteStatement",
                "rds-data:BeginTransaction",
                "rds-data:CommitTransaction",
                "rds-data:ExecuteStatement",
                "rds-data:RollbackTransaction"
            ],
            "Resource": "*"
```

```
        }
    ]
}
```

有关创建 IAM 策略的更多信息，请参阅 AWS Identity and Access Management 用户指南中的创建 IAM 策略。

有关将 IAM 策略添加到用户的信息，请参阅 AWS Identity and Access Management 用户指南中的添加和删除 IAM 身份权限。

在查询编辑器中运行查询

您可以在查询编辑器中对 Aurora 数据库集群运行 SQL 语句。您可以运行的 SQL 受数据 API 限制的约束。有关更多信息，请参阅 the section called “限制”。

在查询编辑器中运行查询

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在 AWS Management Console的右上角，选择在其中创建了您要查询的 Aurora 数据库集群的 AWS 区域。
3. 在导航窗格中，选择数据库。
4. 选择要对其运行 SQL 查询的 Aurora 数据库集群。
5. 对于 Actions (操作)，选择 Query (查询)。如果您之前未连接到数据库，则 Connect to database (连接到数据库) 页面将打开。



6. 输入以下信息：

   a. 对于数据库实例或集群，选择要对其运行 SQL 查询的 Aurora 数据库集群。

   b. 对于 Database username (数据库用户名)，选择要连接的数据库用户的用户名，或者选择 Add new database credentials (添加新的数据库凭证)。如果选择 Add new database credentials (添加新的数据库凭证)，请在 Enter database username (输入数据库用户名) 中为新的数据库凭证输入用户名。

   c. 对于 Enter database password (输入数据库密码)，请为您选择的数据库用户输入密码。

   d. 在最后一个框中，输入要用于 Aurora 数据库集群的数据库或架构的名称。

   e. 选择 Connect to database (连接到数据库)。

> **Note**
>
> 如果连接成功，则连接和身份验证信息将存储在 AWS Secrets Manager 中。您无需再次输入连接信息。

7. 在查询编辑器中，输入要在数据库上运行的 SQL 查询。



每个 SQL 语句可自动提交，或者您可以在脚本中将 SQL 语句作为事务的一部分运行。要控制此行为，请选择查询窗口上方的齿轮图标。



将显示 Query Editor Settings (查询编辑器设置) 窗口。



如果您选择 Auto-commit (自动提交)，则将自动提交每个 SQL 语句。如果选择事务，则可以在脚本中运行一组语句。语句会在脚本结束时自动提交，除非在此之前已明确提交或回滚。此外，您可以启用 Stop on error (错误时停止)，选择当发生错误时停止正在运行的脚本。

> **Note**
>
> 在一组语句中，数据定义语言 (DDL) 语句可能会造成前面的数据操作语言 (DML) 语句提交。您还可以将 `COMMIT` 和 `ROLLBACK` 语句包含在脚本中的一组语句中。

在 Query Editor Settings (查询编辑器设置) 窗口中完成选择后，选择 Save (保存)。

8. 选择 Run (运行) 或按 Ctrl+Enter，查询编辑器将显示查询的结果。

   运行查询后，选择 Save (保存) 将其保存到 Saved queries (保存的查询)。

   选择 Export to csv (导出到 CSV)，将查询结果导出为电子表格格式。

您可以查找、编辑以及重新运行之前的查询。为此，请选择 Recent (最近) 或 Saved queries (保存的查询) 选项卡，选择查询文本，然后选择 Run (运行)。

要更改数据库，请选择 Change database (更改数据库)。

Database Query Metadata Service (DBQMS) API 参考

Database Query Metadata Service (dbqms) 是一项仅限内部使用的服务。它为AWS Management Console上多项AWS服务（包括 Amazon RDS）提供查询编辑器最近的和保存的查询。

支持的 DBQMS 操作

- CreateFavoriteQuery
- CreateQueryHistory
- CreateTab
- DeleteFavoriteQueries
- DeleteQueryHistory
- DeleteTab
- DescribeFavoriteQueries
- DescribeQueryHistory
- DescribeTabs
- GetQueryString
- UpdateFavoriteQuery
- UpdateQueryHistory
- UpdateTab

CreateFavoriteQuery

保存新的常用查询。每个 用户最多可以创建 1000 个保存的查询。此限制将来可能会发生变化。

CreateQueryHistory

保存新的查询历史记录条目。

CreateTab

保存新的查询选项卡。每个 用户最多可以创建 10 个查询选项卡。

DeleteFavoriteQueries

删除一个或多个保存的查询。

DeleteQueryHistory

删除查询历史记录条目。

DeleteTab

删除查询选项卡条目。

DescribeFavoriteQueries

列出用户在给定账户中创建的保存的查询。

DescribeQueryHistory

列出查询历史记录条目。

DescribeTabs

列出用户在给定账户中创建的查询选项卡。

GetQueryString

从查询 ID 检索完整的查询文本。

UpdateFavoriteQuery

更新查询字符串、描述、名称或到期日期。

UpdateQueryHistory

更新查询历史记录的状态。

UpdateTab

更新查询选项卡名称和查询字符串。

使用 Amazon Aurora 机器学习

通过使用 Amazon Aurora 机器学习，您可以将您的 Aurora 数据库集群与以下 AWS 机器学习服务之一集成，具体取决于您的需求。这些服务分别支持特定的机器学习使用案例。

Amazon Bedrock

Amazon Bedrock 是一项完全托管式服务，可通过 API 提供人工智能公司的领先基础模型，并附带有助于构建和扩展生成式人工智能应用程序的开发工具。使用 Amazon Bedrock，您可以付费来对任何第三方根基模型运行推理。定价基于输入令牌和输出令牌的数量，以及您是否为模型购买了预调配吞吐量。有关更多信息，请参阅《Amazon Bedrock 用户指南》中的什么是 Amazon Bedrock？。

Amazon Comprehend

Amazon Comprehend 是一项托管式自然语言处理（NLP）服务，用于从文档中提取见解。使用 Amazon Comprehend，您可以通过分析实体、关键短语、语言和其他特征，根据文档内容推断情绪。要了解更多信息，请参阅《Amazon Comprehend 开发人员指南》中的什么是 Amazon Comprehend？

SageMaker AI

Amazon SageMaker AI 是一项完全托管的机器学习服务。数据科学家和开发人员使用 Amazon SageMaker AI 构建、训练和测试机器学习模型，以完成各种推理任务，例如欺诈检测和产品推荐。当机器学习模型准备就绪，可用于生产中时，可以将其部署到 Amazon SageMaker AI 托管环境中。有关更多信息，请参阅《Amazon SageMaker AI 开发人员指南》中的 What Is Amazon SageMaker AI?

与使用 SageMaker AI 相比，将 Amazon Comprehend 与 Aurora 数据库集群结合使用时的初步设置更少。如果您不熟悉 AWS 机器学习，建议您先从探索 Amazon Comprehend 开始。

主题

- 将 Amazon Aurora 机器学习与 Aurora MySQL 结合使用
- 将 Amazon Aurora 机器学习与 Aurora PostgreSQL 结合使用

将 Amazon Aurora 机器学习与 Aurora MySQL 结合使用

通过将 Amazon Aurora 机器学习与您的 Aurora MySQL 数据库集群结合使用，您可以根据需要使用 Amazon Bedrock、Amazon Comprehend 或 Amazon SageMaker AI。这些服务分别支持不同的机器学习使用案例。

目录

- 将 Aurora 机器学习与 Aurora MySQL 结合使用的要求
- 区域和版本可用性
- Aurora 机器学习与 Aurora MySQL 结合使用时支持的功能和限制
- 设置 Aurora MySQL 数据库集群以使用 Aurora 机器学习
  - 设置 Aurora MySQL 数据库集群以使用 Amazon Bedrock
  - 设置 Aurora MySQL 数据库集群以使用 Amazon Comprehend
  - 设置 Aurora MySQL 数据库集群以使用 SageMaker AI
    - 设置 Aurora MySQL 数据库集群以将 Amazon S3 用于 SageMaker AI（可选）
  - 授予数据库用户访问 Aurora 机器学习的权限
    - 授予对 Amazon Bedrock 函数的访问权限
    - 授予对 Amazon Comprehend 函数的访问权限
    - 授予对 SageMaker AI 函数的访问权限
- 将 Amazon Bedrock 与 Aurora MySQL 数据库集群结合使用
- 将 Amazon Comprehend 与 Aurora MySQL 数据库集群结合使用
- 将 SageMaker AI 与 Aurora MySQL 数据库集群结合使用
  - 返回字符串的 SageMaker AI 函数的字符集要求
  - 将数据导出到 Amazon S3 以进行 SageMaker AI 模型训练（高级）
- 将 Aurora 机器学习与 Aurora MySQL 结合使用的性能注意事项
  - 模型和提示
  - 查询缓存
  - Aurora 机器学习函数调用的批处理优化
- 监控 Aurora 机器学习

将 Aurora 机器学习与 Aurora MySQL 结合使用的要求

AWS 机器学习服务是在其自己的生产环境中设置和运行的托管式服务。Aurora 机器学习支持与 Amazon Bedrock、Amazon Comprehend 和 SageMaker AI 相集成。在尝试将 Aurora MySQL 数据库集群设置为使用 Aurora 机器学习之前，请务必了解以下要求和先决条件。

- 机器学习服务必须与您的 Aurora MySQL 数据库集群在相同的 AWS 区域中运行。您不能在不同区域的 Aurora MySQL 数据库集群中使用机器学习服务。
- 如果 Aurora MySQL 数据库集群与 Amazon Bedrock、Amazon Comprehend 或 SageMaker AI 服务位于不同的虚拟公有云（VPC）中，则 VPC 的安全组需要允许与目标 Aurora 机器学习服务的出站连接。有关更多信息，请参阅《Amazon VPC 用户指南》中的使用安全组控制到 AWS 资源的流量。
- 如果您要将 Aurora 机器学习与该集群结合使用，您可以将运行较低版本 Aurora MySQL 的 Aurora 集群升级到支持的更高版本。有关更多信息，请参阅 Amazon Aurora MySQL 的数据库引擎更新。
- Aurora MySQL 数据库集群必须使用自定义数据库集群参数组。在您要使用的每个 Aurora 机器学习服务的设置过程结束时，添加为该服务创建的关联 IAM 角色的 Amazon 资源名称（ARN）。建议您事先为 Aurora MySQL 创建自定义数据库集群参数组，并将您的 Aurora MySQL 数据库集群配置为使用该参数组，以便在设置过程结束时做好修改准备。
- 对于 SageMaker AI：
  - 您要用于推理的机器学习组件必须设置好并准备就绪。在配置 Aurora MySQL 数据库集群的过程中，确保具有 SageMaker AI 端点的 ARN。您团队中的数据科学家可能最有能力使用 SageMaker AI，以准备模型并处理其他此类任务。要开始使用 Amazon SageMaker AI，请参阅 Get Started with Amazon SageMaker AI。有关推理和端点的更多信息，请参阅实时推理。
  - 要将 SageMaker AI 用于您自己的训练数据，您必须设置 Amazon S3 存储桶，作为 Aurora 机器学习的 Aurora MySQL 配置的一部分。为此，您需要遵循与设置 SageMaker AI 集成相同的常规流程。有关此可选设置过程的摘要，请参阅设置 Aurora MySQL 数据库集群以将 Amazon S3 用于 SageMaker AI（可选）。
- 对于 Aurora Global Database，您可以设置要在构成 Aurora Global Database 的所有 AWS 区域中使用的 Aurora 机器学习服务。例如，如果您想将 Aurora 机器学习以及 SageMaker AI 一起用于 Aurora Global Database，则可以对每个 AWS 区域中的每个 Aurora MySQL 数据库集群执行以下操作：
  - 使用相同的 SageMaker AI 训练模型和端点设置 Amazon SageMaker AI 服务。它们也必须使用相同的名称。
  - 创建 IAM 角色，详见设置 Aurora MySQL 数据库集群以使用 Aurora 机器学习。

- 将 IAM 角色的 ARN 添加到每个 AWS 区域中每个 Aurora MySQL 数据库集群的自定义数据库集群参数组中。

这些任务要求在构成 Aurora Global Database 的所有 AWS 区域中，Aurora 机器学习均可用于您的 Aurora MySQL 版本。

区域和版本可用性

功能可用性和支持因每个 Aurora 数据库引擎的特定版本以及 AWS 区域而异。

- 有关适用于 Aurora MySQL 的 Amazon Comprehend 和 Amazon SageMaker AI 的版本和区域可用性的信息，请参阅使用 Aurora MySQL 的 Aurora 机器学习。
- 仅 Aurora MySQL 版本 3.06 及更高版本支持 Amazon Bedrock。

  有关 Amazon Bedrock 的区域可用性的信息，请参阅《Amazon Bedrock 用户指南》中的 Model support by AWS 区域。

Aurora 机器学习与 Aurora MySQL 结合使用时支持的功能和限制

将 Aurora MySQL 与 Aurora 机器学习结合使用时，以下限制适用：

- Aurora 机器学习扩展不支持向量接口。
- 在触发器中使用时，不支持 Aurora 机器学习集成。
- Aurora 机器学习函数与二进制日志（binlog）复制不兼容。
  - 调用 Aurora 机器学习函数时，设置 `--binlog-format=STATEMENT` 引发异常。
  - Aurora 机器学习函数是不确定的，而不确定的存储函数与 binlog 格式不兼容。

  有关更多信息，请参阅 MySQL 文档中的 Binary Logging Formats。
- 如果存储函数调用带有 generated-always 列的表，则不支持此类函数。这适用于任何 Aurora MySQL 存储函数。要了解有关此列类型的更多信息，请参阅 MySQL 文档中的 CREATE TABLE 和生成的列。
- Amazon Bedrock 函数不支持 `RETURNS JSON`。如果需要，您可以使用 `CONVERT` 或 `CAST` 从 `TEXT` 转换为 `JSON`。
- Amazon Bedrock 不支持批量请求。
- Aurora MySQL 通过将 `ContentType` 设置为 `text/csv`，来支持任何可读取和写入逗号分隔值（CSV）格式的 SageMaker AI 端点。这种格式已被以下内置的 SageMaker AI 算法所接受：

- 线性学习器
- Random Cut Forest
- XGBoost

要了解有关这些算法的更多信息，请参阅《Amazon SageMaker AI 开发人员指南》中的 Choose an Algorithm。

设置 Aurora MySQL 数据库集群以使用 Aurora 机器学习

在以下主题中，您可以找到其中每个 Aurora 机器学习服务的单独设置过程。

主题

- 设置 Aurora MySQL 数据库集群以使用 Amazon Bedrock
- 设置 Aurora MySQL 数据库集群以使用 Amazon Comprehend
- 设置 Aurora MySQL 数据库集群以使用 SageMaker AI
  - 设置 Aurora MySQL 数据库集群以将 Amazon S3 用于 SageMaker AI（可选）
- 授予数据库用户访问 Aurora 机器学习的权限
  - 授予对 Amazon Bedrock 函数的访问权限
  - 授予对 Amazon Comprehend 函数的访问权限
  - 授予对 SageMaker AI 函数的访问权限

设置 Aurora MySQL 数据库集群以使用 Amazon Bedrock

Aurora 机器学习依赖于 AWS Identity and Access Management（IAM）角色和策略来允许 Aurora MySQL 数据库集群访问和使用 Amazon Bedrock 服务。以下过程创建 IAM 权限策略和角色，以便您的数据库集群可以与 Amazon Bedrock 集成。

创建 IAM policy

1. 登录 AWS Management Console，然后打开 IAM 控制台：https://console.aws.amazon.com/iam/。
2. 在导航窗格中选择策略。
3. 选择创建策略。
4. 在指定权限页面上，对于选择服务，选择 Bedrock。

将显示 Amazon Bedrock 权限。

5. 展开读取，然后选择 InvokeModel。
6. 对于资源，选择全部。

指定权限页面应与下图类似。



7. 选择下一步。
8. 在审核并创建页面上，输入策略的名称，例如 **BedrockInvokeModel**。
9. 查看您的策略，然后选择创建策略。

接下来，您将创建使用 Amazon Bedrock 权限策略的 IAM 角色。

创建 IAM 角色

1. 登录AWS Management Console，然后通过以下网址打开 IAM 控制台：https://console.aws.amazon.com/iam/。
2. 在导航窗格中选择 Roles。
3. 选择 Create role（创建角色）。
4. 在选择可信实体页面上，对于使用案例，选择 RDS。
5. 选择 RDS - 向数据库添加角色，然后选择下一步。
6. 在添加权限页面上，对于权限策略，选择您创建的 IAM 策略，然后选择下一步。
7. 在命名、查看和创建页面上，输入角色的名称，例如 **`ams-bedrock-invoke-model-role`**。

   角色应与下图类似。



8. 检查您的角色，然后选择创建角色。

接下来，将 Amazon Bedrock IAM 角色与数据库集群关联。

将 IAM 角色与您的数据库集群关联

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 从导航窗格中选择 Databases (数据库)。
3. 选择要连接到 Amazon Bedrock 服务的 Aurora MySQL 数据库集群。
4. 选择连接和安全性选项卡。
5. 在管理 IAM 角色部分，选择选择要添加到此集群的 IAM。
6. 选择您创建的 IAM，然后选择添加角色。

   IAM 角色与您的数据库集群相关联，最初状态为待定，然后为活动。该过程完成后，您可以在当前该集群的 IAM 角色列表中找到该角色。



您必须将此 IAM 角色的 ARN 添加到与 Aurora MySQL 数据库集群关联的自定义数据库集群参数组的 `aws_default_bedrock_role` 参数中。如果 Aurora MySQL 数据库集群不使用自定义数据库集群参数组，则需要创建一个与 Aurora MySQL 数据库集群结合使用的此类参数组，以完成集成。有关更多信息，请参阅 Amazon Aurora 数据库集群的数据库集群参数组。

配置数据库集群参数

1. 在 Amazon RDS 控制台中，打开 Aurora MySQL 数据库集群的 Configuration（配置）选项卡。
2. 找到为集群配置的数据库集群参数组。选择链接以打开自定义数据库集群参数组，然后选择编辑。
3. 在自定义数据库集群参数组中找到 `aws_default_bedrock_role` 参数。
4. 在值字段中，输入 IAM 角色的 ARN。
5. 选择 Save changes（保存更改）以保存设置。
6. 重启 Aurora MySQL 数据库集群的主实例，以使此参数设置生效。

Amazon Bedrock 的 IAM 集成已完成。按照授予数据库用户访问 Aurora 机器学习的权限所述，继续设置 Aurora MySQL 数据库集群以与 Amazon Bedrock 结合使用。

设置 Aurora MySQL 数据库集群以使用 Amazon Comprehend

Aurora 机器学习依赖于 AWS Identity and Access Management 角色和策略来允许 Aurora MySQL 数据库集群访问和使用 Amazon Comprehend 服务。以下过程会自动为您的集群创建 IAM 角色和策略，以便它可以使用 Amazon Comprehend。

设置 Aurora MySQL 数据库集群以使用 Amazon Comprehend

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 从导航窗格中选择 Databases (数据库)。
3. 选择要连接到 Amazon Comprehend 服务的 Aurora MySQL 数据库集群。
4. 选择连接和安全性选项卡。
5. 对于管理 IAM 角色部分，选择选择一项服务连接到此集群。
6. 从菜单中选择 Amazon Comprehend，然后选择连接服务。



7. Connect cluster to Amazon Comprehend（将集群连接到 Amazon Comprehend）对话框不需要任何其他信息。但是，您可能会看到一条消息，通知您 Aurora 和 Amazon Comprehend 之间的集成目前处于预览阶段。请务必阅读该消息，然后再继续。如果您不想继续，可以选择取消。
8. 选择 Connect service（连接服务）以完成集成过程。

   Aurora 创建 IAM 角色。它还创建允许 Aurora MySQL 数据库集群使用 Amazon Comprehend 服务的策略，并将该策略附加到该角色。该过程完成后，您可以在 Current IAM roles for this cluster（此集群的当前 IAM 角色）列表中找到该角色，如下图所示。



您需要将此 IAM 角色的 ARN 添加到与 Aurora MySQL 数据库集群关联的自定义数据库集群参数组的 `aws_default_comprehend_role` 参数中。如果 Aurora MySQL 数据库集群不使用自定义数据库集群参数组，则需要创建一个与 Aurora MySQL 数据库集群结合使用的此类参数组，以完成集成。有关更多信息，请参阅 Amazon Aurora 数据库集群的数据库集群参数组。

创建自定义数据库集群参数组并将其与 Aurora MySQL 数据库集群关联后，您可以继续执行以下步骤。

如果集群使用自定义数据库集群参数组，请执行以下操作。

a. 在 Amazon RDS 控制台中，打开 Aurora MySQL 数据库集群的 Configuration（配置）选项卡。

b. 找到为集群配置的数据库集群参数组。选择链接以打开自定义数据库集群参数组，然后选择编辑。

c. 在自定义数据库集群参数组中找到 `aws_default_comprehend_role` 参数。

d. 在值字段中，输入 IAM 角色的 ARN。

e. 选择 Save changes（保存更改）以保存设置。在下图中，您可以找到一个示例。



重启 Aurora MySQL 数据库集群的主实例，以使此参数设置生效。

Amazon Comprehend 的 IAM 集成已完成。通过向相应的数据库用户授予访问权限，继续设置 Aurora MySQL 数据库集群以与 Amazon Comprehend 结合使用。

设置 Aurora MySQL 数据库集群以使用 SageMaker AI

以下过程会自动为 Aurora MySQL 数据库集群创建 IAM 角色和策略，以便它可以使用 SageMaker AI。在尝试执行此过程之前，请确保您有 SageMaker AI 端点可用，以便在需要时输入该端点。通常，团队中的数据科学家会努力生成一个可以从 Aurora MySQL 数据库集群使用的端点。您可以在 SageMaker AI 控制台中找到这样的端点。在导航窗格中，打开 Inference（推理）菜单并选择 Endpoints（端点）。在下图中，您可以找到一个示例。



设置 Aurora MySQL 数据库集群以使用 SageMaker AI

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 从 Amazon RDS 导航菜单中选择数据库，然后选择要连接到 SageMaker AI 服务的 Aurora MySQL 数据库集群。
3. 选择连接和安全性选项卡。
4. 滚动至 Manage IAM roles（管理 IAM 角色）部分，然后选择 Select a service to connect to this cluster（选择一个服务以连接到此集群）。从选择器中选择 SageMaker AI。



5. 选择连接服务。
6. 在将集群连接到 SageMaker AI 对话框中，输入 SageMaker AI 端点的 ARN。



7. Aurora 创建 IAM 角色。它还创建允许 Aurora MySQL 数据库集群使用 SageMaker AI 服务的策略，并将该策略附加到角色。该过程完成后，您可以在当前该集群的 IAM 角色列表中找到该角色。
8. 通过 https://console.aws.amazon.com/iam/ 打开 IAM 控制台。
9. 从 AWS Identity and Access Management 导航菜单的 Access management（访问管理）部分中选择 Roles（角色）。
10. 从列出的角色中找到该角色。其名称使用以下模式。

```
rds-sagemaker-your-cluster-name-role-auto-generated-digits
```

11. 打开角色的 Summary（摘要）页面并找到 ARN。记下 ARN 或使用复制小组件复制它。
12. 通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
13. 选择您的 Aurora MySQL 数据库集群，然后选择其 Configuration（配置）选项卡。
14. 找到数据库集群参数组，然后选择链接以打开自定义数据库集群参数组。找到 `aws_default_sagemaker_role` 参数并在 Value（值）字段中输入 IAM 角色的 ARN，然后保存设置。
15. 重启 Aurora MySQL 数据库集群的主实例，以使此参数设置生效。

IAM 设置现已完成。通过向相应的数据库用户授予访问权限，继续设置 Aurora MySQL 数据库集群以与 SageMaker AI 结合使用。

如果您想使用您的 SageMaker AI 模型进行训练，而不是使用预构建的 SageMaker AI 组件，则还需要将 Amazon S3 存储桶添加到您的 Aurora MySQL 数据库集群中，如下文的 设置 Aurora MySQL 数据库集群以将 Amazon S3 用于 SageMaker AI（可选） 中所述。

设置 Aurora MySQL 数据库集群以将 Amazon S3 用于 SageMaker AI（可选）

要将 SageMaker AI 用于您自己的模型，而不是使用 SageMaker AI 提供的预构建组件，您需要为 Aurora MySQL 数据库集群设置一个 Amazon S3 存储桶以供使用。有关创建 Amazon S3 存储桶的更多信息，请参阅 Amazon Simple Storage Service 用户指南中的创建存储桶。

设置 Aurora MySQL 数据库集群以将 Amazon S3 存储桶用于 SageMaker AI

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 从 Amazon RDS 导航菜单中选择数据库，然后选择要连接到 SageMaker AI 服务的 Aurora MySQL 数据库集群。
3. 选择连接和安全性选项卡。
4. 滚动至 Manage IAM roles（管理 IAM 角色）部分，然后选择 Select a service to connect to this cluster（选择一个服务以连接到此集群）。从选择器中选择 Amazon S3。
5. 选择连接服务。
6. 在将集群连接到 Amazon S3 对话框中，输入 Amazon S3 存储桶的 ARN，如下图所示。



7. 选择 Connect service（连接服务）以完成此过程。

有关将 Amazon S3 存储桶与 SageMaker AI 结合使用的更多信息，请参阅《Amazon SageMaker AI 开发人员指南》中的 Specify an Amazon S3 Bucket to Upload Training Datasets and Store Output Data。要了解有关使用 SageMaker AI 的更多信息，请参阅《Amazon SageMaker AI 开发人员指南》中的 Get Started with Amazon SageMaker AI Notebook Instances。

授予数据库用户访问 Aurora 机器学习的权限

必须向数据库用户授予调用 Aurora 机器学习函数的权限。授予权限的方式取决于您用于 Aurora MySQL 数据库集群的 MySQL 版本，如下所述。如何操作取决于您的 Aurora MySQL 数据库集群使用的 MySQL 版本。

- 对于 Aurora MySQL 版本 3（与 MySQL 8.0 兼容），必须向数据库用户授予相应的数据库角色。有关更多信息，请参阅《MySQL 8.0 参考手册》中的 Using Roles。
- 对于 Aurora MySQL 版本 2（与 MySQL 5.7 兼容），向数据库用户授予权限。有关更多信息，请参阅《MySQL 5.7 参考手册》中的 Access Control and Account Management。

下表显示了数据库用户使用机器学习函数所需的角色和权限。

Aurora MySQL 版本 3（角色）	Aurora MySQL 版本 2（权限）
AWS_BEDROCK_ACCESS	–
AWS_COMPREHEND_ACCESS	INVOKE COMPREHEND

Aurora MySQL 版本 3（角色）	Aurora MySQL 版本 2（权限）
AWS_SAGEMAKER_ACCESS	INVOKE SAGEMAKER

授予对 Amazon Bedrock 函数的访问权限

要向数据库用户授予对 Amazon Bedrock 函数的访问权限，请使用以下 SQL 语句：

```
GRANT AWS_BEDROCK_ACCESS TO user@domain-or-ip-address;
```

对于您为使用 Amazon Bedrock 而创建的函数，还需要向数据库用户授予 EXECUTE 权限。

```
GRANT EXECUTE ON FUNCTION database_name.function_name TO user@domain-or-ip-address;
```

最后，数据库用户必须将其角色设置为 AWS_BEDROCK_ACCESS：

```
SET ROLE AWS_BEDROCK_ACCESS;
```

Amazon Bedrock 函数现已可供使用。

授予对 Amazon Comprehend 函数的访问权限

要向数据库用户授予对 Amazon Comprehend 函数的访问权限，请使用适用于您的 Aurora MySQL 版本的相应语句。

- Aurora MySQL 版本 3（与 MySQL 8.0 兼容）

  ```
  GRANT AWS_COMPREHEND_ACCESS TO user@domain-or-ip-address;
  ```

- Aurora MySQL 版本 2（与 MySQL 5.7 兼容）

  ```
  GRANT INVOKE COMPREHEND ON *.* TO user@domain-or-ip-address;
  ```

Amazon Comprehend 函数现已可供使用。有关使用示例，请参阅 将 Amazon Comprehend 与 Aurora MySQL 数据库集群结合使用。

授予对 SageMaker AI 函数的访问权限

要向数据库用户授予对 SageMaker AI 函数的访问权限，请使用适用于您的 Aurora MySQL 版本的相应语句。

- Aurora MySQL 版本 3（与 MySQL 8.0 兼容）

```
GRANT AWS_SAGEMAKER_ACCESS TO user@domain-or-ip-address;
```

- Aurora MySQL 版本 2（与 MySQL 5.7 兼容）

```
GRANT INVOKE SAGEMAKER ON *.* TO user@domain-or-ip-address;
```

对于您为使用 SageMaker AI 而创建的函数，还需要向数据库用户授予 EXECUTE 权限。假设您创建了两个函数 db1.anomoly_score 和 db2.company_forecasts 以调用 SageMaker AI 端点的服务。您应授予执行权限，如以下示例所示。

```
GRANT EXECUTE ON FUNCTION db1.anomaly_score TO user1@domain-or-ip-address1;
GRANT EXECUTE ON FUNCTION db2.company_forecasts TO user2@domain-or-ip-address2;
```

SageMaker AI 函数现已可供使用。有关使用示例，请参阅 将 SageMaker AI 与 Aurora MySQL 数据库集群结合使用。

将 Amazon Bedrock 与 Aurora MySQL 数据库集群结合使用

要使用 Amazon Bedrock，您需要在 Aurora MySQL 数据库中创建一个调用模型的用户定义函数（UDF）。有关更多信息，请参阅《Amazon Bedrock 用户指南》中的 Supported models in Amazon Bedrock。

UDF 使用以下语法：

```
CREATE FUNCTION function_name (argument type)
        [DEFINER = user]
    RETURNS mysql_data_type
    [SQL SECURITY {DEFINER | INVOKER}]
    ALIAS AWS_BEDROCK_INVOKE_MODEL
    MODEL ID 'model_id'
    [CONTENT_TYPE 'content_type']
    [ACCEPT 'content_type']
    [TIMEOUT_MS timeout_in_milliseconds];
```

- Amazon Bedrock 函数不支持 RETURNS JSON。如果需要，您可以使用 CONVERT 或 CAST 从 TEXT 转换为 JSON。
- 如果您未指定 CONTENT_TYPE 或 ACCEPT，则默认值为 application/json。
- 如果您未指定 TIMEOUT_MS，则使用 aurora_ml_inference_timeout 的值。

例如，以下 UDF 调用 Amazon Titan Text Express 模型：

```
CREATE FUNCTION invoke_titan (request_body TEXT)
    RETURNS TEXT
    ALIAS AWS_BEDROCK_INVOKE_MODEL
    MODEL ID 'amazon.titan-text-express-v1'
    CONTENT_TYPE 'application/json'
    ACCEPT 'application/json';
```

要允许数据库用户使用此函数，请使用以下 SQL 命令：

```
GRANT EXECUTE ON FUNCTION database_name.invoke_titan TO user@domain-or-ip-address;
```

然后，用户会像调用任何其它函数一样调用 invoke_titan，如以下示例所示。请务必根据 Amazon Titan 文本模型格式化请求正文。

```
CREATE TABLE prompts (request varchar(1024));
INSERT INTO prompts VALUES (
'{
    "inputText": "Generate synthetic data for daily product sales in various categories
 - include row number, product name, category, date of sale and price. Produce output
 in JSON format. Count records and ensure there are no more than 5.",
    "textGenerationConfig": {
        "maxTokenCount": 1024,
        "stopSequences": [],
        "temperature":0,
        "topP":1
    }
}');

SELECT invoke_titan(request) FROM prompts;

{"inputTextTokenCount":44,"results":[{"tokenCount":296,"outputText":"
```tabular-data-json
{
```
```

```
    "rows": [
        {
            "Row Number": "1",
            "Product Name": "T-Shirt",
            "Category": "Clothing",
            "Date of Sale": "2024-01-01",
            "Price": "$20"
        },
        {
            "Row Number": "2",
            "Product Name": "Jeans",
            "Category": "Clothing",
            "Date of Sale": "2024-01-02",
            "Price": "$30"
        },
        {
            "Row Number": "3",
            "Product Name": "Hat",
            "Category": "Accessories",
            "Date of Sale": "2024-01-03",
            "Price": "$15"
        },
        {
            "Row Number": "4",
            "Product Name": "Watch",
            "Category": "Accessories",
            "Date of Sale": "2024-01-04",
            "Price": "$40"
        },
        {
            "Row Number": "5",
            "Product Name": "Phone Case",
            "Category": "Accessories",
            "Date of Sale": "2024-01-05",
            "Price": "$25"
        }
    ]
}
```","completionReason":"FINISH"}]}
```

对于您使用的其它模型，请确保相应地为其格式化请求正文。有关更多信息，请参阅《Amazon Bedrock 用户指南》中的 Inference parameters for foundation models。
```

将 Amazon Comprehend 与 Aurora MySQL 数据库集群结合使用

对于 Aurora MySQL，Aurora 机器学习提供了以下两个内置函数，用于处理 Amazon Comprehend 和您的文本数据。您提供要分析的文本（`input_data`）并指定语言（`language_code`）。

aws_comprehend_detect_sentiment

此函数将文本识别为具有积极、消极、中立或混合的情绪姿态。此函数的参考文档如下所示。

```
aws_comprehend_detect_sentiment(
  input_text,
  language_code
  [,max_batch_size]
)
```

要了解更多信息，请参阅《Amazon Comprehend 开发人员指南》中的情绪。

aws_comprehend_detect_sentiment_confidence

此函数衡量针对给定文本检测到的情绪的置信度。它返回一个值（类型为 `double`），该值表示 aws_comprehend_detect_sentiment 函数分配给文本的情绪的置信度。置信度是介于 0 和 1 之间的统计指标。置信度越高，可给予结果的权重越大。该函数的文档摘要如下所示。

```
aws_comprehend_detect_sentiment_confidence(
  input_text,
  language_code
  [,max_batch_size]
)
```

在两个函数（aws_comprehend_detect_sentiment_confidence、aws_comprehend_detect_sentiment）中，如果未指定值，则 `max_batch_size` 使用默认值 25。批处理大小应始终大于 0。您可以使用 `max_batch_size` 优化 Amazon Comprehend 函数调用的性能。较大的批处理大小为了提高 Aurora MySQL 数据库集群上的内存使用率而牺牲更快的性能。有关更多信息，请参阅 将 Aurora 机器学习与 Aurora MySQL 结合使用的性能注意事项。

有关 Amazon Comprehend 中情绪检测函数的参数和返回类型的更多信息，请参阅 DetectSentiment

Example 示例：使用 Amazon Comprehend 函数的简单查询

以下是一个简单查询的示例，它调用了这两个函数，以查看您的客户对您的支持团队的满意程度。假设您有一个数据库表（support），该表存储每次请求帮助后的客户反馈。此示例查询将这两个内置函数应用于表的 feedback 列中的文本，并输出结果。函数返回的置信度值是介于 0.0 和 1.0 之间的双精度（double）值。为了获得更具可读性的输出，此查询将结果四舍五入到 6 个小数点。为了便于比较，此查询还按降序对结果进行排序，首先从置信度最高的结果开始。

```
SELECT feedback AS 'Customer feedback',
    aws_comprehend_detect_sentiment(feedback, 'en') AS Sentiment,
    ROUND(aws_comprehend_detect_sentiment_confidence(feedback, 'en'), 6)
    AS Confidence FROM support
    ORDER BY Confidence DESC;
+-----------------------------------------------------------+-----------+------------+
| Customer feedback                                         | Sentiment | Confidence |
+-----------------------------------------------------------+-----------+------------+
| Thank you for the excellent customer support!             | POSITIVE  |   0.999771 |
| The latest version of this product stinks!                | NEGATIVE  |   0.999184 |
| Your support team is just awesome! I am blown away.       | POSITIVE  |   0.997774 |
| Your product is too complex, but your support is great.   | MIXED     |   0.957958 |
| Your support tech helped me in fifteen minutes.           | POSITIVE  |   0.949491 |
| My problem was never resolved!                            | NEGATIVE  |   0.920644 |
| When will the new version of this product be released?    | NEUTRAL   |   0.902706 |
| I cannot stand that chatbot.                              | NEGATIVE  |   0.895219 |
| Your support tech talked down to me.                      | NEGATIVE  |   0.868598 |
| It took me way too long to get a real person.             | NEGATIVE  |   0.481805 |
+-----------------------------------------------------------+-----------+------------+
 10 rows in set (0.1898 sec)
```

Example 示例：确定高于特定置信度的文本的平均情绪

典型的 Amazon Comprehend 查询会查找情绪为特定值且置信度大于特定数字的行。例如，以下查询显示了如何确定数据库中文档的平均情绪。该查询只考虑评估的置信度至少为 80% 的文档。

```
SELECT AVG(CASE aws_comprehend_detect_sentiment(productTable.document, 'en')
  WHEN 'POSITIVE' THEN 1.0
  WHEN 'NEGATIVE' THEN -1.0
  ELSE 0.0 END) AS avg_sentiment, COUNT(*) AS total
FROM productTable
WHERE productTable.productCode = 1302 AND
    aws_comprehend_detect_sentiment_confidence(productTable.document, 'en') >= 0.80;
```

将 SageMaker AI 与 Aurora MySQL 数据库集群结合使用

要从 Aurora MySQL 数据库集群中使用 SageMaker AI 功能，您需要创建存储函数，以嵌入对 SageMaker AI 端点的调用及其推理功能。为此，您可以使用 MySQL 的 CREATE FUNCTION，其方式与您在 Aurora MySQL 数据库集群上执行其他处理任务的方式大致相同。

要使用 SageMaker AI 中部署的模型进行推理，请使用 MySQL 数据定义语言（DDL）语句为存储函数创建用户定义的函数。每个存储函数表示托管模型的 SageMaker AI 端点。定义此类函数时，请指定模型的输入参数、要调用的特定 SageMaker AI 端点以及返回类型。该函数在对输入参数应用模型后，将返回由 SageMaker AI 端点计算的推理。

所有 Aurora 机器学习存储函数均返回数字类型或 VARCHAR。您可以使用除 BIT 以外的任何数字类型。不允许使用其他类型，例如 JSON、BLOB、TEXT 和 DATE。

以下示例显示与 SageMaker AI 结合使用的 CREATE FUNCTION 语法。

```
CREATE FUNCTION function_name (
     arg1 type1,
     arg2 type2, ...)
     [DEFINER = user]
  RETURNS mysql_type
  [SQL SECURITY { DEFINER | INVOKER } ]
  ALIAS AWS_SAGEMAKER_INVOKE_ENDPOINT
  ENDPOINT NAME 'endpoint_name'
  [MAX_BATCH_SIZE max_batch_size];
```

这是常规 CREATE FUNCTION DDL 语句的扩展。在定义 SageMaker AI 函数的 CREATE FUNCTION 语句中，不要指定函数体。而是要指定函数体通常所在的关键字 ALIAS。目前，Aurora 机器学习只支持 aws_sagemaker_invoke_endpoint 使用这种扩展语法。您还必须指定 endpoint_name 参数。每个模型的 SageMaker AI 端点可以具有不同的特性。

> **Note**
>
> 有关 CREATE FUNCTION 的更多信息，请参阅《MySQL 8.0 参考手册》中的 CREATE PROCEDURE 和 CREATE FUNCTION 语句。

max_batch_size 参数是可选的。默认情况下，最大批处理大小为 10000。您可以在函数中使用此参数来限制针对 SageMaker AI 的批处理请求中处理的最大输入数。max_batch_size 参数可以帮助避免因输入过大而导致的错误，或者使 SageMaker AI 更快地返回响应。此参数影响用于 SageMaker AI

请求处理的内部缓冲区的大小。为 max_batch_size 指定太大的值可能会导致数据库实例产生大量内存开销。

建议您将 MANIFEST 设置保留为其默认值 OFF。尽管可以使用 MANIFEST ON 选项，但某些 SageMaker AI 功能无法直接使用通过此选项导出的 CSV。清单格式与 SageMaker AI 预期的清单格式不兼容。

您为每个 SageMaker AI 模型创建一个单独的存储函数。需要将函数映射到模型，因为终端节点与特定的模型关联，并且每个模型接受的参数不同。将 SQL 类型用于模型输入和模型输出类型有助于避免在 AWS 服务之间来回传递数据时出现类型转换错误。您可以控制谁可以应用模型。也可以通过指定表示最大批处理大小的参数来控制运行时特性。

目前，所有 Aurora 机器学习函数都具有 NOT DETERMINISTIC 属性。如果您未明确指定该属性，Aurora 会自动设置 NOT DETERMINISTIC。提出这一要求是因为可以在不向数据库发出任何通知的情况下更改 SageMaker AI 模型。如果发生这种情况，则调用 Aurora 机器学习函数可能会在单个事务中针对同一输入返回不同的结果。

不能在 CONTAINS SQL 语句中使用特性 NO SQL、READS SQL DATA、MODIFIES SQL DATA 或 CREATE FUNCTION。

下面是调用 SageMaker AI 端点来检测异常的示例用法。这里有一个 SageMaker AI 端点 random-cut-forest-model。random-cut-forest 算法已对相应的模型进行了训练。对于每个输入，模型都会返回一个异常分数。本例显示了分数比平均分数大 3 个标准偏差（大约为 99.9%）的数据点。

```
CREATE FUNCTION anomaly_score(value real) returns real
  alias aws_sagemaker_invoke_endpoint endpoint name 'random-cut-forest-model-demo';

set @score_cutoff = (select avg(anomaly_score(value)) + 3 * std(anomaly_score(value))
 from nyc_taxi);

select *, anomaly_detection(value) score from nyc_taxi
  where anomaly_detection(value) > @score_cutoff;
```

返回字符串的 SageMaker AI 函数的字符集要求

建议为返回字符串值的 SageMaker AI 函数指定一个字符集 utf8mb4 作为返回类型。如果这不切实际，则为返回类型使用足够大的字符串长度，以容纳以 utf8mb4 字符集表示的值。下面的示例展示了如何为您的函数声明 utf8mb4 字符集。

```
CREATE FUNCTION my_ml_func(...) RETURNS VARCHAR(5) CHARSET utf8mb4 ALIAS ...
```

目前，每个返回字符串的 SageMaker AI 函数都使用字符集 utf8mb4 作为返回值。返回值仍使用此字符集，即使您的 SageMaker AI 函数为其返回类型隐式或显式声明了不同的字符集，也是如此。如果 SageMaker AI 函数为返回值声明了不同的字符集，那么如果将返回的数据存储在不够长的表列中，则该数据可能会被无提示截断。例如，带有 DISTINCT 子句的查询将创建一个临时表。因此，由于查询期间在内部处理字符串的方式，SageMaker AI 函数结果可能会被截断。

将数据导出到 Amazon S3 以进行 SageMaker AI 模型训练（高级）

建议您使用提供的一些算法开始使用 Aurora 机器学习和 SageMaker AI，并建议您团队中的数据科学家为您提供可与 SQL 代码结合使用的 SageMaker AI 端点。在下文中，您可以找到有关将自己的 Amazon S3 存储桶与自己的 SageMaker AI 模型和 Aurora MySQL 数据库集群结合使用的极少量信息。

机器学习包括两个主要步骤：训练和推理。要训练 SageMaker AI 模型，请将数据导出到 Amazon S3 存储桶。Jupyter SageMaker AI 笔记本实例使用 Amazon S3 存储桶在部署模型之前对其进行训练。您可以使用 SELECT INTO OUTFILE S3 语句从 Aurora MySQL 数据库集群中查询数据，并将数据直接保存到 Amazon S3 存储桶中存储的文本文件。然后，笔记本实例将使用 Amazon S3 存储桶中的数据进行训练。

Aurora 机器学习将扩展 Aurora MySQL 中现有的 SELECT INTO OUTFILE 语法，以将数据导出为 CSV 格式。需要此格式的模型可以直接使用生成的 CSV 文件进行训练。

```
SELECT * INTO OUTFILE S3 's3_uri' [FORMAT {CSV|TEXT} [HEADER]] FROM table_name;
```

该扩展支持标准 CSV 格式。

- 格式 TEXT 与现有的 MySQL 导出格式相同。这是默认格式。
- 格式 CSV 是一种新引入的格式，遵循 RFC-4180 中的规范。
- 如果指定可选关键字 HEADER，则输出文件包含一个标题行。标题行中的标签与 SELECT 语句中的列名称相对应。
- 您仍可使用关键字 CSV 和 HEADER 作为标识符。

现在，SELECT INTO 的扩展句法和语法如下：

```
INTO OUTFILE S3 's3_uri'
[CHARACTER SET charset_name]
[FORMAT {CSV|TEXT} [HEADER]]
[{FIELDS | COLUMNS}
```

```
  [TERMINATED BY 'string']
  [[OPTIONALLY] ENCLOSED BY 'char']
  [ESCAPED BY 'char']
]
[LINES
  [STARTING BY 'string']
  [TERMINATED BY 'string']
]
```

将 Aurora 机器学习与 Aurora MySQL 结合使用的性能注意事项

Amazon Bedrock、Amazon Comprehend 和 SageMaker AI 服务在由 Aurora 机器学习函数调用时会完成大部分工作。这意味着您可以根据需要独立扩缩这些资源。对于您的 Aurora MySQL 数据库集群，您可以使函数调用尽可能高效。接下来，您可以找到使用 Aurora 机器学习时需要注意的一些性能注意事项。

模型和提示

使用 Amazon Bedrock 时的性能在很大程度上取决于您使用的模型和提示。选择最适合您的使用案例的模型和提示。

查询缓存

Aurora MySQL 查询缓存对 Aurora 机器学习函数无效。对于任何调用 Aurora 机器学习函数的 SQL 语句，Aurora MySQL 不会将查询结果存储到查询缓存中。

Aurora 机器学习函数调用的批处理优化

您可以从 Aurora 集群影响 Aurora 机器学习性能的主要方面是针对 Aurora 机器学习存储函数调用的批处理模式设置。机器学习函数通常需要大量开销，因此，分别为每行内容调用外部服务是不切实际的。Aurora 机器学习可以通过将为多行内容进行的外部 Aurora 机器学习服务调用合并为同一批次，来尽可能减少此类开销。Aurora 机器学习将接收对输入行的所有响应，同时在查询运行时以一次一行的形式将响应传回给查询。这种优化可提高 Aurora 查询的吞吐量并减少延迟，而不会改变结果。

创建连接到 SageMaker AI 端点的 Aurora 存储函数时，需要定义批处理大小参数。该参数影响每个对 SageMaker AI 的基础调用所传输的行数。对于处理大量行的查询，为每一行单独进行 SageMaker AI 调用的开销可能很大。存储过程处理的数据集越大，批处理大小就可以越大。

如果批处理模式优化可以应用于 SageMaker AI 函数，您可以通过查看 `EXPLAIN PLAN` 语句生成的查询计划来确定。在本例中，执行计划中的 `extra` 列包括 `Batched machine learning`。以下示例显示了对使用批处理模式的 SageMaker AI 函数的调用。

```
mysql> CREATE FUNCTION anomaly_score(val real) returns real alias
 aws_sagemaker_invoke_endpoint endpoint name 'my-rcf-model-20191126';
Query OK, 0 rows affected (0.01 sec)

mysql> explain select timestamp, value, anomaly_score(value) from nyc_taxi;
+----+-------------+----------+------------+------+---------------+------+---------
+------+------+----------+-------------------------+
| id | select_type | table    | partitions | type | possible_keys | key  | key_len |
 ref  | rows | filtered | Extra                   |
+----+-------------+----------+------------+------+---------------+------+---------
+------+------+----------+-------------------------+
|  1 | SIMPLE      | nyc_taxi | NULL       | ALL  | NULL          | NULL | NULL    |
 NULL |   48 |   100.00 | Batched machine learning |
+----+-------------+----------+------------+------+---------------+------+---------
+------+------+----------+-------------------------+
1 row in set, 1 warning (0.01 sec)
```

调用内置 Amazon Comprehend 函数之一时，可通过指定可选的 `max_batch_size` 参数来控制批处理大小。该参数限制每个批次中处理的 `input_text` 值的最大数。通过一次发送多个项目，减少了 Aurora 和 Amazon Comprehend 之间的往返次数。在使用 `LIMIT` 子句的查询等情况下，限制批处理大小非常有用。通过使用一个较小的 `max_batch_size` 值，可以避免调用 Amazon Comprehend 的次数超过输入文本的次数。

用于评估 Aurora 机器学习函数的批处理优化适用于以下情况：

- 选择列表或 `SELECT` 语句的 `WHERE` 子句中的函数调用
- `INSERT` 和 `REPLACE` 语句的 `VALUES` 列表中的函数调用
- `UPDATE` 语句的 `SET` 值中的 SageMaker AI 函数：

```
INSERT INTO MY_TABLE (col1, col2, col3) VALUES
  (ML_FUNC(1), ML_FUNC(2), ML_FUNC(3)),
  (ML_FUNC(4), ML_FUNC(5), ML_FUNC(6));
UPDATE MY_TABLE SET col1 = ML_FUNC(col2), SET col3 = ML_FUNC(col4) WHERE ...;
```

监控 Aurora 机器学习

您可以通过查询多个全局变量来监控 Aurora 机器学习批量操作，如以下示例所示。

```
show status like 'Aurora_ml%';
```

可以使用 FLUSH STATUS 语句重置状态变量。因此，自上次重置变量以来，所有数字均表示总计、平均值等。

Aurora_ml_logical_request_cnt

自上次状态重置以来，数据库实例评估的要发送到 Aurora 机器学习服务的逻辑请求数。根据是否使用了批处理，此值可能高于 Aurora_ml_actual_request_cnt。

Aurora_ml_logical_response_cnt

在数据库实例用户运行的所有查询中，Aurora MySQL 从 Aurora 机器学习服务接收的响应次数总计。

Aurora_ml_actual_request_cnt

在数据库实例用户运行的所有查询中，Aurora MySQL 对 Aurora 机器学习服务发出的请求次数总计。

Aurora_ml_actual_response_cnt

在数据库实例用户运行的所有查询中，Aurora MySQL 从 Aurora 机器学习服务接收的响应次数总计。

Aurora_ml_cache_hit_cnt

在数据库实例用户运行的所有查询中，Aurora MySQL 从 Aurora 机器学习服务接收的内部缓存命中次数总计。

Aurora_ml_retry_request_cnt

自上次状态重置以来，数据库实例已向 Aurora 机器学习服务发送的重试请求数。

Aurora_ml_single_request_cnt

在数据库实例用户运行的所有查询中，非批处理模式评估的 Aurora 机器学习函数总计。

有关监控从 Aurora 机器学习函数调用的 SageMaker AI 操作的性能的信息，请参阅 Monitor Amazon SageMaker AI。

将 Amazon Aurora 机器学习与 Aurora PostgreSQL 结合使用

通过将 Amazon Aurora 机器学习与您的 Aurora PostgreSQL 数据库集群结合使用，您可以根据需要使用 Amazon Comprehend、Amazon SageMaker AI 或 Amazon Bedrock。这些服务各自支持特定的机器学习使用案例。

只有某些 AWS 区域 和特定版本的 Aurora PostgreSQL 才支持 Aurora 机器学习。在尝试设置 Aurora 机器学习之前，请检查您的 Aurora PostgreSQL 版本和区域的可用性。有关详细信息，请参阅使用 Aurora PostgreSQL 的 Aurora 机器学习。

主题

- 将 Aurora 机器学习与 Aurora PostgreSQL 结合使用的要求
- Aurora 机器学习与 Aurora PostgreSQL 结合使用时支持的功能和限制
- 设置 Aurora PostgreSQL 数据库集群以使用 Aurora 机器学习
- 将 Amazon Bedrock 与 Aurora PostgreSQL 数据库集群结合使用
- 将 Amazon Comprehend 与 Aurora PostgreSQL 数据库集群结合使用
- 将 SageMaker AI 与 Aurora PostgreSQL 数据库集群结合使用
- 将数据导出到 Amazon S3 以进行 SageMaker AI 模型训练（高级）
- 将 Aurora 机器学习与 Aurora PostgreSQL 结合使用的性能注意事项
- 监控 Aurora 机器学习

将 Aurora 机器学习与 Aurora PostgreSQL 结合使用的要求

AWS 机器学习服务是在其自己的生产环境中设置和运行的托管式服务。Aurora 机器学习支持与 Amazon Comprehend、SageMaker AI 和 Amazon Bedrock 相集成。在尝试将 Aurora PostgreSQL 数据库集群设置为使用 Aurora 机器学习之前，请务必了解以下要求和先决条件。

- Amazon Comprehend、SageMaker AI 和 Amazon Bedrock 服务必须与您的 Aurora PostgreSQL 数据库集群在相同的 AWS 区域中运行。您不能在不同区域的 Aurora PostgreSQL 数据库集群中使用 Amazon Comprehend、SageMaker AI 或 Amazon Bedrock 服务。
- 如果 Aurora PostgreSQL 数据库集群与 Amazon Comprehend 和 SageMaker AI 服务位于不同的基于 Amazon VPC 服务的虚拟公有云（VPC）中，则 VPC 的安全组需要允许与目标 Aurora 机器学习服务的出站连接。有关更多信息，请参阅 启用从 Amazon Aurora 到其它 AWS 服务的网络通信。
- 对于 SageMaker AI，您要用于推理的机器学习组件必须设置好并准备就绪。在配置 Aurora PostgreSQL 数据库集群的过程中，您需要具有 SageMaker AI 端点的 Amazon 资源名称（ARN）。您团队中的数据科学家可能最有能力使用 SageMaker AI，以准备模型并处理其他此类任务。要开始使用 Amazon SageMaker AI，请参阅 Get Started with Amazon SageMaker AI。有关推理和端点的更多信息，请参阅实时推理。
- 对于 Amazon Bedrock，您需要在 Aurora PostgreSQL 数据库集群配置过程中提供要用于推理的 Bedrock 模型的模型 ID。您团队中的数据科学家可能最有能力与 Bedrock 合作，以决定使用哪些模

型，在需要时对其进行微调，并处理其他此类任务。要开始使用 Amazon Bedrock，请参阅 How to setup Bedrock。

- Amazon Bedrock 用户需要先请求模型访问权限，然后才能使用模型。如果您想要添加其他文本、聊天和图像生成模型，则需要在 Amazon Bedrock 中请求模型访问权限。有关更多信息，请参阅模型访问。

Aurora 机器学习与 Aurora PostgreSQL 结合使用时支持的功能和限制

Aurora 机器学习通过 `ContentType` 的 `text/csv` 值支持任何可读取和写入逗号分隔值（CSV）格式的 SageMaker AI 端点。目前接受此格式的内置 SageMaker AI 算法有以下几种。

- 线性学习器
- Random Cut Forest
- XGBoost

  要了解有关这些算法的更多信息，请参阅《Amazon SageMaker AI 开发人员指南》中的 Choose an Algorithm。

将 Amazon Bedrock 与 Aurora 机器学习结合使用时，以下限制适用：

- 用户定义函数（UDF）提供了与 Amazon Bedrock 交互的原生方式。UDF 没有特定的请求或响应要求，因此它们可以使用任何模型。
- 您可以使用 UDF 来构建所需的任何工作流程。例如，您可以组合基本基元（例如 `pg_cron`）来运行查询、提取数据、生成推理以及写入表以直接提供查询。
- UDF 不支持批处理调用或并行调用。
- Aurora 机器学习扩展不支持向量接口。作为扩展的一部分，可以使用函数以 `float8[]` 格式输出模型响应的嵌入内容，以将这些嵌入内容存储在 Aurora 中。有关使用 `float8[]` 的更多信息，请参阅将 Amazon Bedrock 与 Aurora PostgreSQL 数据库集群结合使用。

设置 Aurora PostgreSQL 数据库集群以使用 Aurora 机器学习

要让 Aurora 机器学习与您的 Aurora PostgreSQL 数据库集群结合使用，您需要为您要使用的每项服务创建一个 AWS Identity and Access Management（IAM）角色。IAM 角色允许您的 Aurora PostgreSQL 数据库集群代表集群使用 Aurora 机器学习服务。您还需要安装 Aurora 机器学习扩展。在以下主题中，您可以找到其中每个 Aurora 机器学习服务的设置过程。

主题

- 设置 Aurora PostgreSQL 以使用 Amazon Bedrock
- 设置 Aurora PostgreSQL 以使用 Amazon Comprehend
- 设置 Aurora PostgreSQL 以使用 Amazon SageMaker AI
  - 设置 Aurora PostgreSQL 以将 Amazon S3 用于 SageMaker AI（高级）
- 安装 Aurora 机器学习扩展

设置 Aurora PostgreSQL 以使用 Amazon Bedrock

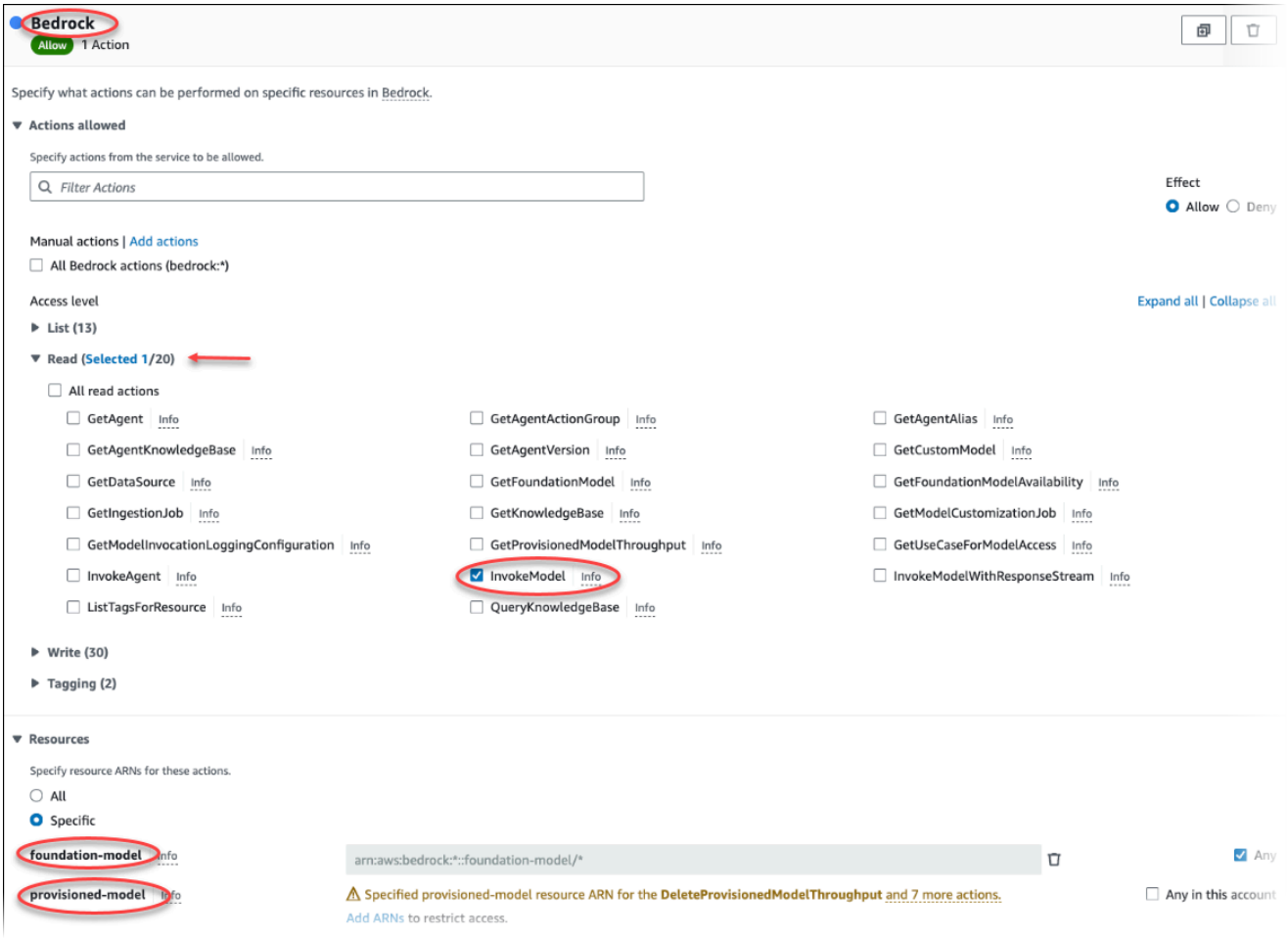
在接下来的过程中，您首先创建 IAM 角色和策略，它们授予 Aurora PostgreSQL 代表集群使用 Amazon Bedrock 的权限。然后，您将策略附加到 Aurora PostgreSQL 数据库集群用来与 Amazon Bedrock 结合使用的 IAM 角色。为简单起见，此过程使用 AWS Management Console 来完成所有任务。

设置 Aurora PostgreSQL 数据库集群以使用 Amazon Bedrock

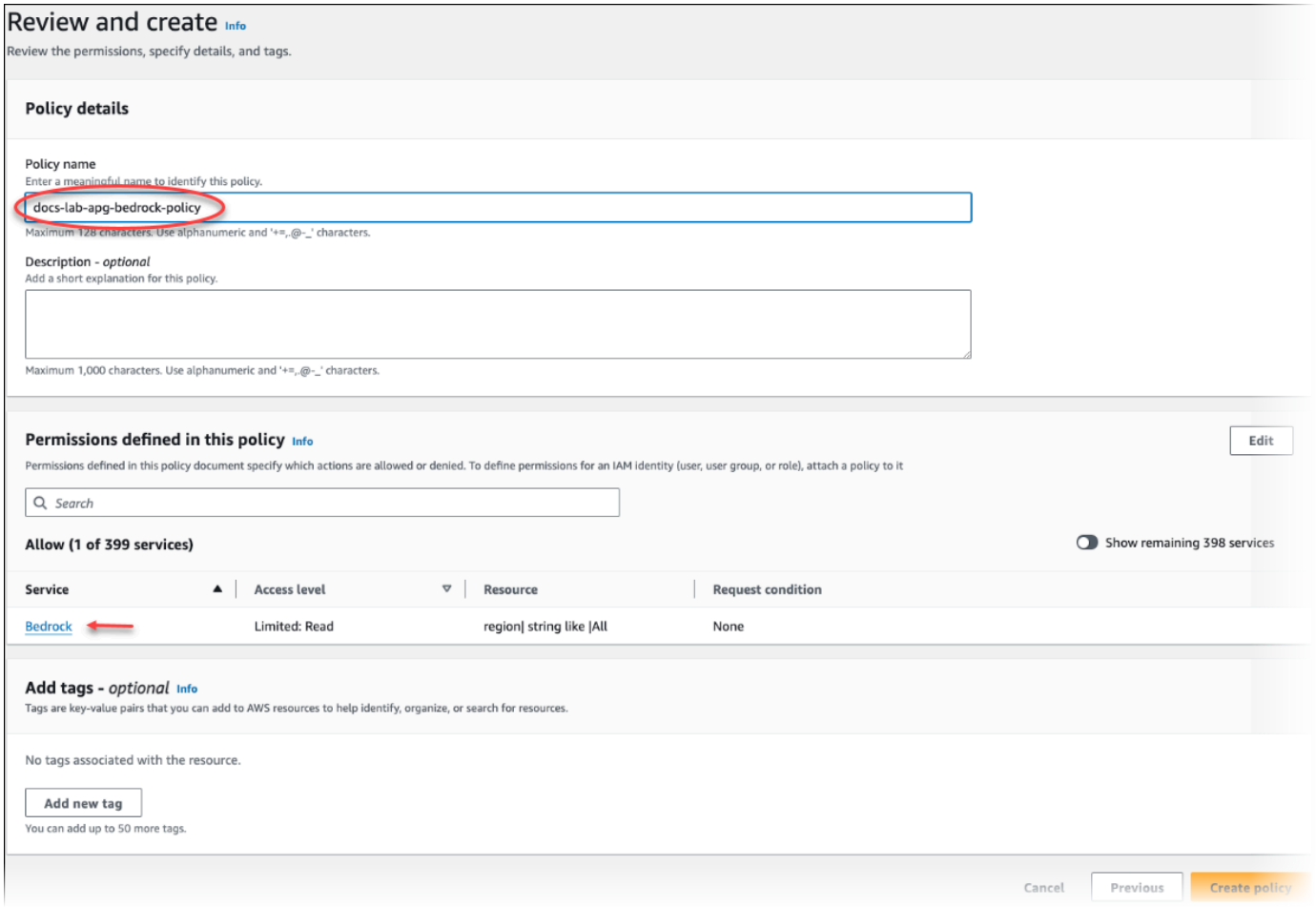
1. 登录 AWS Management Console，然后打开 IAM 控制台：https://console.aws.amazon.com/iam/。
2. 通过 https://console.aws.amazon.com/iam/ 打开 IAM 控制台。
3. 在 AWS Identity and Access Management（IAM）控制台菜单上选择 Policies（策略）[在 Access management（访问管理）下]。

   a. 选择 创建策略。在可视化编辑器页面中，选择服务，然后在“选择服务”字段中输入 Bedrock。展开读取访问权限级别。从 Amazon Bedrock 读取设置中选择 InvokeModel。

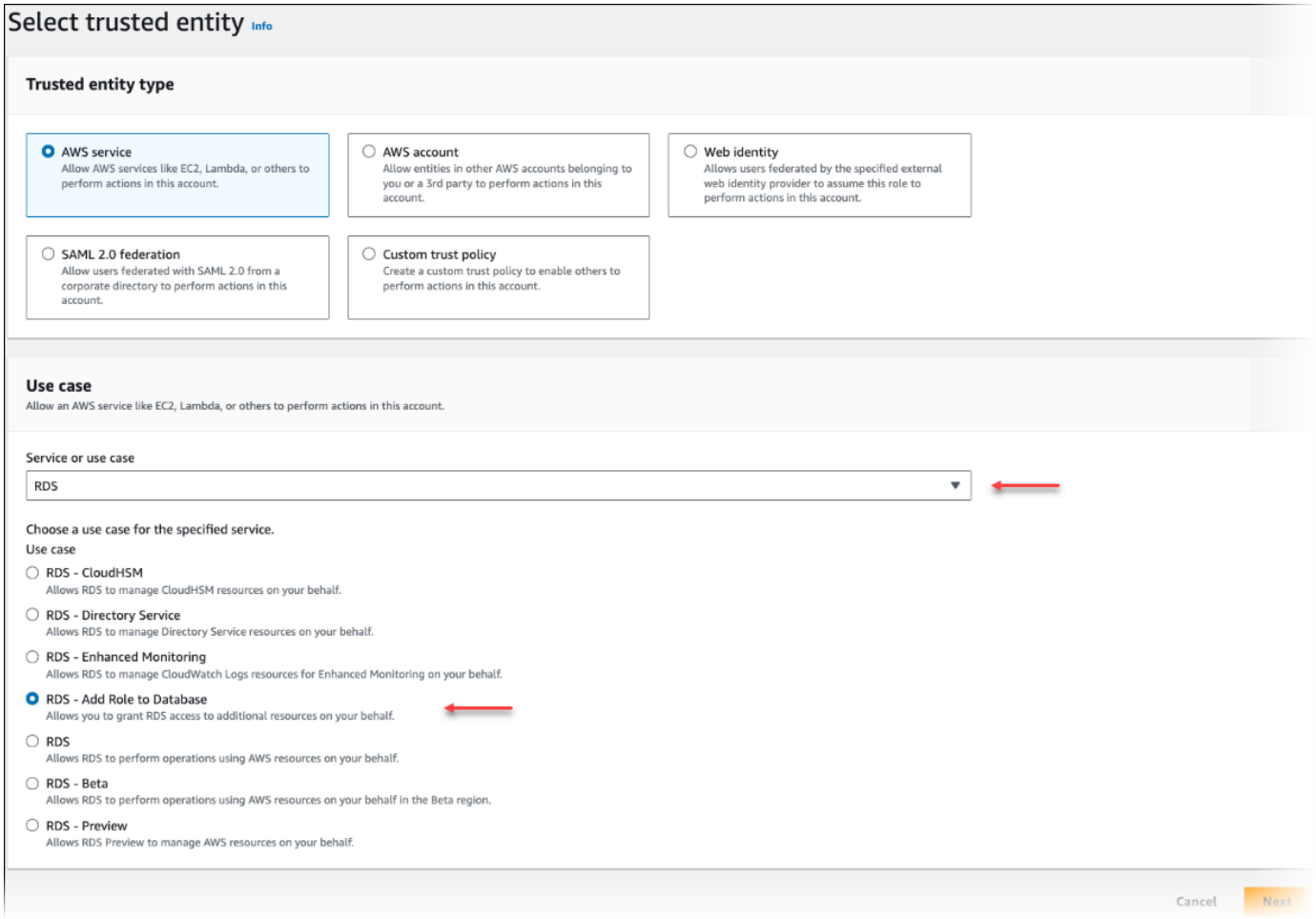
   b. 选择要通过策略授予读取权限的根基/预调配模型。



4. 选择 Next: Tags（下一步：标签）并定义任何标签（这是可选的）。选择下一步：审核。输入策略的名称和说明，如图中所示。

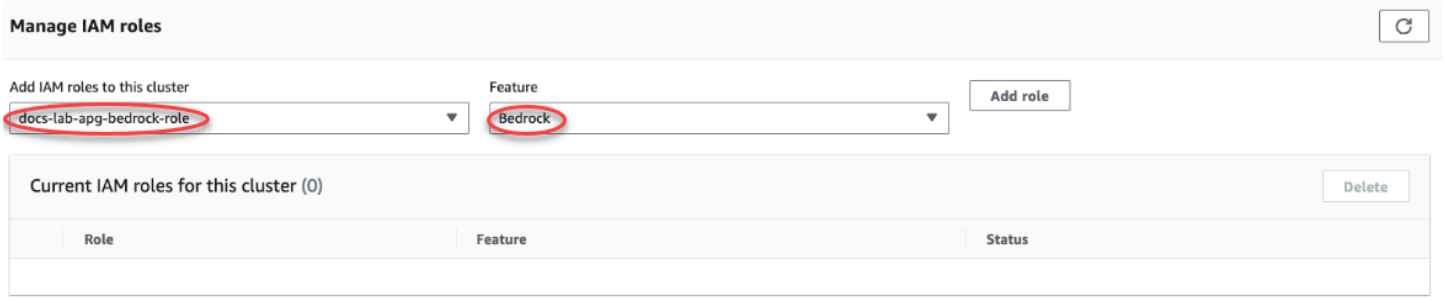


5. 选择 创建策略。保存策略后，控制台会显示提示。您可以在 Policies（策略）列表中找到它。
6. 在 IAM 控制台上选择 Roles（角色）[在 Access management（访问管理）下]。
7. 选择 Create role（创建角色）。
8. 在“选择可信实体”页面上，选择 AWS 服务磁贴，然后选择 RDS 以打开选择器。
9. 选择 RDS – Add Role to Database（RDS – 将角色添加到数据库）。



10. 选择下一步。在 Add permissions（添加权限）页面上，找到您在上一步中创建的策略，并从列出的策略中选择此策略。选择下一步。
11. Next: Review（下一步：审核）。输入 IAM 角色的名称和说明。
12. 通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
13. 导航到 Aurora PostgreSQL 数据库集群所在的 AWS 区域。
14. 在导航窗格中，选择数据库，然后选择要与 Bedrock 结合使用的 Aurora PostgreSQL 数据库集群。
15. 选择 Connectivity & security（连接和安全性）选项卡，然后滚动以找到该页面的 Manage IAM roles（管理 IAM 角色）部分。从 Add IAM roles to this cluster（将 IAM 角色添加到此集群）选择器中，选择您在前面的步骤中创建的角色。在功能选择器中，选择 Bedrock，然后选择添加角色。

该角色（及其策略）与 Aurora PostgreSQL 数据库集群相关联。该过程完成后，该角色将在 Current IAM roles for this cluster（此集群的当前 IAM 角色）列表中列出，如下所示。



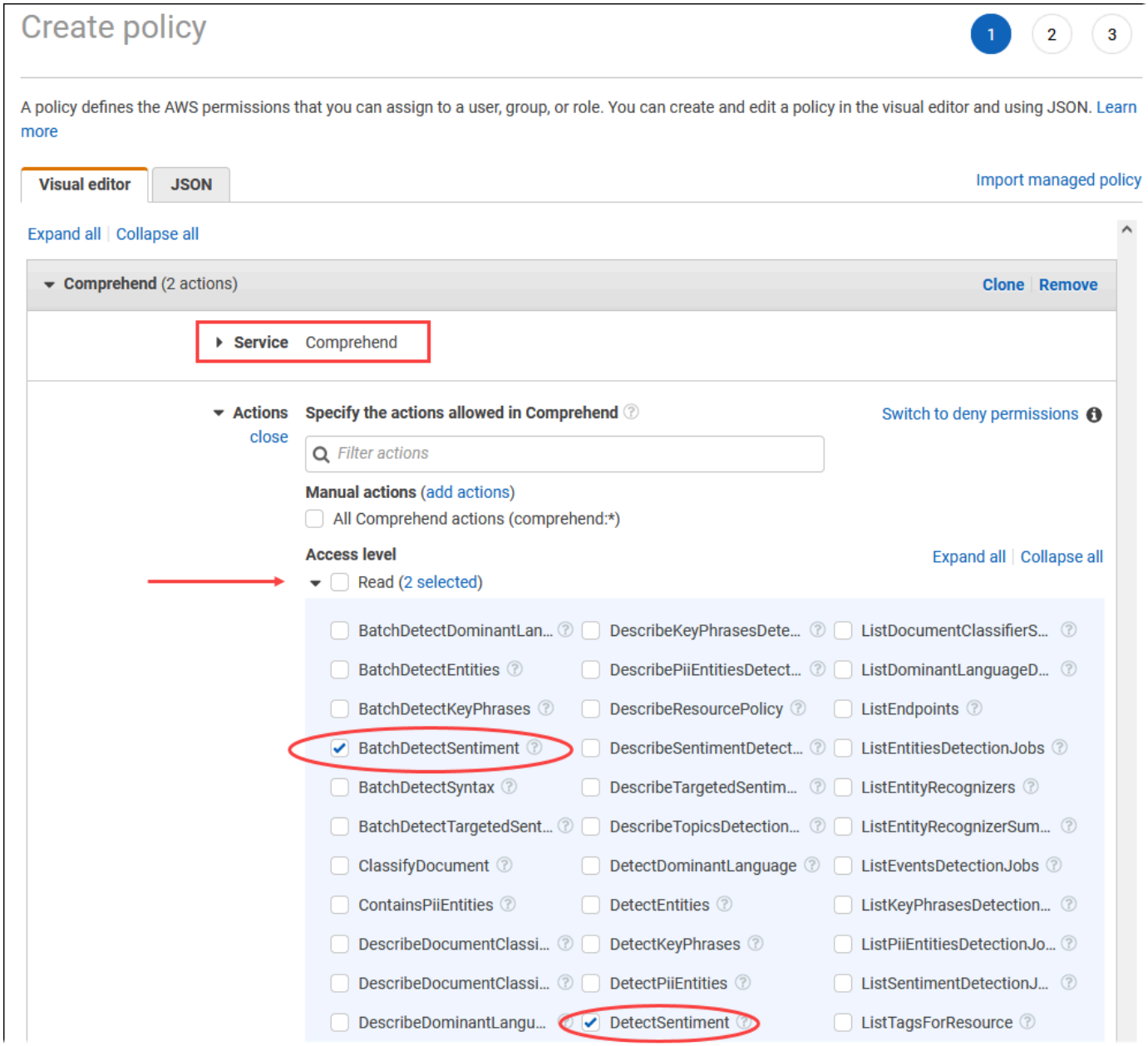
Amazon Bedrock 的 IAM 设置已完成。通过安装扩展继续设置 Aurora PostgreSQL 以使用 Aurora 机器学习，详见 安装 Aurora 机器学习扩展。

设置 Aurora PostgreSQL 以使用 Amazon Comprehend

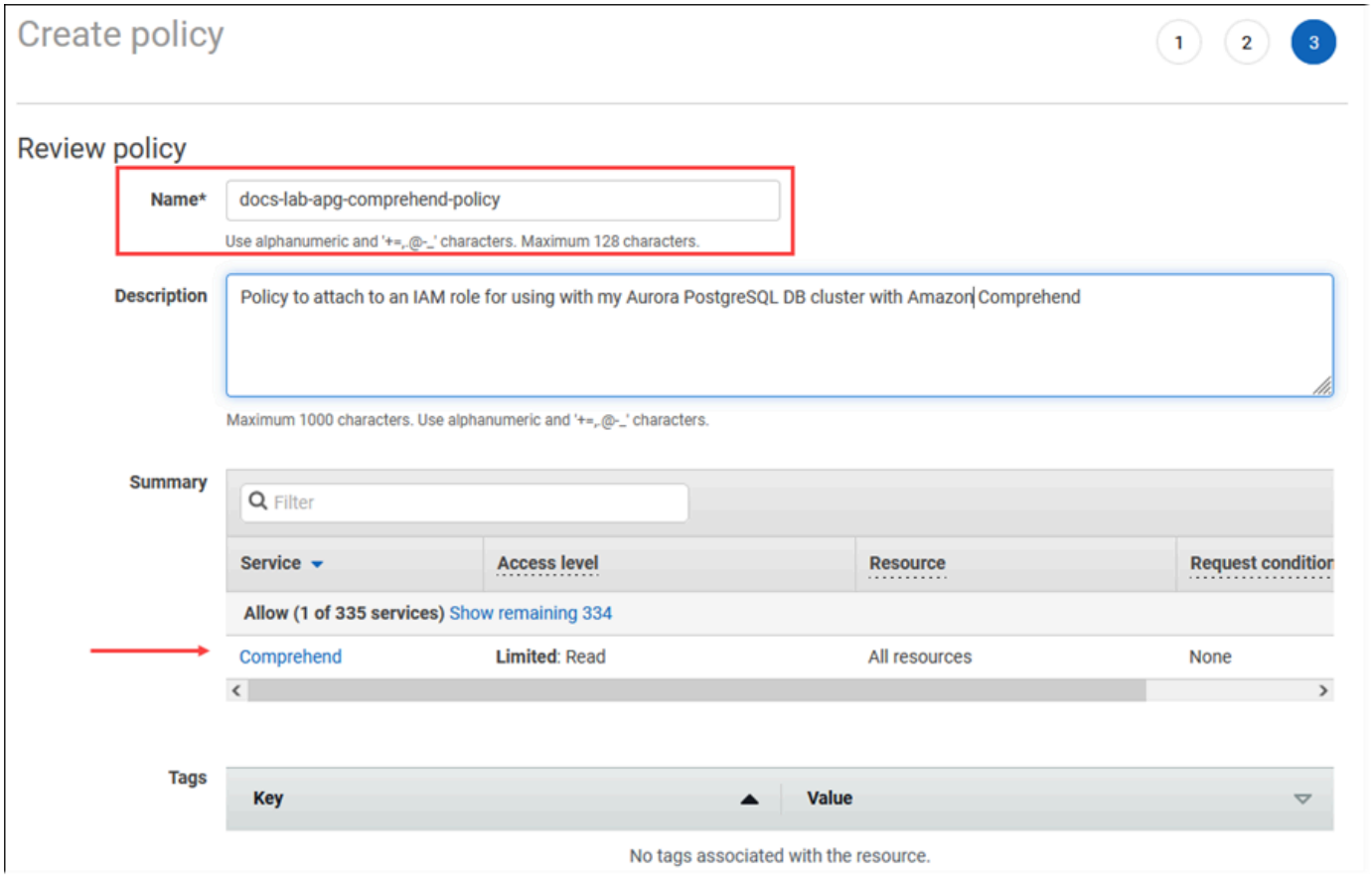
在接下来的过程中，您首先创建 IAM 角色和策略，它们授予 Aurora PostgreSQL 代表集群使用 Amazon Comprehend 的权限。然后，您将策略附加到 Aurora PostgreSQL 数据库集群用来与 Amazon Comprehend 结合使用的 IAM 角色。为简单起见，此过程使用 AWS Management Console来完成所有任务。

设置 Aurora PostgreSQL 数据库集群以使用 Amazon Comprehend

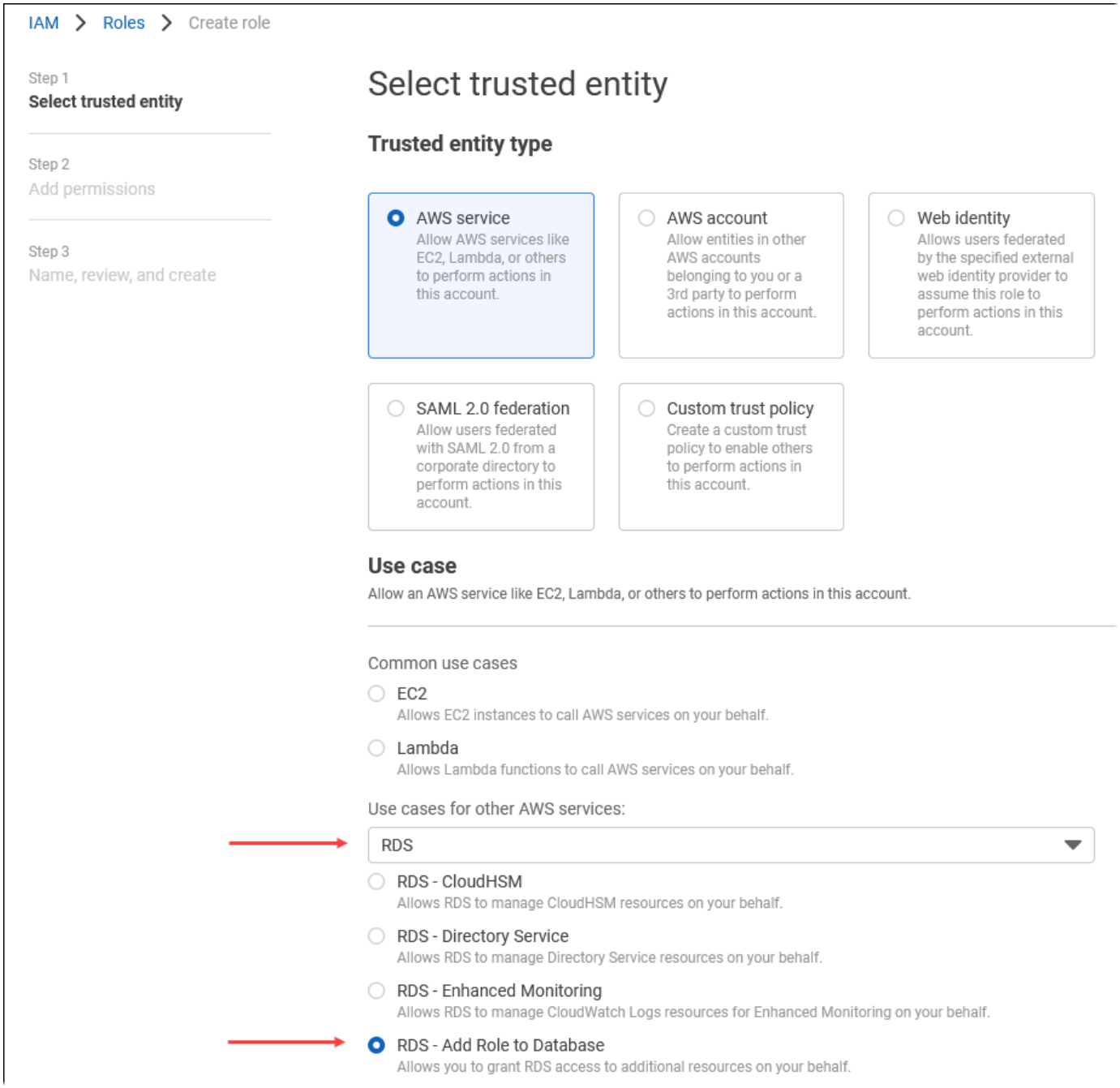
1. 登录 AWS Management Console，然后打开 IAM 控制台：https://console.aws.amazon.com/iam/。
2. 通过 https://console.aws.amazon.com/iam/ 打开 IAM 控制台。
3. 在 AWS Identity and Access Management（IAM）控制台菜单上选择 Policies（策略）[在 Access management（访问管理）下]。



4. 选择 创建策略。在可视化编辑器页面中，选择 Service（服务），然后在 Select a service（选择服务）字段中输入 Comprehend。展开读取访问权限级别。从 Amazon Comprehend 读取设置中选择 BatchDetectSentiment 和 DetectSentiment。
5. 选择 Next: Tags（下一步：标签）并定义任何标签（这是可选的）。选择下一步：审核。输入策略的名称和说明，如图中所示。



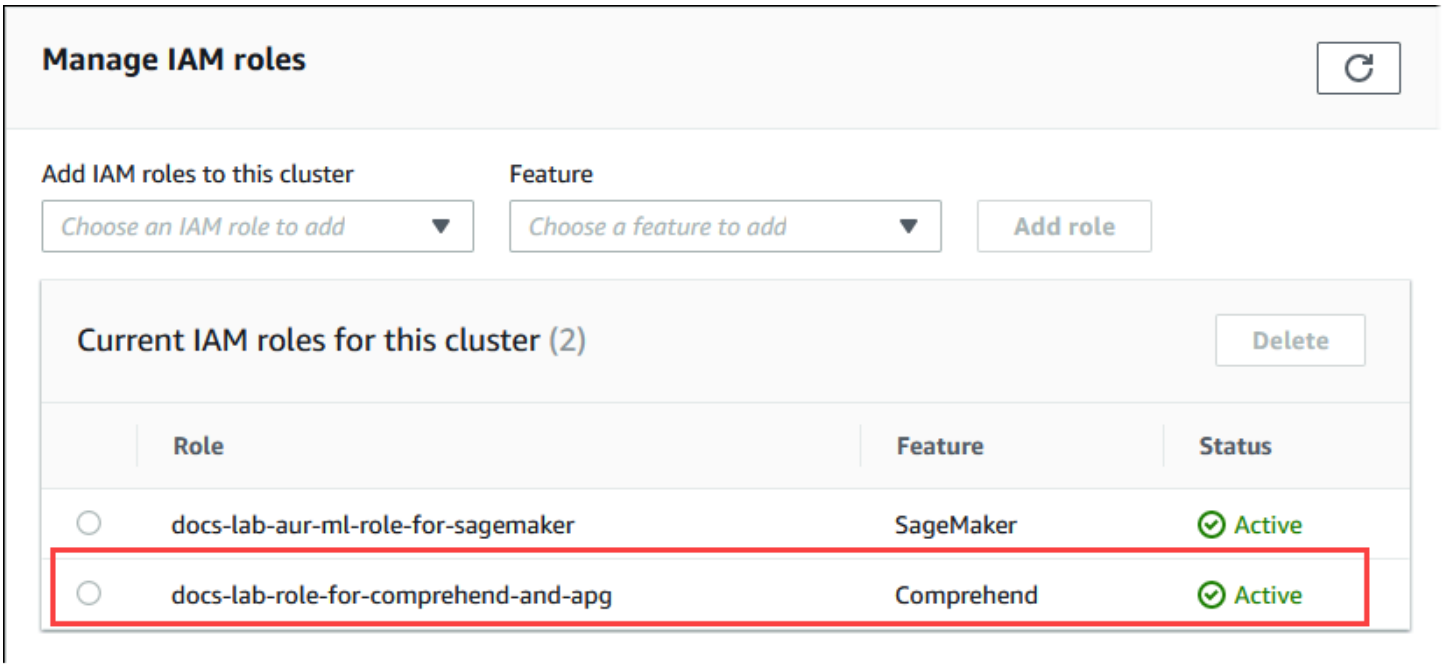
6. 选择 创建策略。保存策略后，控制台会显示提示。您可以在 Policies（策略）列表中找到它。
7. 在 IAM 控制台上选择 Roles（角色）[在 Access management（访问管理）下]。
8. 选择 Create role（创建角色）。
9. 在“选择可信实体”页面上，选择 AWS 服务磁贴，然后选择 RDS 以打开选择器。
10. 选择 RDS – Add Role to Database（RDS – 将角色添加到数据库）。



11. 选择下一步。在 Add permissions（添加权限）页面上，找到您在上一步中创建的策略，并从列出的策略中选择此策略。选择 Next（下一步）
12. Next: Review（下一步：审核）。输入 IAM 角色的名称和说明。
13. 通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
14. 导航到 Aurora PostgreSQL 数据库集群所在的 AWS 区域。
15. 在导航窗格中，选择 Databases（数据库），然后选择要与 Amazon Comprehend 结合使用的 Aurora PostgreSQL 数据库集群。

16. 选择 Connectivity & security（连接和安全性）选项卡，然后滚动以找到该页面的 Manage IAM roles（管理 IAM 角色）部分。从 Add IAM roles to this cluster（将 IAM 角色添加到此集群）选择器中，选择您在前面的步骤中创建的角色。在功能选择器中，选择 Comprehend，然后选择添加角色。

该角色（及其策略）与 Aurora PostgreSQL 数据库集群相关联。该过程完成后，该角色将在 Current IAM roles for this cluster（此集群的当前 IAM 角色）列表中列出，如下所示。



Amazon Comprehend 的 IAM 设置已完成。通过安装扩展继续设置 Aurora PostgreSQL 以使用 Aurora 机器学习，详见 安装 Aurora 机器学习扩展。

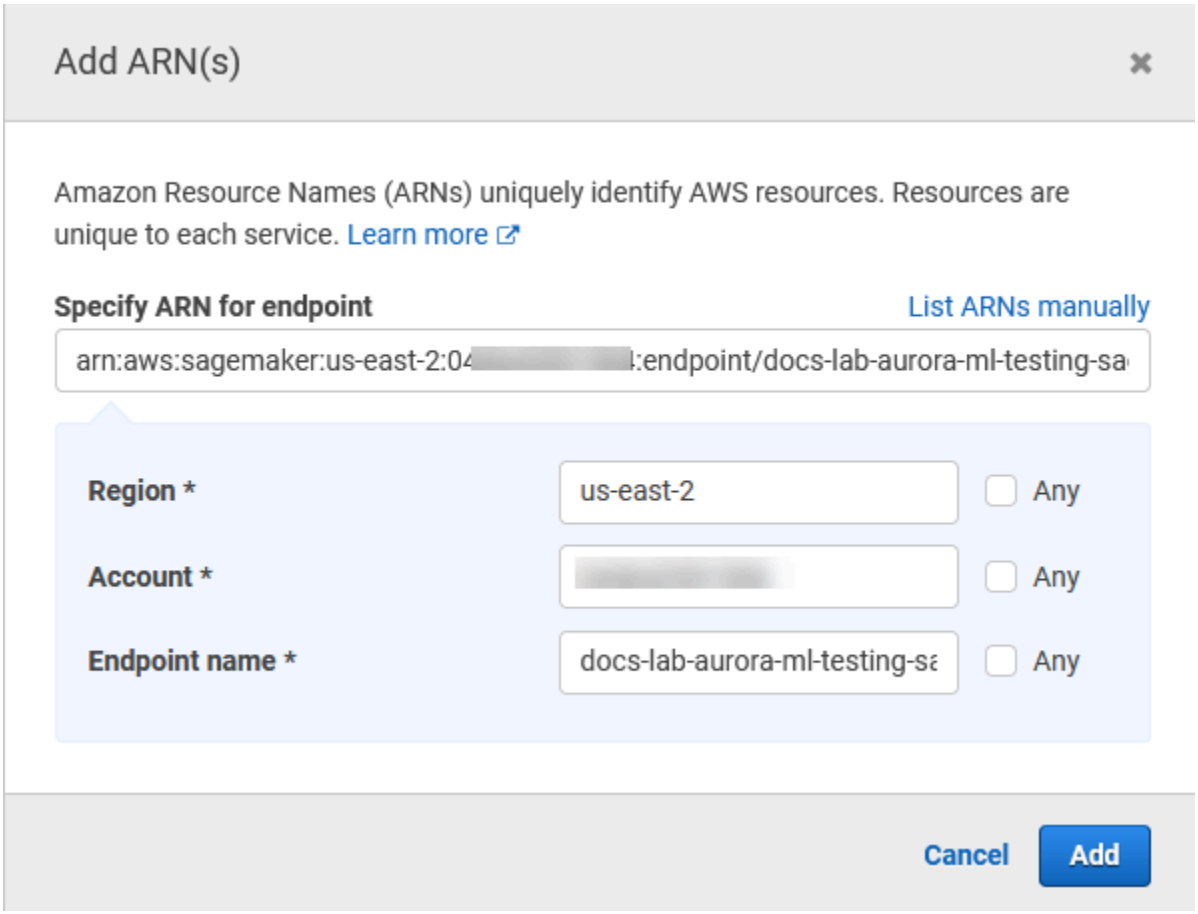
设置 Aurora PostgreSQL 以使用 Amazon SageMaker AI

在为 Aurora PostgreSQL 数据库集群创建 IAM 策略和角色之前，您需要设置 SageMaker AI 模型且端点可用。

设置 Aurora PostgreSQL 数据库集群以使用 SageMaker AI

1. 登录 AWS Management Console，然后打开 IAM 控制台：https://console.aws.amazon.com/iam/。
2. 在 AWS Identity and Access Management（IAM）控制台菜单上选择 Policies（策略）[在 Access management（访问管理）下]，然后选择 Create policy（创建策略）。在可视化编辑器中，对于 Service（服务）选择 SageMaker。对于 Actions（操作），打开 Read（读取）选择器 [在 Access level（访问级别）下]，然后选择 InvokeEndpoint。当您这样做时，会显示一个警告图标。

3. 打开 Resources（资源）选择器，然后在 InvokeEndpoint 操作的 Specify endpoint resource ARN（指定端点资源 ARN）下选择 Add ARN to restrict access（添加 ARN 以限制访问权限）链接。
4. 输入 SageMaker AI 资源的 AWS 区域和端点的名称。您的 AWS 账户已预先填入。



5. 选择 Add（添加）以保存。选择 Next: Tags（下一步：标签）和 Next: Review（下一步：检查），以进入策略创建过程的最后一页。
6. 输入此策略的 Name（名称）和 Description（描述），然后选择 Create policy（创建角色）。策略已创建并添加到 Policies（策略）列表中。发生这种情况时，您会在控制台中看到提示。
7. 在 IAM 控制台中，选择 Roles（角色）。
8. 选择 Create role（创建角色）。
9. 在“选择可信实体”页面上，选择 AWS 服务磁贴，然后选择 RDS 以打开选择器。
10. 选择 RDS – Add Role to Database（RDS – 将角色添加到数据库）。
11. 选择下一步。在 Add permissions（添加权限）页面上，找到您在上一步中创建的策略，并从列出的策略中选择此策略。选择 Next（下一步）
12. Next: Review（下一步：审核）。输入 IAM 角色的名称和说明。

13. 通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
14. 导航到 Aurora PostgreSQL 数据库集群所在的 AWS 区域。
15. 在导航窗格中，选择数据库，然后选择要与 SageMaker AI 结合使用的 Aurora PostgreSQL 数据库集群。
16. 选择 Connectivity & security（连接和安全性）选项卡，然后滚动以找到该页面的 Manage IAM roles（管理 IAM 角色）部分。从 Add IAM roles to this cluster（将 IAM 角色添加到此集群）选择器中，选择您在前面的步骤中创建的角色。在功能选择器中，选择 SageMaker AI，然后选择添加角色。

该角色（及其策略）与 Aurora PostgreSQL 数据库集群相关联。该过程完成后，该角色将在 Current IAM roles for this cluster（此集群的当前 IAM 角色）列表中列出。

SageMaker AI 的 IAM 设置已完成。通过安装扩展继续设置 Aurora PostgreSQL 以使用 Aurora 机器学习，详见 安装 Aurora 机器学习扩展。

设置 Aurora PostgreSQL 以将 Amazon S3 用于 SageMaker AI（高级）

要将 SageMaker AI 用于自己的模型，而不是使用 SageMaker AI 提供的预构建组件，您需要为 Aurora PostgreSQL 数据库集群设置一个 Amazon Simple Storage Service（Amazon S3）存储桶以供使用。这是一个高级主题，本《Amazon Aurora 用户指南》中没有完整记载。一般过程与集成对 SageMaker AI 的支持的过程一样，如下所示。

1. 为 Amazon S3 创建 IAM 策略和角色。
2. 添加 IAM 角色，并在 Aurora PostgreSQL 数据库集群的 Connectivity & security（连接和安全性）选项卡上将 Amazon S3 导入或导出添加为一项功能。
3. 将角色的 ARN 添加到 Aurora 数据库集群的自定义数据库集群参数组中。

有关基本使用信息，请参阅 将数据导出到 Amazon S3 以进行 SageMaker AI 模型训练（高级）。

安装 Aurora 机器学习扩展

Aurora 机器学习扩展 `aws_ml 1.0` 提供了两个函数，您可以使用它们来调用 Amazon Comprehend、SageMaker AI 服务；而 `aws_ml 2.0` 提供了两个其他函数，您可以使用它们来调用 Amazon Bedrock 服务。在 Aurora PostgreSQL 数据库集群上安装这些扩展还会为该功能创建管理角色。

> **Note**
>
> 使用这些函数取决于已完成 Aurora 机器学习服务（Amazon Comprehend、SageMaker AI、Amazon Bedrock）的 IAM 设置，详见设置 Aurora PostgreSQL 数据库集群以使用 Aurora 机器学习。

- aws_comprehend.detect_sentiment – 您可以使用此函数将情感分析应用于存储在 Aurora PostgreSQL 数据库集群上的数据库中的文本。
- aws_sagemaker.invoke_endpoint – 您在 SQL 代码中使用此函数与集群中的 SageMaker AI 端点进行通信。
- aws_bedrock.invoke_model – 您在 SQL 代码中使用此函数与集群中的 Bedrock 模型进行通信。此函数的响应将采用 TEXT 格式，因此，如果模型以 JSON 正文的格式进行响应，则此函数的输出将以字符串的格式传递给最终用户。
- aws_bedrock.invoke_model_get_embeddings – 您可以在 SQL 代码中使用此函数来调用 Bedrock 模型，这些模型在 JSON 响应中返回输出嵌入内容。当您想提取与 json-key 直接关联的嵌入内容时，可以利用这一点来简化任何自行管理的工作流程的响应。

在 Aurora PostgreSQL 数据库集群中安装 Aurora 机器学习扩展

- 使用 psql 连接到 Aurora PostgreSQL 数据库集群的写入器实例。连接到要在其中安装 aws_ml 扩展的特定数据库。

```
psql --host=cluster-instance-1.111122223333.aws-region.rds.amazonaws.com --
port=5432 --username=postgres --password --dbname=labdb
```

```
labdb=> CREATE EXTENSION IF NOT EXISTS aws_ml CASCADE;
NOTICE:  installing required extension "aws_commons"
CREATE EXTENSION
labdb=>
```

安装 aws_ml 扩展还会创建 aws_ml 管理角色和三个新架构，如下所示。

- aws_comprehend – 适用于 Amazon Comprehend 服务和 detect_sentiment 函数（aws_comprehend.detect_sentiment）的来源的模式。

- aws_sagemaker – 适用于 SageMaker AI 服务和 invoke_endpoint 函数（aws_sagemaker.invoke_endpoint）的来源的模式。
- aws_bedrock – 适用于 Amazon Bedrock 服务以及 invoke_model(aws_bedrock.invoke_model) 和 invoke_model_get_embeddings(aws_bedrock.invoke_model_get_embeddings) 函数的来源的架构。

rds_superuser 角色被授予 aws_ml 管理角色，并成为这三个 Aurora 机器学习架构的 OWNER。要允许其他数据库用户访问 Aurora 机器学习函数，rds_superuser 需要授予对 Aurora 机器学习函数的 EXECUTE 权限。默认情况下，对于这两个 Aurora 机器学习模式中的函数，将从 PUBLIC 中撤销 EXECUTE 权限。

在多租户数据库配置中，您可以通过对您要保护的特定 Aurora 机器学习模式使用 REVOKE USAGE，以阻止租户访问 Aurora 机器学习函数。

将 Amazon Bedrock 与 Aurora PostgreSQL 数据库集群结合使用

对于 Aurora PostgreSQL，Aurora 机器学习提供了以下 Amazon Bedrock 函数来处理文本数据。只有在安装 aws_ml 2.0 扩展并完成所有设置过程后，此函数才可用。有关更多信息，请参阅 设置 Aurora PostgreSQL 数据库集群以使用 Aurora 机器学习。

aws_bedrock.invoke_model

此函数采用 JSON 格式的文本作为输入，并针对托管在 Amazon Bedrock 上的各种模型对其进行处理，然后从模型中获取 JSON 文本响应。此响应可能包含文本、图像或嵌入内容。该函数的文档摘要如下所示。

```
aws_bedrock.invoke_model(
   IN model_id      varchar,
   IN content_type  text,
   IN accept_type   text,
   IN model_input   text,
   OUT model_output varchar)
```

此函数的输入和输出如下所示。

- model_id – 模型的标识符。
- content_type – 对 Bedrock 的模型的请求类型。

- accept_type – 期望从 Bedrock 的模型中得到的响应类型。对于大多数模型，通常为应用程序/JSON。
- model_input – 提示；模型的一组特定输入，格式由 content_type 指定。有关模型接受的请求格式/结构的更多信息，请参阅 Inference parameters for foundation models。
- model_output – Bedrock 模型的输出（文本形式）。

以下示例说明如何使用 invoke_model 为 Bedrock 调用 Anthropic Claude 2 模型。

Example 示例：使用 Amazon Bedrock 函数的简单查询

```
SELECT aws_bedrock.invoke_model (
    model_id    := 'anthropic.claude-v2',
    content_type:= 'application/json',
    accept_type := 'application/json',
    model_input := '{"prompt": "\n\nHuman: You are a helpful assistant that answers
 questions directly and only using the information provided in the context below.
\nDescribe the answer
    in detail.\n\nContext: %s \n\nQuestion: %s \n
\nAssistant:","max_tokens_to_sample":4096,"temperature":0.5,"top_k":250,"top_p":0.5,"stop_seque
[]}'
);
```

aws_bedrock.invoke_model_get_embeddings

在某些情况下，模型输出可以指向向量嵌入内容。鉴于每个模型的响应各不相同，可以利用另一个函数 invoke_model_get_embeddings，它的工作原理与 invoke_model 完全一样，但是通过指定相应的 json-key 来输出嵌入内容。

```
aws_bedrock.invoke_model_get_embeddings(
   IN model_id      varchar,
   IN content_type  text,
   IN json_key      text,
   IN model_input   text,
   OUT model_output float8[])
```

此函数的输入和输出如下所示。

- model_id – 模型的标识符。

- content_type – 对 Bedrock 的模型的请求类型。在这里，accept_type 设置为默认值 application/json。
- model_input – 提示；模型的一组特定输入，格式由 content_type 指定。有关模型接受的请求格式/结构的更多信息，请参阅 Inference parameters for foundation models。
- json_key - 对要从中提取嵌入内容的字段的引用。如果嵌入模型发生变化，则可能会有所不同。
- model_output – Bedrock 模型的输出，作为一个具有 16 位小数的嵌入内容数组。

以下示例显示了如何使用 Titan Embeddings G1 - 文本嵌入模型为 PostgreSQL I/O 监控视图短语生成嵌入内容。

Example 示例：使用 Amazon Bedrock 函数的简单查询

```
SELECT aws_bedrock.invoke_model_get_embeddings(
   model_id      := 'amazon.titan-embed-text-v1',
   content_type  := 'application/json',
   json_key      := 'embedding',
   model_input   := '{ "inputText": "PostgreSQL I/O monitoring views"}') AS embedding;
```

将 Amazon Comprehend 与 Aurora PostgreSQL 数据库集群结合使用

对于 Aurora PostgreSQL，Aurora 机器学习提供了以下 Amazon Comprehend 函数来处理文本数据。只有在安装 aws_ml 扩展并完成所有设置过程后，此函数才可用。有关更多信息，请参阅 设置 Aurora PostgreSQL 数据库集群以使用 Aurora 机器学习。

aws_comprehend.detect_sentiment

此函数将文本作为输入，并评估文本是否具有正面、负面、中立或混合的情感姿态。它会输出这种情绪以及其评估的置信度。该函数的文档摘要如下所示。

```
aws_comprehend.detect_sentiment(
   IN input_text varchar,
   IN language_code varchar,
   IN max_rows_per_batch int,
   OUT sentiment varchar,
   OUT confidence real)
```

此函数的输入和输出如下所示。

- input_text – 用于评估和分配情绪（负面、正面、中立、混合）的文本。
- language_code – 使用带有区域子标签（根据需要）的双字母 ISO 639-1 标识符或 ISO 639-2 三字母代码（视情况而定）标识的 input_text 的语言。例如，en 是英语的代码，zh 是简体中文的代码。有关更多信息，请参阅《Amazon Comprehend 开发人员指南》中的支持的语言。
- max_rows_per_batch – 批处理模式处理的每个批处理的最大行数。有关更多信息，请参阅 了解批处理模式和 Aurora 机器学习函数。
- sentiment – 输入文本的情绪，标识为 POSITIVE、NEGATIVE、NEUTRAL 或 MIXED。
- confidence – 指定 sentiment 的准确性的置信度。值范围为 0.0 到 1.0。

在下文中，您可以找到如何使用此函数的示例。

Example 示例：使用 Amazon Comprehend 函数的简单查询

以下是一个简单查询的示例，它调用此函数来评估客户对您的支持团队的满意度。假设您有一个数据库表（support），该表存储每次请求帮助后的客户反馈。此示例查询将 aws_comprehend.detect_sentiment 函数应用于表的 feedback 列中的文本，并输出情绪和该情绪的置信度。此查询还按降序输出结果。

```
SELECT feedback, s.sentiment,s.confidence
    FROM support,aws_comprehend.detect_sentiment(feedback, 'en') s
    ORDER BY s.confidence DESC;
 feedback                          | sentiment | confidence
 ------------------------------------------------------------+-----------+------------
 Thank you for the excellent customer support!              | POSITIVE  |   0.999771
 The latest version of this product stinks!                 | NEGATIVE  |   0.999184
 Your support team is just awesome! I am blown away.        | POSITIVE  |   0.997774
 Your product is too complex, but your support is great.    | MIXED     |   0.957958
 Your support tech helped me in fifteen minutes.            | POSITIVE  |   0.949491
 My problem was never resolved!                             | NEGATIVE  |   0.920644
 When will the new version of this product be released?     | NEUTRAL   |   0.902706
 I cannot stand that chatbot.                               | NEGATIVE  |   0.895219
 Your support tech talked down to me.                       | NEGATIVE  |   0.868598
 It took me way too long to get a real person.              | NEGATIVE  |   0.481805

 (10 rows)
```

要避免针对每个表行支付多次情绪检测费用，您可以对结果进行具体化。队感兴趣的行执行此操作。例如，正在更新临床医生的笔记，以便只有讲法语（fr）的人才能使用情绪检测功能。

```
UPDATE clinician_notes
SET sentiment = (aws_comprehend.detect_sentiment (french_notes, 'fr')).sentiment,
    confidence = (aws_comprehend.detect_sentiment (french_notes, 'fr')).confidence
WHERE
    clinician_notes.french_notes IS NOT NULL AND
    LENGTH(TRIM(clinician_notes.french_notes)) > 0 AND
    clinician_notes.sentiment IS NULL;
```

有关优化函数调用的更多信息，请参阅将 Aurora 机器学习与 Aurora PostgreSQL 结合使用的性能注意事项。

将 SageMaker AI 与 Aurora PostgreSQL 数据库集群结合使用

在如 设置 Aurora PostgreSQL 以使用 Amazon SageMaker AI 中所述设置 SageMaker AI 环境并与 Aurora PostgreSQL 集成后，您可以使用 `aws_sagemaker.invoke_endpoint` 函数调用操作。`aws_sagemaker.invoke_endpoint` 函数仅连接到同一 AWS 区域 中的模型端点。如果您的数据库实例在多个 AWS 区域 中有副本，请确保在每个 AWS 区域 中设置并部署每个 SageMaker AI 模型。

对 `aws_sagemaker.invoke_endpoint` 的调用使用您设置的 IAM 角色进行身份验证，该角色用于将 Aurora PostgreSQL 数据库集群与 SageMaker AI 服务以及您在设置过程中提供的端点相关联。SageMaker AI 模型端点的作用域限于单个账户，并且不是公有的。`endpoint_name` URL 不包含账户 ID。SageMaker AI 通过由数据库实例的 SageMaker AI IAM 角色提供的身份验证令牌确定账户 ID。

aws_sagemaker.invoke_endpoint

此函数将 SageMaker AI 端点作为输入，并将应处理的行数作为一个批次进行处理。它还将 SageMaker AI 模型端点预期的各种参数作为输入。此函数的参考文档如下所示。

```
aws_sagemaker.invoke_endpoint(
  IN endpoint_name varchar,
  IN max_rows_per_batch int,
  VARIADIC model_input "any",
  OUT model_output varchar
  )
```

此函数的输入和输出如下所示。

- endpoint_name – 独立于 AWS 区域 的端点 URL。
- max_rows_per_batch – 批处理模式处理的每个批处理的最大行数。有关更多信息，请参阅 了解批处理模式和 Aurora 机器学习函数。
- model_input – 适用于模型的一个或多个输入参数。它们可以是 SageMaker AI 模型所需的任何数据类型。PostgreSQL 允许您为一个函数指定最多 100 个输入参数。数组数据类型必须是一维的，但可以包含 SageMaker AI 模型期望的尽可能多的元素。SageMaker AI 模型的输入数仅受 SageMaker AI 6 MB 消息大小限制所限。
- model_output – SageMaker AI 模型的输出（文本形式）。

创建用户定义的函数以调用 SageMaker AI 模型

创建单独的用户定义的函数来为每个 SageMaker AI 模型调用 aws_sagemaker.invoke_endpoint。您的用户定义的函数表示托管模型的 SageMaker AI 端点。aws_sagemaker.invoke_endpoint 函数在用户定义的函数中运行。用户定义的函数提供了许多好处：

- 可以为 SageMaker AI 模型指定其自己的名称，而不是仅为所有 SageMaker AI 模型调用 aws_sagemaker.invoke_endpoint。
- 可以仅在 SQL 应用程序代码中的一个位置指定模型端点 URL。
- 可以单独控制每个 Aurora 机器学习函数的 EXECUTE 权限。
- 可以使用 SQL 类型声明模型输入和输出类型。SQL 强制实施传递给 SageMaker AI 模型的参数的数量和类型，并在必要时执行类型转换。使用 SQL 类型还会将 SQL NULL 转换为 SageMaker AI 模型预期的适当默认值。
- 如果要更快地返回前几行，则可以减小最大批处理大小。

要指定用户定义的函数，请使用 SQL 数据定义语言 (DDL) 语句 CREATE FUNCTION。在定义函数时，您指定了以下内容：

- 模型的输入参数。
- 要调用的特定 SageMaker AI 端点。
- 返回类型。

用户定义的函数在对输入参数运行模型后，将返回 SageMaker AI 端点计算的推理。以下示例为带两个输入参数的 SageMaker AI 模型创建用户定义的函数。

```
CREATE FUNCTION classify_event (IN arg1 INT, IN arg2 DATE, OUT category INT)
AS $$
    SELECT aws_sagemaker.invoke_endpoint (
        'sagemaker_model_endpoint_name', NULL,
        arg1, arg2                          -- model inputs are separate arguments
        )::INT                              -- cast the output to INT
$$ LANGUAGE SQL PARALLEL SAFE COST 5000;
```

请注意以下几点：

- aws_sagemaker.invoke_endpoint 函数输入可以是任意数据类型的一个或多个参数。
- 此示例使用 INT 输出类型。如果将输出从 varchar 类型强制转换为其他类型，则必须将其强制转换为 PostgreSQL 内置标量类型，例如 INTEGER、REAL、FLOAT 或 NUMERIC。有关这些类型的更多信息，请参阅 PostgreSQL 文档中的数据类型。
- 指定 PARALLEL SAFE 可启用并行查询处理。有关更多信息，请参阅“通过并行查询处理缩短响应时间”。
- 指定 COST 5000 以估计运行函数的成本。使用正数来提供函数的估计运行成本，单位为 cpu_operator_cost。

将数组作为输入传递到 SageMaker AI 模型

aws_sagemaker.invoke_endpoint 函数最多可具有 100 个输入参数，这是 PostgreSQL 函数的限制。如果 SageMaker AI 模型需要 100 个以上的相同类型的参数，则将模型参数作为数组传递。

以下示例定义一个函数，该函数将数组作为输入传递到 SageMaker AI 回归模型。输出将强制转换为 REAL 值。

```
CREATE FUNCTION regression_model (params REAL[], OUT estimate REAL)
AS $$
    SELECT aws_sagemaker.invoke_endpoint (
      'sagemaker_model_endpoint_name',
      NULL,
      params
      )::REAL
$$ LANGUAGE SQL PARALLEL SAFE COST 5000;
```

在调用 SageMaker AI 模型时指定批处理大小

以下示例为 SageMaker AI 模型创建一个用户定义的函数，该函数将批处理大小的默认值设置为 NULL。此外，在您调用该函数时，它可让您提供其他批处理大小。

```
CREATE FUNCTION classify_event (
    IN event_type INT, IN event_day DATE, IN amount REAL, -- model inputs
    max_rows_per_batch INT DEFAULT NULL,  -- optional batch size limit
    OUT category INT)                     -- model output
AS $$
    SELECT aws_sagemaker.invoke_endpoint (
        'sagemaker_model_endpoint_name', max_rows_per_batch,
        event_type, event_day, COALESCE(amount, 0.0)
        )::INT              -- casts output to type INT
$$ LANGUAGE SQL PARALLEL SAFE COST 5000;
```

请注意以下几点：

- 使用可选的 `max_rows_per_batch` 参数可控制批处理模式函数调用的行数。如果您使用 NULL 值，则查询优化程序会自动选择最大批处理大小。有关更多信息，请参阅 [了解批处理模式和 Aurora 机器学习函数](#)。
- 默认情况下，作为参数值传递的 NULL 将转换为空字符串，然后再传递到 SageMaker AI。对于此示例，输入具有不同的类型。
- 如果您具有非文本输入，或具有需要默认为空字符串以外的某个值的文本输入，请使用 `COALESCE` 语句。使用 `COALESCE` 可在对 `aws_sagemaker.invoke_endpoint` 的调用中将 NULL 转换为所需的空替换值。对于此示例中的 `amount` 参数，NULL 值将转换为 0.0。

调用带多个输出的 SageMaker AI 模型

以下示例为返回多个输出的 SageMaker AI 模型创建一个用户定义的函数。您的函数需要将 `aws_sagemaker.invoke_endpoint` 函数的输出强制转换为相应的数据类型。例如，可以将内置 PostgreSQL 点类型用于 (x,y) 对或用户定义的复合类型。

此用户定义的函数将返回一个模型中的值，该模型通过对输出使用复合类型来返回多个输出。

```
CREATE TYPE company_forecasts AS (
    six_month_estimated_return real,
    one_year_bankruptcy_probability float);
CREATE FUNCTION analyze_company (
    IN free_cash_flow NUMERIC(18, 6),
```

```
    IN debt NUMERIC(18,6),
    IN max_rows_per_batch INT DEFAULT NULL,
    OUT prediction company_forecasts)
AS $$
    SELECT (aws_sagemaker.invoke_endpoint('endpt_name',
       max_rows_per_batch,free_cash_flow, debt))::company_forecasts;


$$ LANGUAGE SQL PARALLEL SAFE COST 5000;
```

对于复合类型，按照字段在模型输出中的显示顺序来使用字段，并将 `aws_sagemaker.invoke_endpoint` 的输出强制转换为复合类型。调用方可以按名称或使用 PostgreSQL“.*”表示法来提取各个字段。

将数据导出到 Amazon S3 以进行 SageMaker AI 模型训练（高级）

建议您使用提供的算法和示例来熟悉 Aurora 机器学习和 SageMaker AI，而不是尝试训练自己的模型。有关更多信息，请参阅 Get Started with Amazon SageMaker AI。

要训练 SageMaker AI 模型，请将数据导出到 Amazon S3 存储桶。SageMaker AI 使用 Amazon S3 存储桶在部署模型之前对其进行训练。您可以从 Aurora PostgreSQL 数据库集群中查询数据，并将数据直接保存到 Amazon S3 存储桶中存储的文本文件。然后，SageMaker AI 使用 Amazon S3 存储桶中的数据进行训练。有关 SageMaker AI 模型训练的更多信息，请参阅 Train a model with Amazon SageMaker AI。

> **Note**
>
> 在为 SageMaker AI 模型训练或批处理评分创建 Amazon S3 存储桶时，请在 Amazon S3 存储桶名称中使用 `sagemaker`。有关更多信息，请参阅《Amazon SageMaker AI 开发人员指南》中的 Specify a Amazon S3 Bucket to Upload Training Datasets and Store Output Data。

有关导出数据的更多信息，请参阅将数据从 Aurora PostgreSQL 数据库集群导出到 Amazon S3。

将 Aurora 机器学习与 Aurora PostgreSQL 结合使用的性能注意事项

Amazon Comprehend 和 SageMaker AI 服务在由 Aurora 机器学习函数调用时会完成大部分工作。这意味着您可以根据需要独立扩缩这些资源。对于您的 Aurora PostgreSQL 数据库集群，您可以使函数调用尽可能高效。接下来，您可以找到从 Aurora PostgreSQL 中使用 Aurora 机器学习时需要注意的一些性能注意事项。

主题

- 了解批处理模式和 Aurora 机器学习函数
- 通过并行查询处理缩短响应时间
- 使用具体化视图和具体化列

了解批处理模式和 Aurora 机器学习函数

通常，PostgreSQL 一次运行一行函数。Aurora 机器学习可通过使用称作批处理模式执行的方法，针对多个行将对外部 Aurora 机器学习服务的调用组合为批处理来减少此开销。在批处理模式中，Aurora 机器学习将接收对一批输入行的响应，然后将响应传回正在运行的查询（一次一行）。此优化将提高 Aurora 查询的吞吐量，而不限制 PostgreSQL 查询优化程序。

如果从 SELECT 列表、WHERE 子句或 HAVING 子句引用函数，则 Aurora 会自动使用批处理模式。请注意，顶级的简单 CASE 表达式符合批处理模式执行的条件。顶级搜索的 CASE 表达式符合批处理模式执行的条件，前提是第一个 WHEN 子句是具有批处理模式函数调用的简单谓词。

您的用户定义的函数必须是一个 LANGUAGE SQL 函数，并且应指定 PARALLEL SAFE 和 COST 5000。

从 SELECT 语句到 FROM 子句的函数迁移

通常，Aurora 会自动将符合批处理模式执行条件的 aws_ml 函数迁移到 FROM 子句。

可以在每个查询级别上手动检查合格的批处理模式函数到 FROM 子句的迁移。为此，您可以使用 EXPLAIN 语句（以及 ANALYZE 和 VERBOSE），并在每个批处理模式 Function Scan 下查找“批处理”信息。您也可以在不运行查询的情况下使用 EXPLAIN（以及 VERBOSE）。然后，您可以观察对函数的调用是否在原始语句中未指定的嵌套循环联接下显示为 Function Scan。

在以下示例中，计划中的嵌套循环联接运算符表明 Aurora 已迁移 anomaly_score 函数。它已将此函数从 SELECT 列表迁移到 FROM 子句，其中它符合批处理模式执行的条件。

```
EXPLAIN (VERBOSE, COSTS false)
SELECT anomaly_score(ts.R.description) from ts.R;
                          QUERY PLAN
---------------------------------------------------------------
 Nested Loop
   Output: anomaly_score((r.description)::text)
   ->  Seq Scan on ts.r
         Output: r.id, r.description, r.score
   ->  Function Scan on public.anomaly_score
```

```
        Output: anomaly_score.anomaly_score
        Function Call: anomaly_score((r.description)::text)
```

要禁用批处理模式执行，请将 apg_enable_function_migration 参数设置为 false。这可阻止 aws_ml 函数从 SELECT 迁移到 FROM 子句。下面演示了如何操作。

```
SET apg_enable_function_migration = false;
```

apg_enable_function_migration 参数是一个由查询计划管理的 Aurora PostgreSQL apg_plan_mgmt 扩展识别的 Grand Unified Configuration (GUC) 参数。要在会话中禁用函数迁移，请使用查询计划管理将生成的计划另存为 approved 计划。在运行时，查询计划管理会使用其 approved 设置强制执行 apg_enable_function_migration 计划。无论 apg_enable_function_migration GUC 参数设置如何，此强制执行都会发生。有关更多信息，请参阅“管理 Aurora PostgreSQL 的查询执行计划”。

使用 max_rows_per_batch 参数

aws_comprehend.detect_sentiment 和 aws_sagemaker.invoke_endpoint 函数都有一个 max_rows_per_batch 参数。此参数指定可以发送到 Aurora 机器学习服务的行数。函数处理的数据集越大，批处理大小就可以越大。

批处理模式函数通过构建批量行来将 Aurora 机器学习函数调用的成本分散在大量行上，从而提高效率。但是，如果 SELECT 语句因 LIMIT 子句而提前完成，则可以在比查询使用的行更多的行上构建批处理。此方法可能会导致您的 AWS 账户产生额外费用。要获得批处理模式执行的好处，并避免构建过大的批处理，请在函数调用中使用较小的 max_rows_per_batch 参数值。

如果您执行使用批处理模式执行的查询的 EXPLAIN（VERBOSE、ANALYZE），则嵌套循环联接下方将显示 FunctionScan 运算符。EXPLAIN 报告的循环数等于已从 FunctionScan 运算符提取某个行的次数。如果语句使用了 LIMIT 子句，则提取次数是一致的。要优化批处理的大小，请将 max_rows_per_batch 参数设置为此值。但是，如果批处理模式函数是在 WHERE 子句或 HAVING 子句的谓词中引用的，则您可能无法预先知道提取次数。在此情况下，使用循环作为指南，并尝试使用 max_rows_per_batch 查找可优化性能的设置。

验证批处理模式执行

要查看函数是否在批处理模式下运行，请使用 EXPLAIN ANALYZE。如果使用了批处理模式执行，则查询计划将在“批处理”部分中包含此信息。

```
EXPLAIN ANALYZE SELECT user-defined-function();
```

```
Batch Processing: num batches=1 avg/min/max batch size=3333.000/3333.000/3333.000
                                avg/min/max batch call time=146.273/146.273/146.273
```

在此示例中，有 1 个批处理包含 3333 个行，其处理时间为 146.273 毫秒。“批处理”部分显示以下内容：

- 针对此函数扫描操作的批处理的数目
- 批处理大小的平均值、最小值和最大值
- 批处理执行时间的平均值、最小值和最大值

最后一个批处理通常会小于其余的批处理，这通常会导致最小批处理大小比平均值小得多。

要更快地返回前几行，请将 `max_rows_per_batch` 参数设置为较小的值。

要在用户定义的函数中使用 `LIMIT` 时减少对 ML 服务的批处理模式调用的次数，请将 `max_rows_per_batch` 参数设置为较小的值。

通过并行查询处理缩短响应时间

要尽快从大量行中获取结果，可以将并行查询处理与批处理模式处理结合使用。您可以对 `SELECT`、`CREATE TABLE AS SELECT` 和 `CREATE MATERIALIZED VIEW` 语句使用并行查询处理。

> **Note**
>
> PostgreSQL 尚不支持对数据操作语言 (DML) 语句进行并行查询。

并行查询处理可同时在数据库和 ML 服务内进行。数据库实例类中的核心数将限制运行查询时可使用的并行度。数据库服务器可以构建并行查询执行计划，来将任务划分到一组并行工作线程中。随后，所有这些工作线程都能构建包含数万个（或每个服务允许的数量）行的批处理的请求。

来自所有并行工件的批处理请求将发送到 SageMaker AI 端点。端点可以支持的并行度受支持此端点的实例的数量和类型所约束。要获得 K 并行度，您需要一个具有至少 K 个核心的数据库实例类。您还需要为模型配置 SageMaker AI 端点，以具有性能足够高的实例类的 K 个初始实例。

要使用并行查询处理，您可以为包含您计划传递的数据的表设置 `parallel_workers` 存储参数。将 `parallel_workers` 设置为批处理模式函数，例如 `aws_comprehend.detect_sentiment`。如果优化程序选择并行查询计划，则可以批量调用和并行调用AWS ML 服务。

可以将以下参数与 aws_comprehend.detect_sentiment 函数结合使用来获取具有四向并行度的计划。如果您更改以下两个参数中的任一参数，则必须重新启动数据库实例才能使更改生效。

```
-- SET max_worker_processes to 8;  -- default value is 8
-- SET max_parallel_workers to 8;  -- not greater than max_worker_processes
SET max_parallel_workers_per_gather to 4;  -- not greater than max_parallel_workers

-- You can set the parallel_workers storage parameter on the table that the data
-- for the Aurora machine learning function is coming from in order to manually
 override the degree of
-- parallelism that would otherwise be chosen by the query optimizer
--
ALTER TABLE yourTable SET (parallel_workers = 4);

-- Example query to exploit both batch-mode execution and parallel query
EXPLAIN (verbose, analyze, buffers, hashes)
SELECT aws_comprehend.detect_sentiment(description, 'en')).*
FROM yourTable
WHERE id < 100;
```

有关控制并行查询的更多信息，请参阅 PostgreSQL 文档中的并行计划。

使用具体化视图和具体化列

当您从数据库中调用 AWS 服务（例如 SageMaker AI 或 Amazon Comprehend）时，将根据该服务的定价策略向您的账户收费。要最大程度地减少向您的账户收取的费用，您可以将调用 AWS 服务的结果具体化为一个具体化列，以便针对每个输入行调用 AWS 服务的次数不会超过一次。如果需要，您可以添加 materializedAt 时间戳列来记录将列具体化的时间。

通用的单行 INSERT 语句的延迟通常比调用批处理模式函数的延迟短得多。因此，如果您为应用程序执行的每个单行 INSERT 调用批处理模式函数，则可能无法满足应用程序的延迟要求。要将调用 AWS 服务的结果具体化为一个具体化列，高性能的应用程序通常需要填充具体化列。为此，它们会定期发布同时在大批行上运行的 UPDATE 语句。

UPDATE 接受可能影响正在运行的应用程序的行级锁定。因此，您可能需要使用 SELECT ... FOR UPDATE SKIP LOCKED，或使用 MATERIALIZED VIEW。

针对大量的行实时运行的分析查询可以将批处理模式具体化与实时处理相结合。为此，这些查询会将预具体化结果的 UNION ALL 与对尚没有具体化结果的行执行的查询相结合。在某些情况下，多个位置需

要此类 UNION ALL，否则查询将由第三方应用程序生成。如果是这样，您可以创建一个 VIEW 来封装 UNION ALL 操作，以便不向 SQL 应用程序的其余部分公开此详细信息。

您可以使用具体化视图在即时快照中具体化任意 SELECT 语句的结果。您还可以将来随时使用它刷新具体化视图。目前，PostgreSQL 不支持增量刷新，因此每次刷新具体化视图时，都会完全重新计算具体化视图。

可以使用 CONCURRENTLY 选项刷新具体化视图，这将在不使用排他锁的情况下更新具体化视图的内容。通过执行此操作，使 SQL 应用程序能够在刷新具体化视图时从该视图中进行读取。

监控 Aurora 机器学习

您可以通过将自定义数据库集群参数组中的 track_functions 参数设置为 all 来监控 aws_ml 函数。默认情况下，此参数设置为 pl，这意味着仅跟踪过程语言函数。通过将其更改为 all，还可以跟踪 aws_ml 函数。有关更多信息，请参阅 PostgreSQL 文档中的运行时统计数据。

有关监控从 Aurora 机器学习函数调用的 SageMaker AI 操作的性能的信息，请参阅《Amazon SageMaker AI 开发人员指南》中的 Monitor Amazon SageMaker AI。

在将 track_functions 设置为 all 时，您可以查询 pg_stat_user_functions 视图，以获取有关您定义和用于调用 Aurora 机器学习服务的函数的统计数据。对于每个函数，该视图提供 calls、total_time 和 self_time 的数量。

要查看 aws_sagemaker.invoke_endpoint 和 aws_comprehend.detect_sentiment 函数的统计数据，您可以使用以下查询按模式名称筛选结果。

```
SELECT * FROM pg_stat_user_functions
    WHERE schemaname
    LIKE 'aws_%';
```

要清除统计数据，请执行以下操作。

```
SELECT pg_stat_reset();
```

您可以通过查询 PostgreSQL pg_proc 系统目录来获取调用 aws_sagemaker.invoke_endpoint 函数的 SQL 函数的名称。该目录存储有关函数、过程等的信息。有关更多信息，请参阅 PostgreSQL 文档中的 pg_proc。下面的示例查询该表，以获取其源代码（prosrc）包含文本 invoke_endpoint 的函数的名称（proname）。

```
SELECT proname FROM pg_proc WHERE prosrc LIKE '%invoke_endpoint%';
```

使用 AWS SDK 的 Aurora 代码示例

以下代码示例显示如何将 Aurora 与 AWS 软件开发工具包（SDK）一起使用。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景是展示如何通过在同一服务中调用多个函数来完成特定任务任务的代码示例。

跨服务示例是指跨多个 AWS 服务工作的示例应用程序。

有关 AWS SDK 开发人员指南和代码示例的完整列表，请参阅 将此服务与 AWS SDK 结合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

开始使用

开始使用 Aurora

以下代码示例显示如何开始使用 Aurora。

.NET

适用于 .NET 的 SDK

> **Note**
>
> 在 GitHub 上查看更多内容。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
using Amazon.RDS;
using Amazon.RDS.Model;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;

namespace AuroraActions;

public static class HelloAurora
{
    static async Task Main(string[] args)
    {
```

```
        // Use the AWS .NET Core Setup package to set up dependency injection for
 the
        // Amazon Relational Database Service (Amazon RDS).
        // Use your AWS profile name, or leave it blank to use the default
 profile.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonRDS>()
            ).Build();

        // Now the client is available for injection. Fetching it directly here
 for example purposes only.
        var rdsClient = host.Services.GetRequiredService<IAmazonRDS>();

        // You can use await and any of the async methods to get a response.
        var response = await rdsClient.DescribeDBClustersAsync(new
 DescribeDBClustersRequest { IncludeShared = true });
        Console.WriteLine($"Hello Amazon RDS Aurora! Let's list some clusters in
 this account:");
        foreach (var cluster in response.DBClusters)
        {
            Console.WriteLine($"\tCluster: database: {cluster.DatabaseName}
 identifier: {cluster.DBClusterIdentifier}.");
        }
    }
}
```

- 有关 API 详细信息，请参阅《适用于 .NET 的 AWS SDK API 参考》中的 DescribeDBClusters。

C++

SDK for C++

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

CMakeLists.txt CMake 文件的代码。

```
# Set the minimum required version of CMake for this project.
cmake_minimum_required(VERSION 3.13)

# Set the AWS service components used by this project.
set(SERVICE_COMPONENTS rds)

# Set this project's name.
project("hello_aurora")

# Set the C++ standard to use to build this target.
# At least C++ 11 is required for the AWS SDK for C++.
set(CMAKE_CXX_STANDARD 11)

# Use the MSVC variable to determine if this is a Windows build.
set(WINDOWS_BUILD ${MSVC})

if (WINDOWS_BUILD) # Set the location where CMake can find the installed
 libraries for the AWS SDK.
    string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
 "${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
    list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
endif ()

# Find the AWS SDK for C++ package.
find_package(AWSSDK REQUIRED COMPONENTS ${SERVICE_COMPONENTS})

if (WINDOWS_BUILD AND AWSSDK_INSTALL_AS_SHARED_LIBS)
     # Copy relevant AWS SDK for C++ libraries into the current binary directory
 for running and debugging.

     # set(BIN_SUB_DIR "/Debug") # If you are building from the command line, you
 may need to uncomment this
                                     # and set the proper subdirectory to the
 executables' location.

     AWSSDK_CPY_DYN_LIBS(SERVICE_COMPONENTS ""
 ${CMAKE_CURRENT_BINARY_DIR}${BIN_SUB_DIR})
endif ()

add_executable(${PROJECT_NAME}
        hello_aurora.cpp)

target_link_libraries(${PROJECT_NAME}
```

```
        ${AWSSDK_LINK_LIBRARIES})
```

hello_aurora.cpp 源文件的代码。

```
#include <aws/core/Aws.h>
#include <aws/rds/RDSClient.h>
#include <aws/rds/model/DescribeDBClustersRequest.h>
#include <iostream>

/*
 *  A "Hello Aurora" starter application which initializes an Amazon Relational
 Database Service (Amazon RDS) client
 *  and describes the Amazon Aurora (Aurora) clusters.
 *
 *  main function
 *
 *  Usage: 'hello_aurora'
 *
 */
int main(int argc, char **argv) {
    Aws::SDKOptions options;
    // Optionally change the log level for debugging.
//   options.loggingOptions.logLevel = Utils::Logging::LogLevel::Debug;
    Aws::InitAPI(options); // Should only be called once.
    int result = 0;
    {
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

        Aws::RDS::RDSClient rdsClient(clientConfig);

        Aws::String marker; // Used for pagination.
        std::vector<Aws::String> clusterIds;
        do {
            Aws::RDS::Model::DescribeDBClustersRequest request;

            Aws::RDS::Model::DescribeDBClustersOutcome outcome =
                    rdsClient.DescribeDBClusters(request);

            if (outcome.IsSuccess()) {
                for (auto &cluster: outcome.GetResult().GetDBClusters()) {
```

```
                    clusterIds.push_back(cluster.GetDBClusterIdentifier());
                }
                marker = outcome.GetResult().GetMarker();
            } else {
                result = 1;
                std::cerr << "Error with Aurora::GDescribeDBClusters. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                break;
            }
        } while (!marker.empty());

        std::cout << clusterIds.size() << " Aurora clusters found." << std::endl;
        for (auto &clusterId: clusterIds) {
            std::cout << "  clusterId " << clusterId << std::endl;
        }
    }

    Aws::ShutdownAPI(options); // Should only be called once.
    return 0;
}
```

- 有关 API 详细信息，请参阅《适用于 C++ 的 AWS SDK API 参考》中的 DescribeDBClusters。

Go

适用于 Go V2 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
package main

import (
 "context"
```

```
 "fmt"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/rds"
)

// main uses the AWS SDK for Go V2 to create an Amazon Aurora client and list up
 to 20
// DB clusters in your account.
// This example uses the default settings specified in your shared credentials
// and config files.
func main() {
 sdkConfig, err := config.LoadDefaultConfig(context.TODO())
 if err != nil {
  fmt.Println("Couldn't load default configuration. Have you set up your AWS
 account?")
  fmt.Println(err)
  return
 }
 auroraClient := rds.NewFromConfig(sdkConfig)
 const maxClusters = 20
 fmt.Printf("Let's list up to %v DB clusters.\n", maxClusters)
 output, err := auroraClient.DescribeDBClusters(context.TODO(),
  &rds.DescribeDBClustersInput{MaxRecords: aws.Int32(maxClusters)})
 if err != nil {
  fmt.Printf("Couldn't list DB clusters: %v\n", err)
  return
 }
 if len(output.DBClusters) == 0 {
  fmt.Println("No DB clusters found.")
 } else {
  for _, cluster := range output.DBClusters {
   fmt.Printf("DB cluster %v has database %v.\n", *cluster.DBClusterIdentifier,
    *cluster.DatabaseName)
  }
 }
}
```

- 有关 API 详细信息，请参阅《适用于 Go 的 AWS SDK API 参考》中的 DescribeDBClusters。

Java

SDK for Java 2.x

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rds.RdsClient;
import software.amazon.awssdk.services.rds.paginators.DescribeDBClustersIterable;

public class DescribeDbClusters {
    public static void main(String[] args) {
        Region region = Region.US_EAST_1;
        RdsClient rdsClient = RdsClient.builder()
                .region(region)
                .build();

        describeClusters(rdsClient);
        rdsClient.close();
    }

    public static void describeClusters(RdsClient rdsClient) {
        DescribeDBClustersIterable clustersIterable =
 rdsClient.describeDBClustersPaginator();
        clustersIterable.stream()
                .flatMap(r -> r.dbClusters().stream())
                .forEach(cluster -> System.out
                        .println("Database name: " + cluster.databaseName() + "
 Arn = " + cluster.dbClusterArn()));
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Java 2.x API 参考》中的 DescribeDBClusters。

Python

SDK for Python (Boto3)

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
import boto3

# Create an RDS client
rds = boto3.client("rds")

# Create a paginator for the describe_db_clusters operation
paginator = rds.get_paginator("describe_db_clusters")

# Use the paginator to get a list of DB clusters
response_iterator = paginator.paginate(
    PaginationConfig={
        "PageSize": 50,  # Adjust PageSize as needed
        "StartingToken": None,
    }
)

# Iterate through the pages of the response
clusters_found = False
for page in response_iterator:
    if "DBClusters" in page and page["DBClusters"]:
        clusters_found = True
        print("Here are your RDS Aurora clusters:")
        for cluster in page["DBClusters"]:
            print(
                f"Cluster ID: {cluster['DBClusterIdentifier']}, Engine:
 {cluster['Engine']}"
            )

if not clusters_found:
    print("No clusters found!")
```

- 有关 API 详细信息，请参阅《AWS SDK for Python (Boto3) API 参考》中的 DescribeDBClusters。

Ruby

适用于 Ruby 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
require 'aws-sdk-rds'

# Creates an Amazon RDS client for the AWS Region
rds = Aws::RDS::Client.new

puts 'Listing clusters in this AWS account...'

# Calls the describe_db_clusters method to get information about clusters
resp = rds.describe_db_clusters(max_records: 20)

# Checks if any clusters are found and prints the appropriate message
if resp.db_clusters.empty?
  puts 'No clusters found!'
else
  # Loops through the array of cluster objects and prints the cluster identifier
  resp.db_clusters.each do |cluster|
    puts "Cluster identifier: #{cluster.db_cluster_identifier}"
  end
end
```

- 有关 API 详细信息，请参阅《适用于 Ruby 的 AWS SDK API 参考》中的 DescribeDBClusters。

Rust

适用于 Rust 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_sdk_rds::Client;

#[derive(Debug)]
struct Error(String);
impl std::fmt::Display for Error {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        write!(f, "{}", self.0)
    }
}
impl std::error::Error for Error {}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt::init();
    let sdk_config = aws_config::from_env().load().await;
    let client = Client::new(&sdk_config);

    let describe_db_clusters_output = client
        .describe_db_clusters()
        .send()
        .await
        .map_err(|e| Error(e.to_string()))?;
    println!(
        "Found {} clusters:",
        describe_db_clusters_output.db_clusters().len()
    );
    for cluster in describe_db_clusters_output.db_clusters() {
        let name = cluster.database_name().unwrap_or("Unknown");
        let engine = cluster.engine().unwrap_or("Unknown");
        let id = cluster.db_cluster_identifier().unwrap_or("Unknown");
```

```
        let class = cluster.db_cluster_instance_class().unwrap_or("Unknown");
        println!("\tDatabase: {name}",);
        println!("\t  Engine: {engine}",);
        println!("\t      ID: {id}",);
        println!("\tInstance: {class}",);
    }

    Ok(())
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Rust API 参考》中的 DescribeDBClusters。

代码示例

- 使用 AWS SDK 的 Aurora 操作
  - 将 CreateDBCluster 与 AWS SDK 或 CLI 配合使用
  - 将 CreateDBClusterParameterGroup 与 AWS SDK 或 CLI 配合使用
  - 将 CreateDBClusterSnapshot 与 AWS SDK 或 CLI 配合使用
  - 将 CreateDBInstance 与 AWS SDK 或 CLI 配合使用
  - 将 DeleteDBCluster 与 AWS SDK 或 CLI 配合使用
  - 将 DeleteDBClusterParameterGroup 与 AWS SDK 或 CLI 配合使用
  - 将 DeleteDBInstance 与 AWS SDK 或 CLI 配合使用
  - 将 DescribeDBClusterParameterGroups 与 AWS SDK 或 CLI 配合使用
  - 将 DescribeDBClusterParameters 与 AWS SDK 或 CLI 配合使用
  - 将 DescribeDBClusterSnapshots 与 AWS SDK 或 CLI 配合使用
  - 将 DescribeDBClusters 与 AWS SDK 或 CLI 配合使用
  - 将 DescribeDBEngineVersions 与 AWS SDK 或 CLI 配合使用
  - 将 DescribeDBInstances 与 AWS SDK 或 CLI 配合使用
  - 将 DescribeOrderableDBInstanceOptions 与 AWS SDK 或 CLI 配合使用
  - 将 ModifyDBClusterParameterGroup 与 AWS SDK 或 CLI 配合使用
- 使用 AWS SDK 的 Aurora 场景
  - 通过 AWS SDK 开始使用 Aurora 数据库集群
- 使用 AWS SDK 的 Aurora 跨服务示例
  - 创建借阅图书馆 REST API

- 创建 Aurora Serverless 工作项跟踪器

使用 AWS SDK 的 Aurora 操作

以下代码示例演示了如何使用 AWS SDK 来执行各个 Aurora 操作。这些代码节选调用了 Aurora API，是必须在上下文中运行的较大型程序的代码节选。每个示例都包含一个指向 GitHub 的链接，您可以在其中找到有关设置和运行代码的说明。

以下示例仅包括最常用的操作。有关完整列表，请参阅 Amazon Aurora API 参考。

示例

- 将 CreateDBCluster 与 AWS SDK 或 CLI 配合使用
- 将 CreateDBClusterParameterGroup 与 AWS SDK 或 CLI 配合使用
- 将 CreateDBClusterSnapshot 与 AWS SDK 或 CLI 配合使用
- 将 CreateDBInstance 与 AWS SDK 或 CLI 配合使用
- 将 DeleteDBCluster 与 AWS SDK 或 CLI 配合使用
- 将 DeleteDBClusterParameterGroup 与 AWS SDK 或 CLI 配合使用
- 将 DeleteDBInstance 与 AWS SDK 或 CLI 配合使用
- 将 DescribeDBClusterParameterGroups 与 AWS SDK 或 CLI 配合使用
- 将 DescribeDBClusterParameters 与 AWS SDK 或 CLI 配合使用
- 将 DescribeDBClusterSnapshots 与 AWS SDK 或 CLI 配合使用
- 将 DescribeDBClusters 与 AWS SDK 或 CLI 配合使用
- 将 DescribeDBEngineVersions 与 AWS SDK 或 CLI 配合使用
- 将 DescribeDBInstances 与 AWS SDK 或 CLI 配合使用
- 将 DescribeOrderableDBInstanceOptions 与 AWS SDK 或 CLI 配合使用
- 将 ModifyDBClusterParameterGroup 与 AWS SDK 或 CLI 配合使用

将 `CreateDBCluster` 与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `CreateDBCluster`。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 开始使用数据库集群

.NET

适用于 .NET 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
/// <summary>
/// Create a new cluster and database.
/// </summary>
/// <param name="dbName">The name of the new database.</param>
/// <param name="clusterIdentifier">The identifier of the cluster.</param>
/// <param name="parameterGroupName">The name of the parameter group.</param>
/// <param name="dbEngine">The engine to use for the new cluster.</param>
/// <param name="dbEngineVersion">The version of the engine to use.</param>
/// <param name="adminName">The admin username.</param>
/// <param name="adminPassword">The primary admin password.</param>
/// <returns>The cluster object.</returns>
public async Task<DBCluster> CreateDBClusterWithAdminAsync(
    string dbName,
    string clusterIdentifier,
    string parameterGroupName,
    string dbEngine,
    string dbEngineVersion,
    string adminName,
    string adminPassword)
{
    var request = new CreateDBClusterRequest
    {
        DatabaseName = dbName,
        DBClusterIdentifier = clusterIdentifier,
        DBClusterParameterGroupName = parameterGroupName,
        Engine = dbEngine,
        EngineVersion = dbEngineVersion,
        MasterUsername = adminName,
        MasterUserPassword = adminPassword,
    };
```

```
        var response = await _amazonRDS.CreateDBClusterAsync(request);
        return response.DBCluster;
    }
```

- 有关 API 详细信息，请参阅《适用于 .NET 的 AWS SDK API 参考》中的 CreateDBCluster。

C++

SDK for C++

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

        Aws::RDS::Model::CreateDBClusterRequest request;
        request.SetDBClusterIdentifier(DB_CLUSTER_IDENTIFIER);
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);
        request.SetEngine(engineName);
        request.SetEngineVersion(engineVersionName);
        request.SetMasterUsername(administratorName);
        request.SetMasterUserPassword(administratorPassword);

        Aws::RDS::Model::CreateDBClusterOutcome outcome =
                client.CreateDBCluster(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB cluster creation has started."
                      << std::endl;
        }
        else {
            std::cerr << "Error with Aurora::CreateDBCluster. "
                      << outcome.GetError().GetMessage()
```

```
                    << std::endl;
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, "", "", client);
            return false;
        }
```

- 有关 API 详细信息，请参阅《适用于 C++ 的 AWS SDK API 参考》中的 CreateDBCluster。

Go

适用于 Go V2 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
type DbClusters struct {
 AuroraClient *rds.Client
}



// CreateDbCluster creates a DB cluster that is configured to use the specified
 parameter group.
// The newly created DB cluster contains a database that uses the specified
 engine and
// engine version.
func (clusters *DbClusters) CreateDbCluster(clusterName string,
 parameterGroupName string,
 dbName string, dbEngine string, dbEngineVersion string, adminName string,
 adminPassword string) (
 *types.DBCluster, error) {

 output, err := clusters.AuroraClient.CreateDBCluster(context.TODO(),
 &rds.CreateDBClusterInput{
  DBClusterIdentifier:         aws.String(clusterName),
  Engine:                      aws.String(dbEngine),
  DBClusterParameterGroupName: aws.String(parameterGroupName),
  DatabaseName:                aws.String(dbName),
```

```
  EngineVersion:               aws.String(dbEngineVersion),
  MasterUserPassword:          aws.String(adminPassword),
  MasterUsername:              aws.String(adminName),
 })
 if err != nil {
  log.Printf("Couldn't create DB cluster %v: %v\n", clusterName, err)
  return nil, err
 } else {
  return output.DBCluster, err
 }
}
```

- 有关 API 详细信息，请参阅《适用于 Go 的 AWS SDK API 参考》中的 CreateDBCluster。

Java

SDK for Java 2.x

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
   public static String createDBCluster(RdsClient rdsClient, String
dbParameterGroupFamily, String dbName,
           String dbClusterIdentifier, String userName, String password) {
       try {
           CreateDbClusterRequest clusterRequest =
CreateDbClusterRequest.builder()
                   .databaseName(dbName)
                   .dbClusterIdentifier(dbClusterIdentifier)
                   .dbClusterParameterGroupName(dbParameterGroupFamily)
                   .engine("aurora-mysql")
                   .masterUsername(userName)
                   .masterUserPassword(password)
                   .build();

           CreateDbClusterResponse response =
rdsClient.createDBCluster(clusterRequest);
```

```
            return response.dbCluster().dbClusterArn();

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
        return "";
    }
```

- 有关 API 详细信息，请参阅《AWS SDK for Java 2.x API 参考》中的 CreateDBCluster。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
suspend fun createDBCluster(
    dbParameterGroupFamilyVal: String?,
    dbName: String?,
    dbClusterIdentifierVal: String?,
    userName: String?,
    password: String?,
): String? {
    val clusterRequest =
        CreateDbClusterRequest {
            databaseName = dbName
            dbClusterIdentifier = dbClusterIdentifierVal
            dbClusterParameterGroupName = dbParameterGroupFamilyVal
            engine = "aurora-mysql"
            masterUsername = userName
            masterUserPassword = password
        }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbCluster(clusterRequest)
        return response.dbCluster?.dbClusterArn
```

```
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Kotlin API 参考》中的 CreateDBCluster。

Python

SDK for Python (Boto3)

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def create_db_cluster(
        self,
        cluster_name,
        parameter_group_name,
        db_name,
        db_engine,
        db_engine_version,
```

```
        admin_name,
        admin_password,
    ):
        """
        Creates a DB cluster that is configured to use the specified parameter
group.
        The newly created DB cluster contains a database that uses the specified
engine and
        engine version.

        :param cluster_name: The name of the DB cluster to create.
        :param parameter_group_name: The name of the parameter group to associate
with
                                     the DB cluster.
        :param db_name: The name of the database to create.
        :param db_engine: The database engine of the database that is created,
such as MySql.
        :param db_engine_version: The version of the database engine.
        :param admin_name: The user name of the database administrator.
        :param admin_password: The password of the database administrator.
        :return: The newly created DB cluster.
        """
        try:
            response = self.rds_client.create_db_cluster(
                DatabaseName=db_name,
                DBClusterIdentifier=cluster_name,
                DBClusterParameterGroupName=parameter_group_name,
                Engine=db_engine,
                EngineVersion=db_engine_version,
                MasterUsername=admin_name,
                MasterUserPassword=admin_password,
            )
            cluster = response["DBCluster"]
        except ClientError as err:
            logger.error(
                "Couldn't create database %s. Here's why: %s: %s",
                db_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return cluster
```

- 有关 API 详细信息，请参阅《AWS SDK for Python (Boto3) API 参考》中的 CreateDBCluster。

Rust

适用于 Rust 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
   // Get a list of allowed engine versions.
rds.DescribeDbEngineVersions(Engine='aurora-mysql', DBParameterGroupFamily=<the
family used to create your parameter group in step 2>)
   // Create an Aurora DB cluster database cluster that contains a MySql
database and uses the parameter group you created.
   // Wait for DB cluster to be ready. Call rds.DescribeDBClusters and check for
Status == 'available'.
   // Get a list of instance classes available for the selected engine
and engine version. rds.DescribeOrderableDbInstanceOptions(Engine='mysql',
EngineVersion=).

   // Create a database instance in the cluster.
   // Wait for DB instance to be ready. Call rds.DescribeDbInstances and check
for DBInstanceStatus == 'available'.
   pub async fn start_cluster_and_instance(&mut self) -> Result<(),
ScenarioError> {
       if self.password.is_none() {
           return Err(ScenarioError::with(
               "Must set Secret Password before starting a cluster",
           ));
       }
       let create_db_cluster = self
           .rds
           .create_db_cluster(
               DB_CLUSTER_IDENTIFIER,
               DB_CLUSTER_PARAMETER_GROUP_NAME,
```

```
                DB_ENGINE,
                self.engine_version.as_deref().expect("engine version"),
                self.username.as_deref().expect("username"),
                self.password
                    .replace(SecretString::new("".to_string()))
                    .expect("password"),
            )
            .await;
        if let Err(err) = create_db_cluster {
            return Err(ScenarioError::new(
                "Failed to create DB Cluster with cluster group",
                &err,
            ));
        }

        self.db_cluster_identifier = create_db_cluster
            .unwrap()
            .db_cluster
            .and_then(|c| c.db_cluster_identifier);

        if self.db_cluster_identifier.is_none() {
            return Err(ScenarioError::with("Created DB Cluster missing
Identifier"));
        }

        info!(
            "Started a db cluster: {}",
            self.db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing ARN")
        );

        let create_db_instance = self
            .rds
            .create_db_instance(
                self.db_cluster_identifier.as_deref().expect("cluster name"),
                DB_INSTANCE_IDENTIFIER,
                self.instance_class.as_deref().expect("instance class"),
                DB_ENGINE,
            )
            .await;
        if let Err(err) = create_db_instance {
            return Err(ScenarioError::new(
                "Failed to create Instance in DB Cluster",
```

```
                &err,
            ));
        }

        self.db_instance_identifier = create_db_instance
            .unwrap()
            .db_instance
            .and_then(|i| i.db_instance_identifier);

        // Cluster creation can take up to 20 minutes to become available
        let cluster_max_wait = Duration::from_secs(20 * 60);
        let waiter = Waiter::builder().max(cluster_max_wait).build();
        while waiter.sleep().await.is_ok() {
            let cluster = self
                .rds
                .describe_db_clusters(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = cluster {
                warn!(?err, "Failed to describe cluster while waiting for
ready");
                continue;
            }

            let instance = self
                .rds
                .describe_db_instance(
                    self.db_instance_identifier
                        .as_deref()
                        .expect("instance identifier"),
                )
                .await;
            if let Err(err) = instance {
                return Err(ScenarioError::new(
                    "Failed to find instance for cluster",
                    &err,
                ));
            }

            let instances_available = instance
```

```
                .unwrap()
                .db_instances()
                .iter()
                .all(|instance| instance.db_instance_status() ==
Some("Available"));

            let endpoints = self
                .rds
                .describe_db_cluster_endpoints(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = endpoints {
                return Err(ScenarioError::new(
                    "Failed to find endpoint for cluster",
                    &err,
                ));
            }

            let endpoints_available = endpoints
                .unwrap()
                .db_cluster_endpoints()
                .iter()
                .all(|endpoint| endpoint.status() == Some("available"));

            if instances_available && endpoints_available {
                return Ok(());
            }
        }

        Err(ScenarioError::with("timed out waiting for cluster"))
    }

    pub async fn create_db_cluster(
        &self,
        name: &str,
        parameter_group: &str,
        engine: &str,
        version: &str,
        username: &str,
        password: SecretString,
```

```
    ) -> Result<CreateDbClusterOutput, SdkError<CreateDBClusterError>> {
        self.inner
            .create_db_cluster()
            .db_cluster_identifier(name)
            .db_cluster_parameter_group_name(parameter_group)
            .engine(engine)
            .engine_version(version)
            .master_username(username)
            .master_user_password(password.expose_secret())
            .send()
            .await
    }

#[tokio::test]
async fn test_start_cluster_and_instance() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
```

```
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|id| {
            Ok(DescribeDbClustersOutput::builder()

.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_describe_db_instance()
        .with(eq("RustSDKCodeExamplesDBInstance"))
        .return_once(|name| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_instance_identifier(name)
                        .db_instance_status("Available")
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

.db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
```

```
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
        assert!(scenario
            .password
            .replace(SecretString::new("BAD SECRET".into()))
            .unwrap()
            .expose_secret()
            .is_empty());
        assert_eq!(
            scenario.db_cluster_identifier,
            Some("RustSDKCodeExamplesDBCluster".into())
        );
    });
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Err(SdkError::service_error(
                CreateDBClusterError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });
```

```
    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _}) if message
 == "Failed to create DB Cluster with cluster group")
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_missing_id() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()
                .db_cluster(DbCluster::builder().build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context:_ }) if message
 == "Created DB Cluster missing Identifier");
}

#[tokio::test]
async fn test_start_cluster_and_instance_instance_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
```

```
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .return_once(|_, _, _, _| {
            Err(SdkError::service_error(
                CreateDBInstanceError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db instance error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _ }) if message
 == "Failed to create Instance in DB Cluster")
}

#[tokio::test]
async fn test_start_cluster_and_instance_wait_hiccup() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
```

```
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

.db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
```

```
                    "describe cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        })
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()

.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

   mock_rds.expect_describe_db_instance().return_once(|name| {
        Ok(DescribeDbInstancesOutput::builder()
            .db_instances(
                DbInstance::builder()
                    .db_instance_identifier(name)
                    .db_instance_status("Available")
                    .build(),
            )
            .build())
   });

   mock_rds
        .expect_describe_db_cluster_endpoints()
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

.db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

   let mut scenario = AuroraScenario::new(mock_rds);
   scenario.engine_version = Some("aurora-mysql8.0".into());
   scenario.instance_class = Some("m5.large".into());
   scenario.username = Some("test username".into());
   scenario.password = Some(SecretString::new("test password".into()));

   tokio::time::pause();
   let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
```

```
        assert!(create.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Rust API 参考》中的 CreateDBCluster。

有关 AWS SDK 开发人员指南和代码示例的完整列表，请参阅 将此服务与 AWS SDK 结合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

将 **CreateDBClusterParameterGroup** 与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 CreateDBClusterParameterGroup。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 开始使用数据库集群

.NET

适用于 .NET 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
    /// <summary>
    /// Create a custom cluster parameter group.
    /// </summary>
    /// <param name="parameterGroupFamily">The family of the parameter group.</
param>
    /// <param name="groupName">The name for the new parameter group.</param>
```

```
    /// <param name="description">A description for the new parameter group.</
param>
    /// <returns>The new parameter group object.</returns>
    public async Task<DBClusterParameterGroup>
 CreateCustomClusterParameterGroupAsync(
        string parameterGroupFamily,
        string groupName,
        string description)
    {
        var request = new CreateDBClusterParameterGroupRequest
        {
            DBParameterGroupFamily = parameterGroupFamily,
            DBClusterParameterGroupName = groupName,
            Description = description,
        };

        var response = await
 _amazonRDS.CreateDBClusterParameterGroupAsync(request);
        return response.DBClusterParameterGroup;
    }
```

- 有关 API 详细信息，请参阅《适用于 .NET 的 AWS SDK API 参考》中的 CreateDBClusterParameterGroup。

C++

SDK for C++

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);
```

```
        Aws::RDS::Model::CreateDBClusterParameterGroupRequest request;
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);
        request.SetDBParameterGroupFamily(dbParameterGroupFamily);
        request.SetDescription("Example cluster parameter group.");

        Aws::RDS::Model::CreateDBClusterParameterGroupOutcome outcome =
                client.CreateDBClusterParameterGroup(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB cluster parameter group was successfully
created."
                      << std::endl;
        }
        else {
            std::cerr << "Error with Aurora::CreateDBClusterParameterGroup. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
```

- 有关 API 详细信息，请参阅《适用于 C++ 的 AWS SDK API 参考》中的 CreateDBClusterParameterGroup。

Go

适用于 Go V2 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
type DbClusters struct {
 AuroraClient *rds.Client
}
```

```
// CreateParameterGroup creates a DB cluster parameter group that is based on the
 specified
// parameter group family.
func (clusters *DbClusters) CreateParameterGroup(
 parameterGroupName string, parameterGroupFamily string, description string) (
 *types.DBClusterParameterGroup, error) {

 output, err :=
 clusters.AuroraClient.CreateDBClusterParameterGroup(context.TODO(),
  &rds.CreateDBClusterParameterGroupInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
   DBParameterGroupFamily:      aws.String(parameterGroupFamily),
   Description:                 aws.String(description),
  })
 if err != nil {
  log.Printf("Couldn't create parameter group %v: %v\n", parameterGroupName, err)
  return nil, err
 } else {
  return output.DBClusterParameterGroup, err
 }
}
```

- 有关 API 详细信息，请参阅《适用于 Go 的 AWS SDK API 参考》中的 CreateDBClusterParameterGroup。

Java

SDK for Java 2.x

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
    public static void createDBClusterParameterGroup(RdsClient rdsClient, String
 dbClusterGroupName,
            String dbParameterGroupFamily) {
        try {
```

```
            CreateDbClusterParameterGroupRequest groupRequest =
 CreateDbClusterParameterGroupRequest.builder()
                    .dbClusterParameterGroupName(dbClusterGroupName)
                    .dbParameterGroupFamily(dbParameterGroupFamily)
                    .description("Created by using the AWS SDK for Java")
                    .build();

            CreateDbClusterParameterGroupResponse response =
 rdsClient.createDBClusterParameterGroup(groupRequest);
            System.out.println("The group name is " +
 response.dbClusterParameterGroup().dbClusterParameterGroupName());

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- 有关 API 详细信息，请参阅《AWS SDK for Java 2.x API 参考》中的 CreateDBClusterParameterGroup。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
suspend fun createDBClusterParameterGroup(
    dbClusterGroupNameVal: String?,
    dbParameterGroupFamilyVal: String?,
) {
    val groupRequest =
        CreateDbClusterParameterGroupRequest {
            dbClusterParameterGroupName = dbClusterGroupNameVal
            dbParameterGroupFamily = dbParameterGroupFamilyVal
            description = "Created by using the AWS SDK for Kotlin"
```

```
        }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbClusterParameterGroup(groupRequest)
        println("The group name is
 ${response.dbClusterParameterGroup?.dbClusterParameterGroupName}")
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Kotlin API 参考》中的 CreateDBClusterParameterGroup。

Python

SDK for Python (Boto3)

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)
```

```
    def create_parameter_group(
        self, parameter_group_name, parameter_group_family, description
    ):
        """
        Creates a DB cluster parameter group that is based on the specified
parameter group
        family.

        :param parameter_group_name: The name of the newly created parameter
group.
        :param parameter_group_family: The family that is used as the basis of
the new
                                       parameter group.
        :param description: A description given to the parameter group.
        :return: Data about the newly created parameter group.
        """
        try:
            response = self.rds_client.create_db_cluster_parameter_group(
                DBClusterParameterGroupName=parameter_group_name,
                DBParameterGroupFamily=parameter_group_family,
                Description=description,
            )
        except ClientError as err:
            logger.error(
                "Couldn't create parameter group %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response
```

- 有关 API 详细信息，请参阅《AWS SDK for Python (Boto3) API 参考》中的 CreateDBClusterParameterGroup。

Rust

适用于 Rust 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
   // Select an engine family and create a custom DB cluster parameter group.
rds.CreateDbClusterParameterGroup(DBParameterGroupFamily='aurora-mysql8.0')
   pub async fn set_engine(&mut self, engine: &str, version: &str) -> Result<(),
ScenarioError> {
       self.engine_family = Some(engine.to_string());
       self.engine_version = Some(version.to_string());
       let create_db_cluster_parameter_group = self
           .rds
           .create_db_cluster_parameter_group(
               DB_CLUSTER_PARAMETER_GROUP_NAME,
               DB_CLUSTER_PARAMETER_GROUP_DESCRIPTION,
               engine,
           )
           .await;

       match create_db_cluster_parameter_group {
           Ok(CreateDbClusterParameterGroupOutput {
               db_cluster_parameter_group: None,
               ..
           }) => {
               return Err(ScenarioError::with(
                   "CreateDBClusterParameterGroup had empty response",
               ));
           }
           Err(error) => {
               if error.code() == Some("DBParameterGroupAlreadyExists") {
                   info!("Cluster Parameter Group already exists, nothing to
do");
               } else {
                   return Err(ScenarioError::new(
                       "Could not create Cluster Parameter Group",
                       &error,
```

```
                    ));
                }
            }
            _ => {
                info!("Created Cluster Parameter Group");
            }
        }

        Ok(())
    }

    pub async fn create_db_cluster_parameter_group(
        &self,
        name: &str,
        description: &str,
        family: &str,
    ) -> Result<CreateDbClusterParameterGroupOutput,
 SdkError<CreateDBClusterParameterGroupError>>
    {
        self.inner
            .create_db_cluster_parameter_group()
            .db_cluster_parameter_group_name(name)
            .description(description)
            .db_parameter_group_family(family)
            .send()
            .await
    }

#[tokio::test]
async fn test_scenario_set_engine() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .with(
            eq("RustSDKCodeExamplesDBParameterGroup"),
            eq("Parameter Group created by Rust SDK Code Example"),
            eq("aurora-mysql"),
        )
        .return_once(|_, _, _| {
            Ok(CreateDbClusterParameterGroupOutput::builder()

 .db_cluster_parameter_group(DbClusterParameterGroup::builder().build())
                .build())
```

```
        });

    let mut scenario = AuroraScenario::new(mock_rds);

    let set_engine = scenario.set_engine("aurora-mysql", "aurora-
mysql8.0").await;

    assert_eq!(set_engine, Ok(()));
    assert_eq!(Some("aurora-mysql"), scenario.engine_family.as_deref());
    assert_eq!(Some("aurora-mysql8.0"), scenario.engine_version.as_deref());
}

#[tokio::test]
async fn test_scenario_set_engine_not_create() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .with(
            eq("RustSDKCodeExamplesDBParameterGroup"),
            eq("Parameter Group created by Rust SDK Code Example"),
            eq("aurora-mysql"),
        )
        .return_once(|_, _, _|
 Ok(CreateDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);

    let set_engine = scenario.set_engine("aurora-mysql", "aurora-
mysql8.0").await;

    assert!(set_engine.is_err());
}

#[tokio::test]
async fn test_scenario_set_engine_param_group_exists() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .withf(|_, _, _| true)
        .return_once(|_, _, _| {
            Err(SdkError::service_error(
```

```
 CreateDBClusterParameterGroupError::DbParameterGroupAlreadyExistsFault(
                    DbParameterGroupAlreadyExistsFault::builder().build(),
                ),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);

    let set_engine = scenario.set_engine("aurora-mysql", "aurora-
mysql8.0").await;

    assert!(set_engine.is_err());
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Rust API 参考》中的 CreateDBClusterParameterGroup。

有关 AWS SDK 开发人员指南和代码示例的完整列表，请参阅 将此服务与 AWS SDK 结合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

将 **CreateDBClusterSnapshot** 与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 CreateDBClusterSnapshot。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 开始使用数据库集群

.NET

适用于 .NET 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
    /// <summary>
    /// Create a snapshot of a cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <param name="snapshotIdentifier">Identifier for the snapshot.</param>
    /// <returns>DB snapshot object.</returns>
    public async Task<DBClusterSnapshot>
CreateClusterSnapshotByIdentifierAsync(string dbClusterIdentifier, string
snapshotIdentifier)
    {
        var response = await _amazonRDS.CreateDBClusterSnapshotAsync(
            new CreateDBClusterSnapshotRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                DBClusterSnapshotIdentifier = snapshotIdentifier,
            });

        return response.DBClusterSnapshot;
    }
```

- 有关 API 详细信息，请参阅《适用于 .NET 的 AWS SDK API 参考》中的 CreateDBClusterSnapshot。

C++

SDK for C++

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

            Aws::RDS::Model::CreateDBClusterSnapshotRequest request;
            request.SetDBClusterIdentifier(DB_CLUSTER_IDENTIFIER);
            request.SetDBClusterSnapshotIdentifier(snapshotID);

            Aws::RDS::Model::CreateDBClusterSnapshotOutcome outcome =
                    client.CreateDBClusterSnapshot(request);

            if (outcome.IsSuccess()) {
                std::cout << "Snapshot creation has started."
                          << std::endl;
            }
            else {
                std::cerr << "Error with Aurora::CreateDBClusterSnapshot. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                                 DB_CLUSTER_IDENTIFIER, DB_INSTANCE_IDENTIFIER,
client);
                return false;
            }
```

- 有关 API 详细信息，请参阅《适用于 C++ 的 AWS SDK API 参考》中的 CreateDBClusterSnapshot。

Go

适用于 Go V2 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
type DbClusters struct {
 AuroraClient *rds.Client
}



// CreateClusterSnapshot creates a snapshot of a DB cluster.
func (clusters *DbClusters) CreateClusterSnapshot(clusterName string,
 snapshotName string) (
 *types.DBClusterSnapshot, error) {
 output, err := clusters.AuroraClient.CreateDBClusterSnapshot(context.TODO(),
 &rds.CreateDBClusterSnapshotInput{
  DBClusterIdentifier:         aws.String(clusterName),
  DBClusterSnapshotIdentifier: aws.String(snapshotName),
 })
 if err != nil {
  log.Printf("Couldn't create snapshot %v: %v\n", snapshotName, err)
  return nil, err
 } else {
  return output.DBClusterSnapshot, nil
 }
}
```

- 有关 API 详细信息，请参阅《适用于 Go 的 AWS SDK API 参考》中的 CreateDBClusterSnapshot。

Java

SDK for Java 2.x

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
   public static void createDBClusterSnapshot(RdsClient rdsClient, String
dbInstanceClusterIdentifier,
           String dbSnapshotIdentifier) {
       try {
           CreateDbClusterSnapshotRequest snapshotRequest =
CreateDbClusterSnapshotRequest.builder()
                   .dbClusterIdentifier(dbInstanceClusterIdentifier)
                   .dbClusterSnapshotIdentifier(dbSnapshotIdentifier)
                   .build();

           CreateDbClusterSnapshotResponse response =
rdsClient.createDBClusterSnapshot(snapshotRequest);
           System.out.println("The Snapshot ARN is " +
response.dbClusterSnapshot().dbClusterSnapshotArn());

       } catch (RdsException e) {
           System.out.println(e.getLocalizedMessage());
           System.exit(1);
       }
   }
```

- 有关 API 详细信息，请参阅《AWS SDK for Java 2.x API 参考》中的 CreateDBClusterSnapshot。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
suspend fun createDBClusterSnapshot(
    dbInstanceClusterIdentifier: String?,
    dbSnapshotIdentifier: String?,
) {
    val snapshotRequest =
        CreateDbClusterSnapshotRequest {
            dbClusterIdentifier = dbInstanceClusterIdentifier
            dbClusterSnapshotIdentifier = dbSnapshotIdentifier
        }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbClusterSnapshot(snapshotRequest)
        println("The Snapshot ARN is
 ${response.dbClusterSnapshot?.dbClusterSnapshotArn}")
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Kotlin API 参考》中的 CreateDBClusterSnapshot。

Python

SDK for Python (Boto3)

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def create_cluster_snapshot(self, snapshot_id, cluster_id):
        """
        Creates a snapshot of a DB cluster.

        :param snapshot_id: The ID to give the created snapshot.
        :param cluster_id: The DB cluster to snapshot.
        :return: Data about the newly created snapshot.
        """
        try:
            response = self.rds_client.create_db_cluster_snapshot(
                DBClusterSnapshotIdentifier=snapshot_id,
 DBClusterIdentifier=cluster_id
            )
            snapshot = response["DBClusterSnapshot"]
        except ClientError as err:
            logger.error(
                "Couldn't create snapshot of %s. Here's why: %s: %s",
                cluster_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return snapshot
```

- 有关 API 详细信息，请参阅《AWS SDK for Python (Boto3) API 参考》中的 CreateDBClusterSnapshot。

Rust

适用于 Rust 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
   // Get a list of allowed engine versions.
rds.DescribeDbEngineVersions(Engine='aurora-mysql', DBParameterGroupFamily=<the
family used to create your parameter group in step 2>)
   // Create an Aurora DB cluster database cluster that contains a MySql
database and uses the parameter group you created.
   // Wait for DB cluster to be ready. Call rds.DescribeDBClusters and check for
Status == 'available'.
   // Get a list of instance classes available for the selected engine
and engine version. rds.DescribeOrderableDbInstanceOptions(Engine='mysql',
EngineVersion=).

   // Create a database instance in the cluster.
   // Wait for DB instance to be ready. Call rds.DescribeDbInstances and check
for DBInstanceStatus == 'available'.
   pub async fn start_cluster_and_instance(&mut self) -> Result<(),
ScenarioError> {
       if self.password.is_none() {
           return Err(ScenarioError::with(
               "Must set Secret Password before starting a cluster",
           ));
       }
       let create_db_cluster = self
           .rds
           .create_db_cluster(
               DB_CLUSTER_IDENTIFIER,
```

```
                DB_CLUSTER_PARAMETER_GROUP_NAME,
                DB_ENGINE,
                self.engine_version.as_deref().expect("engine version"),
                self.username.as_deref().expect("username"),
                self.password
                    .replace(SecretString::new("".to_string()))
                    .expect("password"),
            )
            .await;
        if let Err(err) = create_db_cluster {
            return Err(ScenarioError::new(
                "Failed to create DB Cluster with cluster group",
                &err,
            ));
        }

        self.db_cluster_identifier = create_db_cluster
            .unwrap()
            .db_cluster
            .and_then(|c| c.db_cluster_identifier);

        if self.db_cluster_identifier.is_none() {
            return Err(ScenarioError::with("Created DB Cluster missing
Identifier"));
        }

        info!(
            "Started a db cluster: {}",
            self.db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing ARN")
        );

        let create_db_instance = self
            .rds
            .create_db_instance(
                self.db_cluster_identifier.as_deref().expect("cluster name"),
                DB_INSTANCE_IDENTIFIER,
                self.instance_class.as_deref().expect("instance class"),
                DB_ENGINE,
            )
            .await;
        if let Err(err) = create_db_instance {
            return Err(ScenarioError::new(
```

```
                "Failed to create Instance in DB Cluster",
                &err,
            ));
        }

        self.db_instance_identifier = create_db_instance
            .unwrap()
            .db_instance
            .and_then(|i| i.db_instance_identifier);

        // Cluster creation can take up to 20 minutes to become available
        let cluster_max_wait = Duration::from_secs(20 * 60);
        let waiter = Waiter::builder().max(cluster_max_wait).build();
        while waiter.sleep().await.is_ok() {
            let cluster = self
                .rds
                .describe_db_clusters(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = cluster {
                warn!(?err, "Failed to describe cluster while waiting for
ready");
                continue;
            }

            let instance = self
                .rds
                .describe_db_instance(
                    self.db_instance_identifier
                        .as_deref()
                        .expect("instance identifier"),
                )
                .await;
            if let Err(err) = instance {
                return Err(ScenarioError::new(
                    "Failed to find instance for cluster",
                    &err,
                ));
            }
```

```
            let instances_available = instance
                .unwrap()
                .db_instances()
                .iter()
                .all(|instance| instance.db_instance_status() ==
Some("Available"));

            let endpoints = self
                .rds
                .describe_db_cluster_endpoints(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = endpoints {
                return Err(ScenarioError::new(
                    "Failed to find endpoint for cluster",
                    &err,
                ));
            }

            let endpoints_available = endpoints
                .unwrap()
                .db_cluster_endpoints()
                .iter()
                .all(|endpoint| endpoint.status() == Some("available"));

            if instances_available && endpoints_available {
                return Ok(());
            }
        }

        Err(ScenarioError::with("timed out waiting for cluster"))
    }

    pub async fn snapshot_cluster(
        &self,
        db_cluster_identifier: &str,
        snapshot_name: &str,
    ) -> Result<CreateDbClusterSnapshotOutput,
SdkError<CreateDBClusterSnapshotError>> {
        self.inner
```

```
            .create_db_cluster_snapshot()
            .db_cluster_identifier(db_cluster_identifier)
            .db_cluster_snapshot_identifier(snapshot_name)
            .send()
            .await
    }

#[tokio::test]
async fn test_start_cluster_and_instance() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
```

```
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|id| {
            Ok(DescribeDbClustersOutput::builder()

.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_describe_db_instance()
        .with(eq("RustSDKCodeExamplesDBInstance"))
        .return_once(|name| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_instance_identifier(name)
                        .db_instance_status("Available")
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

.db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
```

```
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
        assert!(scenario
            .password
            .replace(SecretString::new("BAD SECRET".into()))
            .unwrap()
            .expose_secret()
            .is_empty());
        assert_eq!(
            scenario.db_cluster_identifier,
            Some("RustSDKCodeExamplesDBCluster".into())
        );
    });
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Err(SdkError::service_error(
                CreateDBClusterError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));
```

```
    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _}) if message
 == "Failed to create DB Cluster with cluster group")
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_missing_id() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()
                .db_cluster(DbCluster::builder().build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context:_ }) if message
 == "Created DB Cluster missing Identifier");
}

#[tokio::test]
async fn test_start_cluster_and_instance_instance_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
```

```
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .return_once(|_, _, _, _| {
            Err(SdkError::service_error(
                CreateDBInstanceError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db instance error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _ }) if message
 == "Failed to create Instance in DB Cluster")
}

#[tokio::test]
async fn test_start_cluster_and_instance_wait_hiccup() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
```

```
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

.db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        })
```

```
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()

.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds.expect_describe_db_instance().return_once(|name| {
        Ok(DescribeDbInstancesOutput::builder()
            .db_instances(
                DbInstance::builder()
                    .db_instance_identifier(name)
                    .db_instance_status("Available")
                    .build(),
            )
            .build())
    });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

.db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
```

```
    let _ = assertions.await;
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Rust API 参考》中的 CreateDBClusterSnapshot。

有关 AWS SDK 开发人员指南和代码示例的完整列表，请参阅 将此服务与 AWS SDK 结合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

将 `CreateDBInstance` 与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `CreateDBInstance`。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 开始使用数据库集群

.NET

适用于 .NET 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
    /// <summary>
    /// Create an Amazon Relational Database Service (Amazon RDS) DB instance
    /// with a particular set of properties. Use the action
DescribeDBInstancesAsync
    /// to determine when the DB instance is ready to use.
    /// </summary>
    /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <param name="dbEngine">The engine for the DB instance.</param>
    /// <param name="dbEngineVersion">Version for the DB instance.</param>
    /// <param name="instanceClass">Class for the DB instance.</param>
```

```
    /// <returns>DB instance object.</returns>
    public async Task<DBInstance> CreateDBInstanceInClusterAsync(
        string dbClusterIdentifier,
        string dbInstanceIdentifier,
        string dbEngine,
        string dbEngineVersion,
        string instanceClass)
    {
        // When creating the instance within a cluster, do not specify the name
or size.
        var response = await _amazonRDS.CreateDBInstanceAsync(
            new CreateDBInstanceRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                DBInstanceIdentifier = dbInstanceIdentifier,
                Engine = dbEngine,
                EngineVersion = dbEngineVersion,
                DBInstanceClass = instanceClass
            });

        return response.DBInstance;
    }
```

- 有关 API 详细信息，请参阅《适用于 .NET 的 AWS SDK API 参考》中的 CreateDBInstance。

C++

SDK for C++

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";
```

```
    Aws::RDS::RDSClient client(clientConfig);

        Aws::RDS::Model::CreateDBInstanceRequest request;
        request.SetDBInstanceIdentifier(DB_INSTANCE_IDENTIFIER);
        request.SetDBClusterIdentifier(DB_CLUSTER_IDENTIFIER);
        request.SetEngine(engineName);
        request.SetDBInstanceClass(dbInstanceClass);

        Aws::RDS::Model::CreateDBInstanceOutcome outcome =
                client.CreateDBInstance(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB instance creation has started."
                      << std::endl;
        }
        else {
            std::cerr << "Error with RDS::CreateDBInstance. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, DB_CLUSTER_IDENTIFIER,
"",
                             client);
            return false;
        }
```

- 有关 API 详细信息，请参阅《适用于 C++ 的 AWS SDK API 参考》中的 CreateDBInstance。

Go

适用于 Go V2 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
type DbClusters struct {
```

```
 AuroraClient *rds.Client
}



// CreateInstanceInCluster creates a database instance in an existing DB cluster.
 The first database that is
// created defaults to a read-write DB instance.
func (clusters *DbClusters) CreateInstanceInCluster(clusterName string,
 instanceName string,
 dbEngine string, dbInstanceClass string) (*types.DBInstance, error) {
 output, err := clusters.AuroraClient.CreateDBInstance(context.TODO(),
 &rds.CreateDBInstanceInput{
  DBInstanceIdentifier: aws.String(instanceName),
  DBClusterIdentifier:  aws.String(clusterName),
  Engine:               aws.String(dbEngine),
  DBInstanceClass:      aws.String(dbInstanceClass),
 })
 if err != nil {
  log.Printf("Couldn't create instance %v: %v\n", instanceName, err)
  return nil, err
 } else {
  return output.DBInstance, nil
 }
}
```

- 有关 API 详细信息，请参阅《适用于 Go 的 AWS SDK API 参考》中的 CreateDBInstance。

Java

SDK for Java 2.x

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
    public static String createDBInstanceCluster(RdsClient rdsClient,
            String dbInstanceIdentifier,
```

```
            String dbInstanceClusterIdentifier,
            String instanceClass) {
        try {
            CreateDbInstanceRequest instanceRequest =
 CreateDbInstanceRequest.builder()
                    .dbInstanceIdentifier(dbInstanceIdentifier)
                    .dbClusterIdentifier(dbInstanceClusterIdentifier)
                    .engine("aurora-mysql")
                    .dbInstanceClass(instanceClass)
                    .build();

            CreateDbInstanceResponse response =
 rdsClient.createDBInstance(instanceRequest);
            System.out.print("The status is " +
 response.dbInstance().dbInstanceStatus());
            return response.dbInstance().dbInstanceArn();

        } catch (RdsException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        return "";
    }
```

- 有关 API 详细信息，请参阅《AWS SDK for Java 2.x API 参考》中的 CreateDBInstance。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
suspend fun createDBInstanceCluster(
    dbInstanceIdentifierVal: String?,
    dbInstanceClusterIdentifierVal: String?,
    instanceClassVal: String?,
): String? {
```

```
    val instanceRequest =
        CreateDbInstanceRequest {
            dbInstanceIdentifier = dbInstanceIdentifierVal
            dbClusterIdentifier = dbInstanceClusterIdentifierVal
            engine = "aurora-mysql"
            dbInstanceClass = instanceClassVal
        }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbInstance(instanceRequest)
        print("The status is ${response.dbInstance?.dbInstanceStatus}")
        return response.dbInstance?.dbInstanceArn
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Kotlin API 参考》中的 CreateDBInstance。

Python

SDK for Python (Boto3)

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
```

```
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def create_instance_in_cluster(
        self, instance_id, cluster_id, db_engine, instance_class
    ):
        """
        Creates a database instance in an existing DB cluster. The first database
that is
        created defaults to a read-write DB instance.

        :param instance_id: The ID to give the newly created DB instance.
        :param cluster_id: The ID of the DB cluster where the DB instance is
created.
        :param db_engine: The database engine of a database to create in the DB
instance.
                          This must be compatible with the configured parameter
group
                          of the DB cluster.
        :param instance_class: The DB instance class for the newly created DB
instance.
        :return: Data about the newly created DB instance.
        """
        try:
            response = self.rds_client.create_db_instance(
                DBInstanceIdentifier=instance_id,
                DBClusterIdentifier=cluster_id,
                Engine=db_engine,
                DBInstanceClass=instance_class,
            )
            db_inst = response["DBInstance"]
        except ClientError as err:
            logger.error(
                "Couldn't create DB instance %s. Here's why: %s: %s",
                instance_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return db_inst
```

- 有关 API 详细信息，请参阅《AWS SDK for Python (Boto3) API 参考》中的 CreateDBInstance。

Rust

适用于 Rust 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
   // Get a list of allowed engine versions.
rds.DescribeDbEngineVersions(Engine='aurora-mysql', DBParameterGroupFamily=<the
family used to create your parameter group in step 2>)
   // Create an Aurora DB cluster database cluster that contains a MySql
database and uses the parameter group you created.
   // Wait for DB cluster to be ready. Call rds.DescribeDBClusters and check for
Status == 'available'.
   // Get a list of instance classes available for the selected engine
and engine version. rds.DescribeOrderableDbInstanceOptions(Engine='mysql',
EngineVersion=).

   // Create a database instance in the cluster.
   // Wait for DB instance to be ready. Call rds.DescribeDbInstances and check
for DBInstanceStatus == 'available'.
   pub async fn start_cluster_and_instance(&mut self) -> Result<(),
ScenarioError> {
       if self.password.is_none() {
           return Err(ScenarioError::with(
               "Must set Secret Password before starting a cluster",
           ));
       }
       let create_db_cluster = self
           .rds
           .create_db_cluster(
               DB_CLUSTER_IDENTIFIER,
               DB_CLUSTER_PARAMETER_GROUP_NAME,
```

```
                DB_ENGINE,
                self.engine_version.as_deref().expect("engine version"),
                self.username.as_deref().expect("username"),
                self.password
                    .replace(SecretString::new("".to_string()))
                    .expect("password"),
            )
            .await;
        if let Err(err) = create_db_cluster {
            return Err(ScenarioError::new(
                "Failed to create DB Cluster with cluster group",
                &err,
            ));
        }

        self.db_cluster_identifier = create_db_cluster
            .unwrap()
            .db_cluster
            .and_then(|c| c.db_cluster_identifier);

        if self.db_cluster_identifier.is_none() {
            return Err(ScenarioError::with("Created DB Cluster missing
Identifier"));
        }

        info!(
            "Started a db cluster: {}",
            self.db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing ARN")
        );

        let create_db_instance = self
            .rds
            .create_db_instance(
                self.db_cluster_identifier.as_deref().expect("cluster name"),
                DB_INSTANCE_IDENTIFIER,
                self.instance_class.as_deref().expect("instance class"),
                DB_ENGINE,
            )
            .await;
        if let Err(err) = create_db_instance {
            return Err(ScenarioError::new(
                "Failed to create Instance in DB Cluster",
```

```
                &err,
            ));
        }

        self.db_instance_identifier = create_db_instance
            .unwrap()
            .db_instance
            .and_then(|i| i.db_instance_identifier);

        // Cluster creation can take up to 20 minutes to become available
        let cluster_max_wait = Duration::from_secs(20 * 60);
        let waiter = Waiter::builder().max(cluster_max_wait).build();
        while waiter.sleep().await.is_ok() {
            let cluster = self
                .rds
                .describe_db_clusters(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = cluster {
                warn!(?err, "Failed to describe cluster while waiting for
ready");
                continue;
            }

            let instance = self
                .rds
                .describe_db_instance(
                    self.db_instance_identifier
                        .as_deref()
                        .expect("instance identifier"),
                )
                .await;
            if let Err(err) = instance {
                return Err(ScenarioError::new(
                    "Failed to find instance for cluster",
                    &err,
                ));
            }

            let instances_available = instance
```

```
                .unwrap()
                .db_instances()
                .iter()
                .all(|instance| instance.db_instance_status() ==
Some("Available"));

            let endpoints = self
                .rds
                .describe_db_cluster_endpoints(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = endpoints {
                return Err(ScenarioError::new(
                    "Failed to find endpoint for cluster",
                    &err,
                ));
            }

            let endpoints_available = endpoints
                .unwrap()
                .db_cluster_endpoints()
                .iter()
                .all(|endpoint| endpoint.status() == Some("available"));

            if instances_available && endpoints_available {
                return Ok(());
            }
        }

        Err(ScenarioError::with("timed out waiting for cluster"))
    }

    pub async fn create_db_instance(
        &self,
        cluster_name: &str,
        instance_name: &str,
        instance_class: &str,
        engine: &str,
    ) -> Result<CreateDbInstanceOutput, SdkError<CreateDBInstanceError>> {
        self.inner
```

```
            .create_db_instance()
            .db_cluster_identifier(cluster_name)
            .db_instance_identifier(instance_name)
            .db_instance_class(instance_class)
            .engine(engine)
            .send()
            .await
    }

#[tokio::test]
async fn test_start_cluster_and_instance() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
```

```
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|id| {
            Ok(DescribeDbClustersOutput::builder()

.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_describe_db_instance()
        .with(eq("RustSDKCodeExamplesDBInstance"))
        .return_once(|name| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_instance_identifier(name)
                        .db_instance_status("Available")
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

.db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
```

```
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
        assert!(scenario
            .password
            .replace(SecretString::new("BAD SECRET".into()))
            .unwrap()
            .expose_secret()
            .is_empty());
        assert_eq!(
            scenario.db_cluster_identifier,
            Some("RustSDKCodeExamplesDBCluster".into())
        );
    });
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Err(SdkError::service_error(
                CreateDBClusterError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
```

```
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _}) if message
 == "Failed to create DB Cluster with cluster group")
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_missing_id() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()
                .db_cluster(DbCluster::builder().build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context:_ }) if message
 == "Created DB Cluster missing Identifier");
}

#[tokio::test]
async fn test_start_cluster_and_instance_instance_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
```

```
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .return_once(|_, _, _, _| {
            Err(SdkError::service_error(
                CreateDBInstanceError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db instance error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _ }) if message
 == "Failed to create Instance in DB Cluster")
}

#[tokio::test]
async fn test_start_cluster_and_instance_wait_hiccup() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
```

```
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

.db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
```

```
            ))
        })
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()

.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds.expect_describe_db_instance().return_once(|name| {
        Ok(DescribeDbInstancesOutput::builder()
            .db_instances(
                DbInstance::builder()
                    .db_instance_identifier(name)
                    .db_instance_status("Available")
                    .build(),
            )
            .build())
    });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

.db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await;
```

```
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Rust API 参考》中的 CreateDBInstance。

有关 AWS SDK 开发人员指南和代码示例的完整列表，请参阅 将此服务与 AWS SDK 结合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

将 **DeleteDBCluster** 与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 DeleteDBCluster。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 开始使用数据库集群

.NET

适用于 .NET 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
    /// <summary>
    /// Delete a particular DB cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <returns>DB cluster object.</returns>
    public async Task<DBCluster> DeleteDBClusterByIdentifierAsync(string
dbClusterIdentifier)
    {
        var response = await _amazonRDS.DeleteDBClusterAsync(
            new DeleteDBClusterRequest()
```

```
            {
                DBClusterIdentifier = dbClusterIdentifier,
                SkipFinalSnapshot = true
            });

        return response.DBCluster;
    }
```

- 有关 API 详细信息，请参阅《适用于 .NET 的 AWS SDK API 参考》中的 DeleteDBCluster。

C++

SDK for C++

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

            Aws::RDS::Model::DeleteDBClusterRequest request;
            request.SetDBClusterIdentifier(dbClusterIdentifier);
            request.SetSkipFinalSnapshot(true);

            Aws::RDS::Model::DeleteDBClusterOutcome outcome =
                    client.DeleteDBCluster(request);

            if (outcome.IsSuccess()) {
                std::cout << "DB cluster deletion has started."
                          << std::endl;
                clusterDeleting = true;
                std::cout
                        << "Waiting for DB cluster to delete before deleting the
parameter group."
                        << std::endl;
```

```
                std::cout << "This may take a while." << std::endl;
            }
            else {
                std::cerr << "Error with Aurora::DeleteDBCluster. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                result = false;
            }
```

- 有关 API 详细信息，请参阅《适用于 C++ 的 AWS SDK API 参考》中的 DeleteDBCluster。

Go

适用于 Go V2 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
type DbClusters struct {
 AuroraClient *rds.Client
}



// DeleteDbCluster deletes a DB cluster without keeping a final snapshot.
func (clusters *DbClusters) DeleteDbCluster(clusterName string) error {
 _, err := clusters.AuroraClient.DeleteDBCluster(context.TODO(),
 &rds.DeleteDBClusterInput{
  DBClusterIdentifier: aws.String(clusterName),
  SkipFinalSnapshot:   true,
 })
 if err != nil {
  log.Printf("Couldn't delete DB cluster %v: %v\n", clusterName, err)
  return err
 } else {
  return nil
 }
```

```
}
```

- 有关 API 详细信息，请参阅《适用于 Go 的 AWS SDK API 参考》中的 DeleteDBCluster。

Java

SDK for Java 2.x

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
   public static void deleteCluster(RdsClient rdsClient, String
dbInstanceClusterIdentifier) {
       try {
           DeleteDbClusterRequest deleteDbClusterRequest =
DeleteDbClusterRequest.builder()
                   .dbClusterIdentifier(dbInstanceClusterIdentifier)
                   .skipFinalSnapshot(true)
                   .build();

           rdsClient.deleteDBCluster(deleteDbClusterRequest);
           System.out.println(dbInstanceClusterIdentifier + " was deleted!");

       } catch (RdsException e) {
           System.out.println(e.getLocalizedMessage());
           System.exit(1);
       }
   }
```

- 有关 API 详细信息，请参阅《AWS SDK for Java 2.x API 参考》中的 DeleteDBCluster。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
suspend fun deleteCluster(dbInstanceClusterIdentifier: String) {
    val deleteDbClusterRequest =
        DeleteDbClusterRequest {
            dbClusterIdentifier = dbInstanceClusterIdentifier
            skipFinalSnapshot = true
        }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        rdsClient.deleteDbCluster(deleteDbClusterRequest)
        println("$dbInstanceClusterIdentifier was deleted!")
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Kotlin API 参考》中的 DeleteDBCluster。

Python

SDK for Python (Boto3)

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
```

```
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def delete_db_cluster(self, cluster_name):
        """
        Deletes a DB cluster.

        :param cluster_name: The name of the DB cluster to delete.
        """
        try:
            self.rds_client.delete_db_cluster(
                DBClusterIdentifier=cluster_name, SkipFinalSnapshot=True
            )
            logger.info("Deleted DB cluster %s.", cluster_name)
        except ClientError:
            logger.exception("Couldn't delete DB cluster %s.", cluster_name)
            raise
```

- 有关 API 详细信息，请参阅《AWS SDK for Python (Boto3) API 参考》中的 DeleteDBCluster。

Rust

适用于 Rust 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
    pub async fn clean_up(self) -> Result<(), Vec<ScenarioError>> {
        let mut clean_up_errors: Vec<ScenarioError> = vec![];

        // Delete the instance. rds.DeleteDbInstance.
        let delete_db_instance = self
            .rds
            .delete_db_instance(
                self.db_instance_identifier
                    .as_deref()
                    .expect("instance identifier"),
            )
            .await;
        if let Err(err) = delete_db_instance {
            let identifier = self
                .db_instance_identifier
                .as_deref()
                .unwrap_or("Missing Instance Identifier");
            let message = format!("failed to delete db instance {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance to delete
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_instances =
self.rds.describe_db_instances().await;
                if let Err(err) = describe_db_instances {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check instance state during deletion",
                        &err,
                    ));
                    break;
                }
```

```
                let db_instances = describe_db_instances
                    .unwrap()
                    .db_instances()
                    .iter()
                    .filter(|instance| instance.db_cluster_identifier ==
self.db_cluster_identifier)
                    .cloned()
                    .collect::<Vec<DbInstance>>();

                if db_instances.is_empty() {
                    trace!("Delete Instance waited and no instances were found");
                    break;
                }
                match db_instances.first().unwrap().db_instance_status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but instances is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB instance");
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster. rds.DeleteDbCluster.
        let delete_db_cluster = self
            .rds
            .delete_db_cluster(
                self.db_cluster_identifier
                    .as_deref()
                    .expect("cluster identifier"),
            )
            .await;

        if let Err(err) = delete_db_cluster {
            let identifier = self
                .db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing DB Cluster Identifier");
            let message = format!("failed to delete db cluster {identifier}");
```

```
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance and cluster to fully delete.
rds.DescribeDbInstances and rds.DescribeDbClusters until both are not found.
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_clusters = self
                    .rds
                    .describe_db_clusters(
                        self.db_cluster_identifier
                            .as_deref()
                            .expect("cluster identifier"),
                    )
                    .await;
                if let Err(err) = describe_db_clusters {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check cluster state during deletion",
                        &err,
                    ));
                    break;
                }
                let describe_db_clusters = describe_db_clusters.unwrap();
                let db_clusters = describe_db_clusters.db_clusters();
                if db_clusters.is_empty() {
                    trace!("Delete cluster waited and no clusters were found");
                    break;
                }
                match db_clusters.first().unwrap().status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but clusters is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB cluster");
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster parameter group.
rds.DeleteDbClusterParameterGroup.
```

```
        let delete_db_cluster_parameter_group = self
            .rds
            .delete_db_cluster_parameter_group(
                self.db_cluster_parameter_group
                    .map(|g| {
                        g.db_cluster_parameter_group_name
                            .unwrap_or_else(||
 DB_CLUSTER_PARAMETER_GROUP_NAME.to_string())
                    })
                    .as_deref()
                    .expect("cluster parameter group name"),
            )
            .await;
        if let Err(error) = delete_db_cluster_parameter_group {
            clean_up_errors.push(ScenarioError::new(
                "Failed to delete the db cluster parameter group",
                &error,
            ))
        }

        if clean_up_errors.is_empty() {
            Ok(())
        } else {
            Err(clean_up_errors)
        }
    }

    pub async fn delete_db_cluster(
        &self,
        cluster_identifier: &str,
    ) -> Result<DeleteDbClusterOutput, SdkError<DeleteDBClusterError>> {
        self.inner
            .delete_db_cluster()
            .db_cluster_identifier(cluster_identifier)
            .skip_final_snapshot(true)
            .send()
            .await
    }

#[tokio::test]
async fn test_scenario_clean_up() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
```

```
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| Ok(DescribeDbInstancesOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| Ok(DescribeDbClustersOutput::builder().build()));
```

```
    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
 Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_scenario_clean_up_errors() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
```

```
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| {
            Err(SdkError::service_error(
                DescribeDBInstancesError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db instances error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
```

```
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db clusters error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_err());
        let errs = clean_up.unwrap_err();
        assert_eq!(errs.len(), 2);
        assert_matches!(errs.get(0), Some(ScenarioError {message, context: _}) if
message == "Failed to check instance state during deletion");
        assert_matches!(errs.get(1), Some(ScenarioError {message, context: _}) if
message == "Failed to check cluster state during deletion");
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
Describe Instances
```

```
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Rust API 参考》中的 DeleteDBCluster。

有关 AWS SDK 开发人员指南和代码示例的完整列表，请参阅 将此服务与 AWS SDK 结合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

将 **DeleteDBClusterParameterGroup** 与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 DeleteDBClusterParameterGroup。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 开始使用数据库集群

.NET

适用于 .NET 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
    /// <summary>
    /// Delete a particular parameter group by name.
    /// </summary>
    /// <param name="groupName">The name of the parameter group.</param>
    /// <returns>True if successful.</returns>
```

```
    public async Task<bool> DeleteClusterParameterGroupByNameAsync(string
groupName)
    {
        var request = new DeleteDBClusterParameterGroupRequest
        {
            DBClusterParameterGroupName = groupName,
        };

        var response = await
_amazonRDS.DeleteDBClusterParameterGroupAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- 有关 API 详细信息，请参阅《适用于 .NET 的 AWS SDK API 参考》中的 DeleteDBClusterParameterGroup。

C++

SDK for C++

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

        Aws::RDS::Model::DeleteDBClusterParameterGroupRequest request;
        request.SetDBClusterParameterGroupName(parameterGroupName);

        Aws::RDS::Model::DeleteDBClusterParameterGroupOutcome outcome =
                client.DeleteDBClusterParameterGroup(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB parameter group was successfully deleted."
```

```
                    << std::endl;
        }
        else {
            std::cerr << "Error with Aurora::DeleteDBClusterParameterGroup. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            result = false;
        }
```

- 有关 API 详细信息，请参阅《适用于 C++ 的 AWS SDK API 参考》中的 DeleteDBClusterParameterGroup。

Go

适用于 Go V2 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
type DbClusters struct {
 AuroraClient *rds.Client
}



// DeleteParameterGroup deletes the named DB cluster parameter group.
func (clusters *DbClusters) DeleteParameterGroup(parameterGroupName string) error
 {
 _, err := clusters.AuroraClient.DeleteDBClusterParameterGroup(context.TODO(),
  &rds.DeleteDBClusterParameterGroupInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
  })
 if err != nil {
  log.Printf("Couldn't delete parameter group %v: %v\n", parameterGroupName, err)
  return err
 } else {
```

```
  return nil
 }
}
```

- 有关 API 详细信息，请参阅《适用于 Go 的 AWS SDK API 参考》中的 DeleteDBClusterParameterGroup。

Java

SDK for Java 2.x

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
   public static void deleteDBClusterGroup(RdsClient rdsClient, String
dbClusterGroupName, String clusterDBARN)
           throws InterruptedException {
       try {
           boolean isDataDel = false;
           boolean didFind;
           String instanceARN;

           // Make sure that the database has been deleted.
           while (!isDataDel) {
               DescribeDbInstancesResponse response =
rdsClient.describeDBInstances();
               List<DBInstance> instanceList = response.dbInstances();
               int listSize = instanceList.size();
               didFind = false;
               int index = 1;
               for (DBInstance instance : instanceList) {
                   instanceARN = instance.dbInstanceArn();
                   if (instanceARN.compareTo(clusterDBARN) == 0) {
                       System.out.println(clusterDBARN + " still exists");
                       didFind = true;
                   }
```

```
                    if ((index == listSize) && (!didFind)) {
                        // Went through the entire list and did not find the
database ARN.
                        isDataDel = true;
                    }
                    Thread.sleep(sleepTime * 1000);
                    index++;
                }
            }

            DeleteDbClusterParameterGroupRequest clusterParameterGroupRequest =
DeleteDbClusterParameterGroupRequest
                    .builder()
                    .dbClusterParameterGroupName(dbClusterGroupName)
                    .build();


rdsClient.deleteDBClusterParameterGroup(clusterParameterGroupRequest);
            System.out.println(dbClusterGroupName + " was deleted.");

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- 有关 API 详细信息，请参阅《AWS SDK for Java 2.x API 参考》中的 DeleteDBClusterParameterGroup。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
@Throws(InterruptedException::class)
```

```
suspend fun deleteDBClusterGroup(
    dbClusterGroupName: String,
    clusterDBARN: String,
) {
    var isDataDel = false
    var didFind: Boolean
    var instanceARN: String

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        // Make sure that the database has been deleted.
        while (!isDataDel) {
            val response = rdsClient.describeDbInstances()
            val instanceList = response.dbInstances
            val listSize = instanceList?.size
            isDataDel = false
            didFind = false
            var index = 1
            if (instanceList != null) {
                for (instance in instanceList) {
                    instanceARN = instance.dbInstanceArn.toString()
                    if (instanceARN.compareTo(clusterDBARN) == 0) {
                        println("$clusterDBARN still exists")
                        didFind = true
                    }
                    if (index == listSize && !didFind) {
                        // Went through the entire list and did not find the
 database ARN.
                        isDataDel = true
                    }
                    delay(slTime * 1000)
                    index++
                }
            }
        }
        val clusterParameterGroupRequest =
            DeleteDbClusterParameterGroupRequest {
                dbClusterParameterGroupName = dbClusterGroupName
            }

        rdsClient.deleteDbClusterParameterGroup(clusterParameterGroupRequest)
        println("$dbClusterGroupName was deleted.")
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Kotlin API 参考》中的 DeleteDBClusterParameterGroup。

Python

SDK for Python (Boto3)

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def delete_parameter_group(self, parameter_group_name):
        """
        Deletes a DB cluster parameter group.

        :param parameter_group_name: The name of the parameter group to delete.
        :return: Data about the parameter group.
        """
```

```
        try:
            response = self.rds_client.delete_db_cluster_parameter_group(
                DBClusterParameterGroupName=parameter_group_name
            )
        except ClientError as err:
            logger.error(
                "Couldn't delete parameter group %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response
```

- 有关 API 详细信息，请参阅《AWS SDK for Python (Boto3) API 参考》中的 DeleteDBClusterParameterGroup。

Rust

适用于 Rust 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
    pub async fn clean_up(self) -> Result<(), Vec<ScenarioError>> {
        let mut clean_up_errors: Vec<ScenarioError> = vec![];

        // Delete the instance. rds.DeleteDbInstance.
        let delete_db_instance = self
            .rds
            .delete_db_instance(
                self.db_instance_identifier
                    .as_deref()
                    .expect("instance identifier"),
            )
```

```
            .await;
        if let Err(err) = delete_db_instance {
            let identifier = self
                .db_instance_identifier
                .as_deref()
                .unwrap_or("Missing Instance Identifier");
            let message = format!("failed to delete db instance {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance to delete
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_instances =
self.rds.describe_db_instances().await;
                if let Err(err) = describe_db_instances {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check instance state during deletion",
                        &err,
                    ));
                    break;
                }
                let db_instances = describe_db_instances
                    .unwrap()
                    .db_instances()
                    .iter()
                    .filter(|instance| instance.db_cluster_identifier ==
self.db_cluster_identifier)
                    .cloned()
                    .collect::<Vec<DbInstance>>();

                if db_instances.is_empty() {
                    trace!("Delete Instance waited and no instances were found");
                    break;
                }
                match db_instances.first().unwrap().db_instance_status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but instances is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB instance");
                        break;
```

```
                    }
                }
            }
        }

        // Delete the DB cluster. rds.DeleteDbCluster.
        let delete_db_cluster = self
            .rds
            .delete_db_cluster(
                self.db_cluster_identifier
                    .as_deref()
                    .expect("cluster identifier"),
            )
            .await;

        if let Err(err) = delete_db_cluster {
            let identifier = self
                .db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing DB Cluster Identifier");
            let message = format!("failed to delete db cluster {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance and cluster to fully delete.
rds.DescribeDbInstances and rds.DescribeDbClusters until both are not found.
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_clusters = self
                    .rds
                    .describe_db_clusters(
                        self.db_cluster_identifier
                            .as_deref()
                            .expect("cluster identifier"),
                    )
                    .await;
                if let Err(err) = describe_db_clusters {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check cluster state during deletion",
                        &err,
                    ));
                    break;
                }
                let describe_db_clusters = describe_db_clusters.unwrap();
                let db_clusters = describe_db_clusters.db_clusters();
```

```
                if db_clusters.is_empty() {
                    trace!("Delete cluster waited and no clusters were found");
                    break;
                }
                match db_clusters.first().unwrap().status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but clusters is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB cluster");
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster parameter group.
rds.DeleteDbClusterParameterGroup.
        let delete_db_cluster_parameter_group = self
            .rds
            .delete_db_cluster_parameter_group(
                self.db_cluster_parameter_group
                    .map(|g| {
                        g.db_cluster_parameter_group_name
                            .unwrap_or_else(||
DB_CLUSTER_PARAMETER_GROUP_NAME.to_string())
                    })
                    .as_deref()
                    .expect("cluster parameter group name"),
            )
            .await;
        if let Err(error) = delete_db_cluster_parameter_group {
            clean_up_errors.push(ScenarioError::new(
                "Failed to delete the db cluster parameter group",
                &error,
            ))
        }

        if clean_up_errors.is_empty() {
            Ok(())
        } else {
```

```
            Err(clean_up_errors)
        }
    }

    pub async fn delete_db_cluster_parameter_group(
        &self,
        name: &str,
    ) -> Result<DeleteDbClusterParameterGroupOutput,
 SdkError<DeleteDBClusterParameterGroupError>>
    {
        self.inner
            .delete_db_cluster_parameter_group()
            .db_cluster_parameter_group_name(name)
            .send()
            .await
    }

#[tokio::test]
async fn test_scenario_clean_up() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| Ok(DescribeDbInstancesOutput::builder().build()));
```

```
    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| Ok(DescribeDbClustersOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_ok());
    });
```

```
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_scenario_clean_up_errors() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| {
            Err(SdkError::service_error(
                DescribeDBInstancesError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db instances error",
                ))),
```

```
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db clusters error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
```

```
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_err());
        let errs = clean_up.unwrap_err();
        assert_eq!(errs.len(), 2);
        assert_matches!(errs.get(0), Some(ScenarioError {message, context: _}) if
 message == "Failed to check instance state during deletion");
        assert_matches!(errs.get(1), Some(ScenarioError {message, context: _}) if
 message == "Failed to check cluster state during deletion");
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}
```

- 有关 API 详细信息，请参阅适用于 Rust 的 AWS SDK API 参考中的 DeleteDBClusterParameterGroup。

有关 AWS SDK 开发人员指南和代码示例的完整列表，请参阅 将此服务与 AWS SDK 结合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

将 **DeleteDBInstance** 与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 DeleteDBInstance。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 开始使用数据库集群

.NET

适用于 .NET 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
    /// <summary>
    /// Delete a particular DB instance.
    /// </summary>
    /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
    /// <returns>DB instance object.</returns>
    public async Task<DBInstance> DeleteDBInstanceByIdentifierAsync(string
dbInstanceIdentifier)
    {
        var response = await _amazonRDS.DeleteDBInstanceAsync(
            new DeleteDBInstanceRequest()
            {
                DBInstanceIdentifier = dbInstanceIdentifier,
                SkipFinalSnapshot = true,
                DeleteAutomatedBackups = true
            });

        return response.DBInstance;
    }
```

- 有关 API 详细信息，请参阅《适用于 .NET 的 AWS SDK API 参考》中的 DeleteDBInstance。

C++

SDK for C++

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

            Aws::RDS::Model::DeleteDBInstanceRequest request;
            request.SetDBInstanceIdentifier(dbInstanceIdentifier);
            request.SetSkipFinalSnapshot(true);
            request.SetDeleteAutomatedBackups(true);

            Aws::RDS::Model::DeleteDBInstanceOutcome outcome =
                    client.DeleteDBInstance(request);

            if (outcome.IsSuccess()) {
                std::cout << "DB instance deletion has started."
                          << std::endl;
                instanceDeleting = true;
                std::cout
                        << "Waiting for DB instance to delete before deleting the
parameter group."
                        << std::endl;
            }
            else {
                std::cerr << "Error with Aurora::DeleteDBInstance. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                result = false;
            }
```

- 有关 API 详细信息，请参阅《适用于 C++ 的 AWS SDK API 参考》中的 DeleteDBInstance。

Go

适用于 Go V2 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
type DbClusters struct {
 AuroraClient *rds.Client
}



// DeleteInstance deletes a DB instance.
func (clusters *DbClusters) DeleteInstance(instanceName string) error {
 _, err := clusters.AuroraClient.DeleteDBInstance(context.TODO(),
 &rds.DeleteDBInstanceInput{
  DBInstanceIdentifier:   aws.String(instanceName),
  SkipFinalSnapshot:      true,
  DeleteAutomatedBackups: aws.Bool(true),
 })
 if err != nil {
  log.Printf("Couldn't delete instance %v: %v\n", instanceName, err)
  return err
 } else {
  return nil
 }
}
```

- 有关 API 详细信息，请参阅《适用于 Go 的 AWS SDK API 参考》中的 DeleteDBInstance。

Java

SDK for Java 2.x

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
   public static void deleteDatabaseInstance(RdsClient rdsClient, String
dbInstanceIdentifier) {
       try {
           DeleteDbInstanceRequest deleteDbInstanceRequest =
DeleteDbInstanceRequest.builder()
                   .dbInstanceIdentifier(dbInstanceIdentifier)
                   .deleteAutomatedBackups(true)
                   .skipFinalSnapshot(true)
                   .build();

           DeleteDbInstanceResponse response =
rdsClient.deleteDBInstance(deleteDbInstanceRequest);
           System.out.println("The status of the database is " +
response.dbInstance().dbInstanceStatus());

       } catch (RdsException e) {
           System.out.println(e.getLocalizedMessage());
           System.exit(1);
       }
   }
```

- 有关 API 详细信息，请参阅《AWS SDK for Java 2.x API 参考》中的 DeleteDBInstance。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
suspend fun deleteDBInstance(dbInstanceIdentifierVal: String) {
    val deleteDbInstanceRequest =
        DeleteDbInstanceRequest {
            dbInstanceIdentifier = dbInstanceIdentifierVal
            deleteAutomatedBackups = true
            skipFinalSnapshot = true
        }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.deleteDbInstance(deleteDbInstanceRequest)
        print("The status of the database is
 ${response.dbInstance?.dbInstanceStatus}")
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Kotlin API 参考》中的 DeleteDBInstance。

Python

SDK for Python (Boto3)

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""
```

```
    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def delete_db_instance(self, instance_id):
        """
        Deletes a DB instance.

        :param instance_id: The ID of the DB instance to delete.
        :return: Data about the deleted DB instance.
        """
        try:
            response = self.rds_client.delete_db_instance(
                DBInstanceIdentifier=instance_id,
                SkipFinalSnapshot=True,
                DeleteAutomatedBackups=True,
            )
            db_inst = response["DBInstance"]
        except ClientError as err:
            logger.error(
                "Couldn't delete DB instance %s. Here's why: %s: %s",
                instance_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return db_inst
```

- 有关 API 详细信息，请参阅《AWS SDK for Python (Boto3) API 参考》中的 DeleteDBInstance。

Rust

适用于 Rust 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
    pub async fn clean_up(self) -> Result<(), Vec<ScenarioError>> {
        let mut clean_up_errors: Vec<ScenarioError> = vec![];

        // Delete the instance. rds.DeleteDbInstance.
        let delete_db_instance = self
            .rds
            .delete_db_instance(
                self.db_instance_identifier
                    .as_deref()
                    .expect("instance identifier"),
            )
            .await;
        if let Err(err) = delete_db_instance {
            let identifier = self
                .db_instance_identifier
                .as_deref()
                .unwrap_or("Missing Instance Identifier");
            let message = format!("failed to delete db instance {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance to delete
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_instances =
self.rds.describe_db_instances().await;
                if let Err(err) = describe_db_instances {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check instance state during deletion",
```

```
                        &err,
                    ));
                    break;
                }
                let db_instances = describe_db_instances
                    .unwrap()
                    .db_instances()
                    .iter()
                    .filter(|instance| instance.db_cluster_identifier ==
self.db_cluster_identifier)
                    .cloned()
                    .collect::<Vec<DbInstance>>();

                if db_instances.is_empty() {
                    trace!("Delete Instance waited and no instances were found");
                    break;
                }
                match db_instances.first().unwrap().db_instance_status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but instances is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB instance");
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster. rds.DeleteDbCluster.
        let delete_db_cluster = self
            .rds
            .delete_db_cluster(
                self.db_cluster_identifier
                    .as_deref()
                    .expect("cluster identifier"),
            )
            .await;

        if let Err(err) = delete_db_cluster {
            let identifier = self
```

```
                .db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing DB Cluster Identifier");
            let message = format!("failed to delete db cluster {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance and cluster to fully delete.
rds.DescribeDbInstances and rds.DescribeDbClusters until both are not found.
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_clusters = self
                    .rds
                    .describe_db_clusters(
                        self.db_cluster_identifier
                            .as_deref()
                            .expect("cluster identifier"),
                    )
                    .await;
                if let Err(err) = describe_db_clusters {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check cluster state during deletion",
                        &err,
                    ));
                    break;
                }
                let describe_db_clusters = describe_db_clusters.unwrap();
                let db_clusters = describe_db_clusters.db_clusters();
                if db_clusters.is_empty() {
                    trace!("Delete cluster waited and no clusters were found");
                    break;
                }
                match db_clusters.first().unwrap().status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but clusters is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB cluster");
                        break;
                    }
                }
            }
```

```
        }

        // Delete the DB cluster parameter group.
 rds.DeleteDbClusterParameterGroup.
        let delete_db_cluster_parameter_group = self
            .rds
            .delete_db_cluster_parameter_group(
                self.db_cluster_parameter_group
                    .map(|g| {
                        g.db_cluster_parameter_group_name
                            .unwrap_or_else(||
 DB_CLUSTER_PARAMETER_GROUP_NAME.to_string())
                    })
                    .as_deref()
                    .expect("cluster parameter group name"),
            )
            .await;
        if let Err(error) = delete_db_cluster_parameter_group {
            clean_up_errors.push(ScenarioError::new(
                "Failed to delete the db cluster parameter group",
                &error,
            ))
        }

        if clean_up_errors.is_empty() {
            Ok(())
        } else {
            Err(clean_up_errors)
        }
    }

    pub async fn delete_db_instance(
        &self,
        instance_identifier: &str,
    ) -> Result<DeleteDbInstanceOutput, SdkError<DeleteDBInstanceError>> {
        self.inner
            .delete_db_instance()
            .db_instance_identifier(instance_identifier)
            .skip_final_snapshot(true)
            .send()
            .await
    }

#[tokio::test]
```

```
async fn test_scenario_clean_up() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| Ok(DescribeDbInstancesOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
```

```
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| Ok(DescribeDbClustersOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
 Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_scenario_clean_up_errors() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
```

```
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| {
            Err(SdkError::service_error(
                DescribeDBInstancesError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db instances error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
```

```
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db clusters error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_err());
        let errs = clean_up.unwrap_err();
        assert_eq!(errs.len(), 2);
        assert_matches!(errs.get(0), Some(ScenarioError {message, context: _}) if
message == "Failed to check instance state during deletion");
        assert_matches!(errs.get(1), Some(ScenarioError {message, context: _}) if
message == "Failed to check cluster state during deletion");
    });
```

```
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Rust API 参考》中的 DeleteDBInstance。

有关 AWS SDK 开发人员指南和代码示例的完整列表，请参阅 将此服务与 AWS SDK 结合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

将 **DescribeDBClusterParameterGroups** 与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 DescribeDBClusterParameterGroups。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 开始使用数据库集群

.NET

适用于 .NET 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
    /// <summary>
```

```
    /// Get the description of a DB cluster parameter group by name.
    /// </summary>
    /// <param name="name">The name of the DB parameter group to describe.</
param>
    /// <returns>The parameter group description.</returns>
    public async Task<DBClusterParameterGroup?>
 DescribeCustomDBClusterParameterGroupAsync(string name)
    {
        var response = await _amazonRDS.DescribeDBClusterParameterGroupsAsync(
            new DescribeDBClusterParameterGroupsRequest()
            {
                DBClusterParameterGroupName = name
            });
        return response.DBClusterParameterGroups.FirstOrDefault();
    }
```

- 有关 API 详细信息，请参阅《适用于 .NET 的 AWS SDK API 参考》中的 DescribeDBClusterParameterGroups。

C++

SDK for C++

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

        Aws::RDS::Model::DescribeDBClusterParameterGroupsRequest request;
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);

        Aws::RDS::Model::DescribeDBClusterParameterGroupsOutcome outcome =
                client.DescribeDBClusterParameterGroups(request);
```

```
        if (outcome.IsSuccess()) {
            std::cout << "DB cluster parameter group named '" <<
                      CLUSTER_PARAMETER_GROUP_NAME << "' already exists." <<
 std::endl;
            dbParameterGroupFamily =
 outcome.GetResult().GetDBClusterParameterGroups()
[0].GetDBParameterGroupFamily();
        }

        else {
            std::cerr << "Error with Aurora::DescribeDBClusterParameterGroups. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
```

- 有关 API 详细信息，请参阅《适用于 C++ 的 AWS SDK API 参考》中的 DescribeDBClusterParameterGroups。

Go

适用于 Go V2 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
type DbClusters struct {
 AuroraClient *rds.Client
}



// GetParameterGroup gets a DB cluster parameter group by name.
func (clusters *DbClusters) GetParameterGroup(parameterGroupName string) (
 *types.DBClusterParameterGroup, error) {
```

```
 output, err := clusters.AuroraClient.DescribeDBClusterParameterGroups(
  context.TODO(), &rds.DescribeDBClusterParameterGroupsInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
  })
 if err != nil {
  var notFoundError *types.DBParameterGroupNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("Parameter group %v does not exist.\n", parameterGroupName)
   err = nil
  } else {
   log.Printf("Error getting parameter group %v: %v\n", parameterGroupName, err)
  }
  return nil, err
 } else {
  return &output.DBClusterParameterGroups[0], err
 }
}
```

- 有关 API 详细信息，请参阅《适用于 Go 的 AWS SDK API 参考》中的 DescribeDBClusterParameterGroups。

Java

SDK for Java 2.x

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
   public static void describeDbClusterParameterGroups(RdsClient rdsClient,
String dbClusterGroupName) {
       try {
           DescribeDbClusterParameterGroupsRequest groupsRequest =
DescribeDbClusterParameterGroupsRequest.builder()
                   .dbClusterParameterGroupName(dbClusterGroupName)
                   .maxRecords(20)
                   .build();
```

```
            List<DBClusterParameterGroup> groups =
 rdsClient.describeDBClusterParameterGroups(groupsRequest)
                    .dbClusterParameterGroups();
            for (DBClusterParameterGroup group : groups) {
                System.out.println("The group name is " +
 group.dbClusterParameterGroupName());
                System.out.println("The group ARN is " +
 group.dbClusterParameterGroupArn());
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- 有关 API 详细信息，请参阅《AWS SDK for Java 2.x API 参考》中的 DescribeDBClusterParameterGroups。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
suspend fun describeDbClusterParameterGroups(dbClusterGroupName: String?) {
    val groupsRequest =
        DescribeDbClusterParameterGroupsRequest {
            dbClusterParameterGroupName = dbClusterGroupName
            maxRecords = 20
        }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.describeDbClusterParameterGroups(groupsRequest)
        response.dbClusterParameterGroups?.forEach { group ->
```

```
            println("The group name is ${group.dbClusterParameterGroupName}")
            println("The group ARN is ${group.dbClusterParameterGroupArn}")
        }
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Kotlin API 参考》中的 DescribeDBClusterParameterGroups。

Python

SDK for Python (Boto3)

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def get_parameter_group(self, parameter_group_name):
        """
```

```
        Gets a DB cluster parameter group.

        :param parameter_group_name: The name of the parameter group to retrieve.
        :return: The requested parameter group.
        """
        try:
            response = self.rds_client.describe_db_cluster_parameter_groups(
                DBClusterParameterGroupName=parameter_group_name
            )
            parameter_group = response["DBClusterParameterGroups"][0]
        except ClientError as err:
            if err.response["Error"]["Code"] == "DBParameterGroupNotFound":
                logger.info("Parameter group %s does not exist.",
parameter_group_name)
            else:
                logger.error(
                    "Couldn't get parameter group %s. Here's why: %s: %s",
                    parameter_group_name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            return parameter_group
```

- 有关 API 详细信息，请参阅《适用于 Python 的 AWS SDK（Boto3）API 参考》中的 DescribeDBClusterParameterGroups。

有关 AWS SDK 开发人员指南和代码示例的完整列表，请参阅 将此服务与 AWS SDK 结合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

将 **DescribeDBClusterParameters** 与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 DescribeDBClusterParameters。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 开始使用数据库集群

.NET

适用于 .NET 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
    /// <summary>
    /// Describe the cluster parameters in a parameter group.
    /// </summary>
    /// <param name="groupName">The name of the parameter group.</param>
    /// <param name="source">The optional name of the source filter.</param>
    /// <returns>The collection of parameters.</returns>
    public async Task<List<Parameter>>
DescribeDBClusterParametersInGroupAsync(string groupName, string? source = null)
    {
        var paramList = new List<Parameter>();

        DescribeDBClusterParametersResponse response;
        var request = new DescribeDBClusterParametersRequest
        {
            DBClusterParameterGroupName = groupName,
            Source = source,
        };

        // Get the full list if there are multiple pages.
        do
        {
            response = await
_amazonRDS.DescribeDBClusterParametersAsync(request);
            paramList.AddRange(response.Parameters);

            request.Marker = response.Marker;
        }
        while (response.Marker is not null);

        return paramList;
    }
```

- 有关 API 详细信息，请参阅《适用于 .NET 的 AWS SDK API 参考》中的 DescribeDBClusterParameters。

C++

SDK for C++

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);


//! Routine which gets DB parameters using the 'DescribeDBClusterParameters' api.
/*!
 \sa getDBClusterParameters()
 \param parameterGroupName: The name of the cluster parameter group.
 \param namePrefix: Prefix string to filter results by parameter name.
 \param source: A source such as 'user', ignored if empty.
 \param parametersResult: Vector of 'Parameter' objects returned by the routine.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::getDBClusterParameters(const Aws::String
 &parameterGroupName,
                                            const Aws::String &namePrefix,
                                            const Aws::String &source,

 Aws::Vector<Aws::RDS::Model::Parameter> &parametersResult,
                                            const Aws::RDS::RDSClient &client) {
    Aws::String marker; // The marker is used for pagination.
    do {
        Aws::RDS::Model::DescribeDBClusterParametersRequest request;
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);
```

```
        if (!marker.empty()) {
            request.SetMarker(marker);
        }
        if (!source.empty()) {
            request.SetSource(source);
        }

        Aws::RDS::Model::DescribeDBClusterParametersOutcome outcome =
                client.DescribeDBClusterParameters(request);

        if (outcome.IsSuccess()) {
            const Aws::Vector<Aws::RDS::Model::Parameter> &parameters =
                    outcome.GetResult().GetParameters();
            for (const Aws::RDS::Model::Parameter &parameter: parameters) {
                if (!namePrefix.empty()) {
                    if (parameter.GetParameterName().find(namePrefix) == 0) {
                        parametersResult.push_back(parameter);
                    }
                }
                else {
                    parametersResult.push_back(parameter);
                }
            }

            marker = outcome.GetResult().GetMarker();
        }
        else {
            std::cerr << "Error with Aurora::DescribeDBClusterParameters. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    } while (!marker.empty());

    return true;
}
```

- 有关 API 详细信息，请参阅《适用于 C++ 的 AWS SDK API 参考》中的 DescribeDBClusterParameters。

Go

适用于 Go V2 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
type DbClusters struct {
 AuroraClient *rds.Client
}



// GetParameters gets the parameters that are contained in a DB cluster parameter
 group.
func (clusters *DbClusters) GetParameters(parameterGroupName string, source
 string) (
 []types.Parameter, error) {

 var output *rds.DescribeDBClusterParametersOutput
 var params []types.Parameter
 var err error
 parameterPaginator :=
 rds.NewDescribeDBClusterParametersPaginator(clusters.AuroraClient,
  &rds.DescribeDBClusterParametersInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
   Source:                      aws.String(source),
  })
 for parameterPaginator.HasMorePages() {
  output, err = parameterPaginator.NextPage(context.TODO())
  if err != nil {
   log.Printf("Couldn't get paramaeters for %v: %v\n", parameterGroupName, err)
   break
  } else {
   params = append(params, output.Parameters...)
  }
 }
 return params, err
```

```
}
```

- 有关 API 详细信息，请参阅《适用于 Go 的 AWS SDK API 参考》中的 DescribeDBClusterParameters。

Java

SDK for Java 2.x

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
   public static void describeDbClusterParameters(RdsClient rdsClient, String
dbCLusterGroupName, int flag) {
       try {
           DescribeDbClusterParametersRequest dbParameterGroupsRequest;
           if (flag == 0) {
               dbParameterGroupsRequest =
DescribeDbClusterParametersRequest.builder()
                       .dbClusterParameterGroupName(dbCLusterGroupName)
                       .build();
           } else {
               dbParameterGroupsRequest =
DescribeDbClusterParametersRequest.builder()
                       .dbClusterParameterGroupName(dbCLusterGroupName)
                       .source("user")
                       .build();
           }

           DescribeDbClusterParametersResponse response = rdsClient
                   .describeDBClusterParameters(dbParameterGroupsRequest);
           List<Parameter> dbParameters = response.parameters();
           String paraName;
           for (Parameter para : dbParameters) {
               // Only print out information about either auto_increment_offset
or
```

```
                // auto_increment_increment.
                paraName = para.parameterName();
                if ((paraName.compareTo("auto_increment_offset") == 0)
                        || (paraName.compareTo("auto_increment_increment ") ==
0)) {
                    System.out.println("*** The parameter name is  " + paraName);
                    System.out.println("*** The parameter value is  " +
para.parameterValue());
                    System.out.println("*** The parameter data type is " +
para.dataType());
                    System.out.println("*** The parameter description is " +
para.description());
                    System.out.println("*** The parameter allowed values  is " +
para.allowedValues());
                }
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- 有关 API 详细信息，请参阅《AWS SDK for Java 2.x API 参考》中的 DescribeDBClusterParameters。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
suspend fun describeDbClusterParameters(
    dbCLusterGroupName: String?,
    flag: Int,
) {
```

```
    val dbParameterGroupsRequest: DescribeDbClusterParametersRequest
    dbParameterGroupsRequest =
        if (flag == 0) {
            DescribeDbClusterParametersRequest {
                dbClusterParameterGroupName = dbCLusterGroupName
            }
        } else {
            DescribeDbClusterParametersRequest {
                dbClusterParameterGroupName = dbCLusterGroupName
                source = "user"
            }
        }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response =
 rdsClient.describeDbClusterParameters(dbParameterGroupsRequest)
        response.parameters?.forEach { para ->
            // Only print out information about either auto_increment_offset or
 auto_increment_increment.
            val paraName = para.parameterName
            if (paraName != null) {
                if (paraName.compareTo("auto_increment_offset") == 0 ||
 paraName.compareTo("auto_increment_increment ") == 0) {
                    println("*** The parameter name is  $paraName")
                    println("*** The parameter value is  ${para.parameterValue}")
                    println("*** The parameter data type is ${para.dataType}")
                    println("*** The parameter description is
 ${para.description}")
                    println("*** The parameter allowed values  is
 ${para.allowedValues}")
                }
            }
        }
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Kotlin API 参考》中的 DescribeDBClusterParameters。

Python

SDK for Python (Boto3)

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def get_parameters(self, parameter_group_name, name_prefix="", source=None):
        """
        Gets the parameters that are contained in a DB cluster parameter group.

        :param parameter_group_name: The name of the parameter group to query.
        :param name_prefix: When specified, the retrieved list of parameters is
 filtered
                            to contain only parameters that start with this
 prefix.
        :param source: When specified, only parameters from this source are
 retrieved.
                       For example, a source of 'user' retrieves only parameters
 that
```

```
                        were set by a user.
        :return: The list of requested parameters.
        """
        try:
            kwargs = {"DBClusterParameterGroupName": parameter_group_name}
            if source is not None:
                kwargs["Source"] = source
            parameters = []
            paginator =
self.rds_client.get_paginator("describe_db_cluster_parameters")
            for page in paginator.paginate(**kwargs):
                parameters += [
                    p
                    for p in page["Parameters"]
                    if p["ParameterName"].startswith(name_prefix)
                ]
        except ClientError as err:
            logger.error(
                "Couldn't get parameters for %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return parameters
```

- 有关 API 详细信息，请参阅《AWS SDK for Python (Boto3) API 参考》中的 DescribeDBClusterParameters。

Rust

适用于 Rust 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
   // Get the parameter group. rds.DescribeDbClusterParameterGroups
   // Get parameters in the group. This is a long list so you will have to
paginate. Find the auto_increment_offset and auto_increment_increment parameters
(by ParameterName). rds.DescribeDbClusterParameters
   // Parse the ParameterName, Description, and AllowedValues values and display
them.
   pub async fn cluster_parameters(&self) ->
Result<Vec<AuroraScenarioParameter>, ScenarioError> {
       let parameters_output = self
           .rds
           .describe_db_cluster_parameters(DB_CLUSTER_PARAMETER_GROUP_NAME)
           .await;

       if let Err(err) = parameters_output {
           return Err(ScenarioError::new(
               format!("Failed to retrieve parameters for
{DB_CLUSTER_PARAMETER_GROUP_NAME}"),
               &err,
           ));
       }

       let parameters = parameters_output
           .unwrap()
           .into_iter()
           .flat_map(|p| p.parameters.unwrap_or_default().into_iter())
           .filter(|p|
FILTER_PARAMETER_NAMES.contains(p.parameter_name().unwrap_or_default()))
           .map(AuroraScenarioParameter::from)
           .collect::<Vec<_>>();

       Ok(parameters)
   }

   pub async fn describe_db_cluster_parameters(
       &self,
       name: &str,
   ) -> Result<Vec<DescribeDbClusterParametersOutput>,
SdkError<DescribeDBClusterParametersError>>
   {
       self.inner
           .describe_db_cluster_parameters()
           .db_cluster_parameter_group_name(name)
           .into_paginator()
```

```
            .send()
            .try_collect()
            .await
    }

#[tokio::test]
async fn test_scenario_cluster_parameters() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_cluster_parameters()
        .with(eq("RustSDKCodeExamplesDBParameterGroup"))
        .return_once(|_| {
            Ok(vec![DescribeDbClusterParametersOutput::builder()
                .parameters(Parameter::builder().parameter_name("a").build())
                .parameters(Parameter::builder().parameter_name("b").build())
                .parameters(
                    Parameter::builder()
                        .parameter_name("auto_increment_offset")
                        .build(),
                )
                .parameters(Parameter::builder().parameter_name("c").build())
                .parameters(
                    Parameter::builder()
                        .parameter_name("auto_increment_increment")
                        .build(),
                )
                .parameters(Parameter::builder().parameter_name("d").build())
                .build()])
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());

    let params = scenario.cluster_parameters().await.expect("cluster params");
    let names: Vec<String> = params.into_iter().map(|p| p.name).collect();
    assert_eq!(
        names,
        vec!["auto_increment_offset", "auto_increment_increment"]
    );
}

#[tokio::test]
async fn test_scenario_cluster_parameters_error() {
```

```
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_cluster_parameters()
        .with(eq("RustSDKCodeExamplesDBParameterGroup"))
        .return_once(|_| {
            Err(SdkError::service_error(
                DescribeDBClusterParametersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe_db_cluster_parameters_error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());
    let params = scenario.cluster_parameters().await;
    assert_matches!(params, Err(ScenarioError { message, context: _ }) if message
 == "Failed to retrieve parameters for RustSDKCodeExamplesDBParameterGroup");
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Rust API 参考》中的 DescribeDBClusterParameters。

有关 AWS SDK 开发人员指南和代码示例的完整列表，请参阅 将此服务与 AWS SDK 结合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

将 **DescribeDBClusterSnapshots** 与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 DescribeDBClusterSnapshots。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 开始使用数据库集群

.NET

适用于 .NET 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
    /// <summary>
    /// Return a list of DB snapshots for a particular DB cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <returns>List of DB snapshots.</returns>
    public async Task<List<DBClusterSnapshot>>
DescribeDBClusterSnapshotsByIdentifierAsync(string dbClusterIdentifier)
    {
        var results = new List<DBClusterSnapshot>();

        DescribeDBClusterSnapshotsResponse response;
        DescribeDBClusterSnapshotsRequest request = new
DescribeDBClusterSnapshotsRequest
        {
            DBClusterIdentifier = dbClusterIdentifier
        };
        // Get the full list if there are multiple pages.
        do
        {
            response = await _amazonRDS.DescribeDBClusterSnapshotsAsync(request);
            results.AddRange(response.DBClusterSnapshots);
            request.Marker = response.Marker;
        }
        while (response.Marker is not null);
        return results;
    }
```

- 有关 API 详细信息，请参阅《适用于 .NET 的 AWS SDK API 参考》中的 DescribeDBClusterSnapshots。

C++

SDK for C++

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

            Aws::RDS::Model::DescribeDBClusterSnapshotsRequest request;
            request.SetDBClusterSnapshotIdentifier(snapshotID);

            Aws::RDS::Model::DescribeDBClusterSnapshotsOutcome outcome =
                    client.DescribeDBClusterSnapshots(request);

            if (outcome.IsSuccess()) {
                snapshot = outcome.GetResult().GetDBClusterSnapshots()[0];
            }
            else {
                std::cerr << "Error with Aurora::DescribeDBClusterSnapshots. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                                 DB_CLUSTER_IDENTIFIER, DB_INSTANCE_IDENTIFIER,
client);
                return false;
            }
```

- 有关 API 详细信息，请参阅《适用于 C++ 的 AWS SDK API 参考》中的 DescribeDBClusterSnapshots。

Go

适用于 Go V2 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
type DbClusters struct {
 AuroraClient *rds.Client
}



// GetClusterSnapshot gets a DB cluster snapshot.
func (clusters *DbClusters) GetClusterSnapshot(snapshotName string)
 (*types.DBClusterSnapshot, error) {
 output, err := clusters.AuroraClient.DescribeDBClusterSnapshots(context.TODO(),
  &rds.DescribeDBClusterSnapshotsInput{
   DBClusterSnapshotIdentifier: aws.String(snapshotName),
  })
 if err != nil {
  log.Printf("Couldn't get snapshot %v: %v\n", snapshotName, err)
  return nil, err
 } else {
  return &output.DBClusterSnapshots[0], nil
 }
}
```

- 有关 API 详细信息，请参阅《适用于 Go 的 AWS SDK API 参考》中的 DescribeDBClusterSnapshots。

Java

SDK for Java 2.x

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
   public static void waitForSnapshotReady(RdsClient rdsClient, String
dbSnapshotIdentifier,
           String dbInstanceClusterIdentifier) {
       try {
           boolean snapshotReady = false;
           String snapshotReadyStr;
           System.out.println("Waiting for the snapshot to become available.");

           DescribeDbClusterSnapshotsRequest snapshotsRequest =
DescribeDbClusterSnapshotsRequest.builder()
                   .dbClusterSnapshotIdentifier(dbSnapshotIdentifier)
                   .dbClusterIdentifier(dbInstanceClusterIdentifier)
                   .build();

           while (!snapshotReady) {
               DescribeDbClusterSnapshotsResponse response =
rdsClient.describeDBClusterSnapshots(snapshotsRequest);
               List<DBClusterSnapshot> snapshotList =
response.dbClusterSnapshots();
               for (DBClusterSnapshot snapshot : snapshotList) {
                   snapshotReadyStr = snapshot.status();
                   if (snapshotReadyStr.contains("available")) {
                       snapshotReady = true;
                   } else {
                       System.out.println(".");
                       Thread.sleep(sleepTime * 5000);
                   }
               }
           }

           System.out.println("The Snapshot is available!");
```

```
        } catch (RdsException | InterruptedException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- 有关 API 详细信息，请参阅《AWS SDK for Java 2.x API 参考》中的 DescribeDBClusterSnapshots。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
suspend fun waitSnapshotReady(
    dbSnapshotIdentifier: String?,
    dbInstanceClusterIdentifier: String?,
) {
    var snapshotReady = false
    var snapshotReadyStr: String
    println("Waiting for the snapshot to become available.")

    val snapshotsRequest =
        DescribeDbClusterSnapshotsRequest {
            dbClusterSnapshotIdentifier = dbSnapshotIdentifier
            dbClusterIdentifier = dbInstanceClusterIdentifier
        }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        while (!snapshotReady) {
            val response = rdsClient.describeDbClusterSnapshots(snapshotsRequest)
            val snapshotList = response.dbClusterSnapshots
            if (snapshotList != null) {
                for (snapshot in snapshotList) {
                    snapshotReadyStr = snapshot.status.toString()
```

```
                    if (snapshotReadyStr.contains("available")) {
                        snapshotReady = true
                    } else {
                        println(".")
                        delay(slTime * 5000)
                    }
                }
            }
        }
    }
    println("The Snapshot is available!")
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Kotlin API 参考》中的 DescribeDBClusterSnapshots。

Python

SDK for Python (Boto3)

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
```

```
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def get_cluster_snapshot(self, snapshot_id):
        """
        Gets a DB cluster snapshot.

        :param snapshot_id: The ID of the snapshot to retrieve.
        :return: The retrieved snapshot.
        """
        try:
            response = self.rds_client.describe_db_cluster_snapshots(
                DBClusterSnapshotIdentifier=snapshot_id
            )
            snapshot = response["DBClusterSnapshots"][0]
        except ClientError as err:
            logger.error(
                "Couldn't get DB cluster snapshot %s. Here's why: %s: %s",
                snapshot_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return snapshot
```

- 有关 API 详细信息，请参阅《适用于 Python 的 AWS SDK（Boto3）API 参考》中的 DescribeDBClusterSnapshots。

有关 AWS SDK 开发人员指南和代码示例的完整列表，请参阅 将此服务与 AWS SDK 结合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

将 **DescribeDBClusters** 与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 DescribeDBClusters。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 开始使用数据库集群

.NET

适用于 .NET 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
    /// <summary>
    /// Returns a list of DB clusters.
    /// </summary>
    /// <param name="dbInstanceIdentifier">Optional name of a specific DB
cluster.</param>
    /// <returns>List of DB clusters.</returns>
    public async Task<List<DBCluster>> DescribeDBClustersPagedAsync(string?
dbClusterIdentifier = null)
    {
        var results = new List<DBCluster>();

        DescribeDBClustersResponse response;
        DescribeDBClustersRequest request = new DescribeDBClustersRequest
        {
            DBClusterIdentifier = dbClusterIdentifier
        };
        // Get the full list if there are multiple pages.
        do
        {
            response = await _amazonRDS.DescribeDBClustersAsync(request);
            results.AddRange(response.DBClusters);
            request.Marker = response.Marker;
        }
        while (response.Marker is not null);
        return results;
    }
```

- 有关 API 详细信息，请参阅《适用于 .NET 的 AWS SDK API 参考》中的 DescribeDBClusters。

C++

SDK for C++

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

//! Routine which gets a DB cluster description.
/*!
 \sa describeDBCluster()
 \param dbClusterIdentifier: A DB cluster identifier.
 \param clusterResult: The 'DBCluster' object containing the description.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::describeDBCluster(const Aws::String &dbClusterIdentifier,
                                       Aws::RDS::Model::DBCluster &clusterResult,
                                       const Aws::RDS::RDSClient &client) {
    Aws::RDS::Model::DescribeDBClustersRequest request;
    request.SetDBClusterIdentifier(dbClusterIdentifier);

    Aws::RDS::Model::DescribeDBClustersOutcome outcome =
            client.DescribeDBClusters(request);

    bool result = true;
    if (outcome.IsSuccess()) {
        clusterResult = outcome.GetResult().GetDBClusters()[0];
    }
    else if (outcome.GetError().GetErrorType() !=
```

```
            Aws::RDS::RDSErrors::D_B_CLUSTER_NOT_FOUND_FAULT) {
        result = false;
        std::cerr << "Error with Aurora::GDescribeDBClusters. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
    }
        // This example does not log an error if the DB cluster does not exist.
        // Instead, clusterResult is set to empty.
    else {
        clusterResult = Aws::RDS::Model::DBCluster();
    }

    return result;

}
```

- 有关 API 详细信息，请参阅《适用于 C++ 的 AWS SDK API 参考》中的 DescribeDBClusters。

Go

适用于 Go V2 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
type DbClusters struct {
 AuroraClient *rds.Client
}



// GetDbCluster gets data about an Aurora DB cluster.
func (clusters *DbClusters) GetDbCluster(clusterName string) (*types.DBCluster,
 error) {
 output, err := clusters.AuroraClient.DescribeDBClusters(context.TODO(),
```

```
  &rds.DescribeDBClustersInput{
   DBClusterIdentifier: aws.String(clusterName),
  })
 if err != nil {
  var notFoundError *types.DBClusterNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("DB cluster %v does not exist.\n", clusterName)
   err = nil
  } else {
   log.Printf("Couldn't get DB cluster %v: %v\n", clusterName, err)
  }
  return nil, err
 } else {
  return &output.DBClusters[0], err
 }
}
```

- 有关 API 详细信息，请参阅《适用于 Go 的 AWS SDK API 参考》中的 DescribeDBClusters。

Java

SDK for Java 2.x

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
   public static void describeDbClusterParameters(RdsClient rdsClient, String
dbCLusterGroupName, int flag) {
       try {
           DescribeDbClusterParametersRequest dbParameterGroupsRequest;
           if (flag == 0) {
               dbParameterGroupsRequest =
DescribeDbClusterParametersRequest.builder()
                       .dbClusterParameterGroupName(dbCLusterGroupName)
                       .build();
```

```
            } else {
                dbParameterGroupsRequest =
DescribeDbClusterParametersRequest.builder()
                        .dbClusterParameterGroupName(dbCLusterGroupName)
                        .source("user")
                        .build();
            }

            DescribeDbClusterParametersResponse response = rdsClient
                    .describeDBClusterParameters(dbParameterGroupsRequest);
            List<Parameter> dbParameters = response.parameters();
            String paraName;
            for (Parameter para : dbParameters) {
                // Only print out information about either auto_increment_offset
or
                // auto_increment_increment.
                paraName = para.parameterName();
                if ((paraName.compareTo("auto_increment_offset") == 0)
                        || (paraName.compareTo("auto_increment_increment ") ==
0)) {
                    System.out.println("*** The parameter name is  " + paraName);
                    System.out.println("*** The parameter value is  " +
para.parameterValue());
                    System.out.println("*** The parameter data type is " +
para.dataType());
                    System.out.println("*** The parameter description is " +
para.description());
                    System.out.println("*** The parameter allowed values  is " +
para.allowedValues());
                }
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- 有关 API 详细信息，请参阅《AWS SDK for Java 2.x API 参考》中的 DescribeDBClusters。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
suspend fun describeDbClusterParameters(
    dbCLusterGroupName: String?,
    flag: Int,
) {
    val dbParameterGroupsRequest: DescribeDbClusterParametersRequest
    dbParameterGroupsRequest =
        if (flag == 0) {
            DescribeDbClusterParametersRequest {
                dbClusterParameterGroupName = dbCLusterGroupName
            }
        } else {
            DescribeDbClusterParametersRequest {
                dbClusterParameterGroupName = dbCLusterGroupName
                source = "user"
            }
        }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response =
 rdsClient.describeDbClusterParameters(dbParameterGroupsRequest)
        response.parameters?.forEach { para ->
            // Only print out information about either auto_increment_offset or
 auto_increment_increment.
            val paraName = para.parameterName
            if (paraName != null) {
                if (paraName.compareTo("auto_increment_offset") == 0 ||
 paraName.compareTo("auto_increment_increment ") == 0) {
                    println("*** The parameter name is  $paraName")
                    println("*** The parameter value is  ${para.parameterValue}")
                    println("*** The parameter data type is ${para.dataType}")
                    println("*** The parameter description is
 ${para.description}")
```

```
                    println("*** The parameter allowed values  is
 ${para.allowedValues}")
                }
            }
        }
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Kotlin API 参考》中的 DescribeDBClusters。

Python

SDK for Python (Boto3)

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def get_db_cluster(self, cluster_name):
        """
```

```
        Gets data about an Aurora DB cluster.

        :param cluster_name: The name of the DB cluster to retrieve.
        :return: The retrieved DB cluster.
        """
        try:
            response = self.rds_client.describe_db_clusters(
                DBClusterIdentifier=cluster_name
            )
            cluster = response["DBClusters"][0]
        except ClientError as err:
            if err.response["Error"]["Code"] == "DBClusterNotFoundFault":
                logger.info("Cluster %s does not exist.", cluster_name)
            else:
                logger.error(
                    "Couldn't verify the existence of DB cluster %s. Here's why:
%s: %s",
                    cluster_name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            return cluster
```

- 有关 API 详细信息，请参阅《AWS SDK for Python (Boto3) API 参考》中的 DescribeDBClusters。

Rust

适用于 Rust 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
   // Get a list of allowed engine versions.
rds.DescribeDbEngineVersions(Engine='aurora-mysql', DBParameterGroupFamily=<the
family used to create your parameter group in step 2>)
   // Create an Aurora DB cluster database cluster that contains a MySql
database and uses the parameter group you created.
   // Wait for DB cluster to be ready. Call rds.DescribeDBClusters and check for
Status == 'available'.
   // Get a list of instance classes available for the selected engine
and engine version. rds.DescribeOrderableDbInstanceOptions(Engine='mysql',
EngineVersion=).

   // Create a database instance in the cluster.
   // Wait for DB instance to be ready. Call rds.DescribeDbInstances and check
for DBInstanceStatus == 'available'.
   pub async fn start_cluster_and_instance(&mut self) -> Result<(),
ScenarioError> {
       if self.password.is_none() {
           return Err(ScenarioError::with(
               "Must set Secret Password before starting a cluster",
           ));
       }
       let create_db_cluster = self
           .rds
           .create_db_cluster(
               DB_CLUSTER_IDENTIFIER,
               DB_CLUSTER_PARAMETER_GROUP_NAME,
               DB_ENGINE,
               self.engine_version.as_deref().expect("engine version"),
               self.username.as_deref().expect("username"),
               self.password
                   .replace(SecretString::new("".to_string()))
                   .expect("password"),
           )
           .await;
       if let Err(err) = create_db_cluster {
           return Err(ScenarioError::new(
               "Failed to create DB Cluster with cluster group",
               &err,
           ));
       }

       self.db_cluster_identifier = create_db_cluster
           .unwrap()
```

```
            .db_cluster
            .and_then(|c| c.db_cluster_identifier);

        if self.db_cluster_identifier.is_none() {
            return Err(ScenarioError::with("Created DB Cluster missing
Identifier"));
        }

        info!(
            "Started a db cluster: {}",
            self.db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing ARN")
        );

        let create_db_instance = self
            .rds
            .create_db_instance(
                self.db_cluster_identifier.as_deref().expect("cluster name"),
                DB_INSTANCE_IDENTIFIER,
                self.instance_class.as_deref().expect("instance class"),
                DB_ENGINE,
            )
            .await;
        if let Err(err) = create_db_instance {
            return Err(ScenarioError::new(
                "Failed to create Instance in DB Cluster",
                &err,
            ));
        }

        self.db_instance_identifier = create_db_instance
            .unwrap()
            .db_instance
            .and_then(|i| i.db_instance_identifier);

        // Cluster creation can take up to 20 minutes to become available
        let cluster_max_wait = Duration::from_secs(20 * 60);
        let waiter = Waiter::builder().max(cluster_max_wait).build();
        while waiter.sleep().await.is_ok() {
            let cluster = self
                .rds
                .describe_db_clusters(
                    self.db_cluster_identifier
```

```
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = cluster {
                warn!(?err, "Failed to describe cluster while waiting for
ready");
                continue;
            }

            let instance = self
                .rds
                .describe_db_instance(
                    self.db_instance_identifier
                        .as_deref()
                        .expect("instance identifier"),
                )
                .await;
            if let Err(err) = instance {
                return Err(ScenarioError::new(
                    "Failed to find instance for cluster",
                    &err,
                ));
            }

            let instances_available = instance
                .unwrap()
                .db_instances()
                .iter()
                .all(|instance| instance.db_instance_status() ==
Some("Available"));

            let endpoints = self
                .rds
                .describe_db_cluster_endpoints(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = endpoints {
                return Err(ScenarioError::new(
```

```
                    "Failed to find endpoint for cluster",
                    &err,
                ));
            }

            let endpoints_available = endpoints
                .unwrap()
                .db_cluster_endpoints()
                .iter()
                .all(|endpoint| endpoint.status() == Some("available"));

            if instances_available && endpoints_available {
                return Ok(());
            }
        }

        Err(ScenarioError::with("timed out waiting for cluster"))
    }

    pub async fn describe_db_clusters(
        &self,
        id: &str,
    ) -> Result<DescribeDbClustersOutput, SdkError<DescribeDBClustersError>> {
        self.inner
            .describe_db_clusters()
            .db_cluster_identifier(id)
            .send()
            .await
    }

#[tokio::test]
async fn test_start_cluster_and_instance() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
```

```
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

.db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|id| {
            Ok(DescribeDbClustersOutput::builder()

.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_describe_db_instance()
        .with(eq("RustSDKCodeExamplesDBInstance"))
        .return_once(|name| {
            Ok(DescribeDbInstancesOutput::builder()
```

```
                .db_instances(
                    DbInstance::builder()
                        .db_instance_identifier(name)
                        .db_instance_status("Available")
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

 .db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
        assert!(scenario
            .password
            .replace(SecretString::new("BAD SECRET".into()))
            .unwrap()
            .expose_secret()
            .is_empty());
        assert_eq!(
            scenario.db_cluster_identifier,
            Some("RustSDKCodeExamplesDBCluster".into())
        );
    });
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}
```

```
#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Err(SdkError::service_error(
                CreateDBClusterError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _}) if message
 == "Failed to create DB Cluster with cluster group")
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_missing_id() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()
                .db_cluster(DbCluster::builder().build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
```

```
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context:_ }) if message
 == "Created DB Cluster missing Identifier");
}

#[tokio::test]
async fn test_start_cluster_and_instance_instance_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .return_once(|_, _, _, _| {
            Err(SdkError::service_error(
                CreateDBInstanceError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db instance error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
```

```
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _ }) if message
 == "Failed to create Instance in DB Cluster")
}

#[tokio::test]
async fn test_start_cluster_and_instance_wait_hiccup() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
```

```
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        })
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()

.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds.expect_describe_db_instance().return_once(|name| {
        Ok(DescribeDbInstancesOutput::builder()
            .db_instances(
                DbInstance::builder()
                    .db_instance_identifier(name)
                    .db_instance_status("Available")
                    .build(),
            )
            .build())
    });

    mock_rds
```

```
        .expect_describe_db_cluster_endpoints()
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

 .db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Rust API 参考》中的 DescribeDBClusters。

有关 AWS SDK 开发人员指南和代码示例的完整列表，请参阅 将此服务与 AWS SDK 结合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

将 **DescribeDBEngineVersions** 与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 DescribeDBEngineVersions。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 开始使用数据库集群

.NET

适用于 .NET 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
    /// <summary>
    /// Get a list of DB engine versions for a particular DB engine.
    /// </summary>
    /// <param name="engine">The name of the engine.</param>
    /// <param name="parameterGroupFamily">Optional parameter group family
name.</param>
    /// <returns>A list of DBEngineVersions.</returns>
    public async Task<List<DBEngineVersion>>
DescribeDBEngineVersionsForEngineAsync(string engine,
        string? parameterGroupFamily = null)
    {
        var response = await _amazonRDS.DescribeDBEngineVersionsAsync(
            new DescribeDBEngineVersionsRequest()
            {
                Engine = engine,
                DBParameterGroupFamily = parameterGroupFamily
            });
        return response.DBEngineVersions;
    }
```

- 有关 API 详细信息，请参阅《适用于 .NET 的 AWS SDK API 参考》中的 DescribeDBEngineVersions。

C++

SDK for C++

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);


//! Routine which gets available DB engine versions for an engine name and
//! an optional parameter group family.
/*!
 \sa getDBEngineVersions()
 \param engineName: A DB engine name.
 \param parameterGroupFamily: A parameter group family name, ignored if empty.
 \param engineVersionsResult: Vector of 'DBEngineVersion' objects returned by the
 routine.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::getDBEngineVersions(const Aws::String &engineName,
                                         const Aws::String &parameterGroupFamily,

 Aws::Vector<Aws::RDS::Model::DBEngineVersion> &engineVersionsResult,
                                         const Aws::RDS::RDSClient &client) {
    Aws::RDS::Model::DescribeDBEngineVersionsRequest request;
    request.SetEngine(engineName);
    if (!parameterGroupFamily.empty()) {
        request.SetDBParameterGroupFamily(parameterGroupFamily);
    }

    engineVersionsResult.clear();
    Aws::String marker; // The marker is used for pagination.
    do {
```

```
        if (!marker.empty()) {
            request.SetMarker(marker);
        }

        Aws::RDS::Model::DescribeDBEngineVersionsOutcome outcome =
                client.DescribeDBEngineVersions(request);

        if (outcome.IsSuccess()) {
            const Aws::Vector<Aws::RDS::Model::DBEngineVersion> &engineVersions =
                    outcome.GetResult().GetDBEngineVersions();

            engineVersionsResult.insert(engineVersionsResult.end(),
                                        engineVersions.begin(),
 engineVersions.end());
            marker  = outcome.GetResult().GetMarker();
        }
        else {
            std::cerr << "Error with Aurora::DescribeDBEngineVersionsRequest. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
        }
    } while (!marker.empty());

    return true;
}
```

- 有关 API 详细信息，请参阅《适用于 C++ 的 AWS SDK API 参考》中的 DescribeDBEngineVersions。

Go

适用于 Go V2 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
type DbClusters struct {
 AuroraClient *rds.Client
}



// GetEngineVersions gets database engine versions that are available for the
 specified engine
// and parameter group family.
func (clusters *DbClusters) GetEngineVersions(engine string, parameterGroupFamily
 string) (
 []types.DBEngineVersion, error) {
 output, err := clusters.AuroraClient.DescribeDBEngineVersions(context.TODO(),
  &rds.DescribeDBEngineVersionsInput{
   Engine:                 aws.String(engine),
   DBParameterGroupFamily: aws.String(parameterGroupFamily),
  })
 if err != nil {
  log.Printf("Couldn't get engine versions for %v: %v\n", engine, err)
  return nil, err
 } else {
  return output.DBEngineVersions, nil
 }
}
```

- 有关 API 详细信息，请参阅《适用于 Go 的 AWS SDK API 参考》中的 DescribeDBEngineVersions。

Java

SDK for Java 2.x

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
    public static void describeDBEngines(RdsClient rdsClient) {
```

```
        try {
            DescribeDbEngineVersionsRequest engineVersionsRequest =
DescribeDbEngineVersionsRequest.builder()
                    .engine("aurora-mysql")
                    .defaultOnly(true)
                    .maxRecords(20)
                    .build();

            DescribeDbEngineVersionsResponse response =
rdsClient.describeDBEngineVersions(engineVersionsRequest);
            List<DBEngineVersion> engines = response.dbEngineVersions();

            // Get all DBEngineVersion objects.
            for (DBEngineVersion engineOb : engines) {
                System.out.println("The name of the DB parameter group family for
the database engine is "
                        + engineOb.dbParameterGroupFamily());
                System.out.println("The name of the database engine " +
engineOb.engine());
                System.out.println("The version number of the database engine " +
engineOb.engineVersion());
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- 有关 API 详细信息，请参阅《AWS SDK for Java 2.x API 参考》中的 DescribeDBEngineVersions。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
// Get a list of allowed engine versions.
suspend fun getAllowedClusterEngines(dbParameterGroupFamilyVal: String?) {
    val versionsRequest =
        DescribeDbEngineVersionsRequest {
            dbParameterGroupFamily = dbParameterGroupFamilyVal
            engine = "aurora-mysql"
        }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.describeDbEngineVersions(versionsRequest)
        response.dbEngineVersions?.forEach { dbEngine ->
            println("The engine version is ${dbEngine.engineVersion}")
            println("The engine description is ${dbEngine.dbEngineDescription}")
        }
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Kotlin API 参考》中的 DescribeDBEngineVersions。

Python

SDK for Python (Boto3)

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client
```

```
    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def get_engine_versions(self, engine, parameter_group_family=None):
        """
        Gets database engine versions that are available for the specified engine
        and parameter group family.

        :param engine: The database engine to look up.
        :param parameter_group_family: When specified, restricts the returned
list of
                                       engine versions to those that are
compatible with
                                       this parameter group family.
        :return: The list of database engine versions.
        """
        try:
            kwargs = {"Engine": engine}
            if parameter_group_family is not None:
                kwargs["DBParameterGroupFamily"] = parameter_group_family
            response = self.rds_client.describe_db_engine_versions(**kwargs)
            versions = response["DBEngineVersions"]
        except ClientError as err:
            logger.error(
                "Couldn't get engine versions for %s. Here's why: %s: %s",
                engine,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return versions
```

- 有关 API 详细信息，请参阅《AWS SDK for Python (Boto3) API 参考》中的 DescribeDBEngineVersions。

Rust

适用于 Rust 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
   // Get available engine families for Aurora MySql.
rds.DescribeDbEngineVersions(Engine='aurora-mysql') and build a set of the
'DBParameterGroupFamily' field values. I get {aurora-mysql8.0, aurora-mysql5.7}.
   pub async fn get_engines(&self) -> Result<HashMap<String, Vec<String>>,
ScenarioError> {
       let describe_db_engine_versions =
self.rds.describe_db_engine_versions(DB_ENGINE).await;
       trace!(versions=?describe_db_engine_versions, "full list of versions");

       if let Err(err) = describe_db_engine_versions {
           return Err(ScenarioError::new(
               "Failed to retrieve DB Engine Versions",
               &err,
           ));
       };

       let version_count = describe_db_engine_versions
           .as_ref()
           .map(|o| o.db_engine_versions().len())
           .unwrap_or_default();
       info!(version_count, "got list of versions");

       // Create a map of engine families to their available versions.
       let mut versions = HashMap::<String, Vec<String>>::new();
       describe_db_engine_versions
           .unwrap()
           .db_engine_versions()
           .iter()
           .filter_map(
               |v| match (&v.db_parameter_group_family, &v.engine_version) {
                   (Some(family), Some(version)) => Some((family.clone(),
version.clone())),
```

```
                    _ => None,
                },
            )
            .for_each(|(family, version)|
 versions.entry(family).or_default().push(version));

        Ok(versions)
    }

    pub async fn describe_db_engine_versions(
        &self,
        engine: &str,
    ) -> Result<DescribeDbEngineVersionsOutput,
 SdkError<DescribeDBEngineVersionsError>> {
        self.inner
            .describe_db_engine_versions()
            .engine(engine)
            .send()
            .await
    }

#[tokio::test]
async fn test_scenario_get_engines() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_engine_versions()
        .with(eq("aurora-mysql"))
        .return_once(|_| {
            Ok(DescribeDbEngineVersionsOutput::builder()
                .db_engine_versions(
                    DbEngineVersion::builder()
                        .db_parameter_group_family("f1")
                        .engine_version("f1a")
                        .build(),
                )
                .db_engine_versions(
                    DbEngineVersion::builder()
                        .db_parameter_group_family("f1")
                        .engine_version("f1b")
                        .build(),
                )
                .db_engine_versions(
                    DbEngineVersion::builder()
```

```
                        .db_parameter_group_family("f2")
                        .engine_version("f2a")
                        .build(),
                )
                .db_engine_versions(DbEngineVersion::builder().build())
                .build())
        });

    let scenario = AuroraScenario::new(mock_rds);

    let versions_map = scenario.get_engines().await;

    assert_eq!(
        versions_map,
        Ok(HashMap::from([
            ("f1".into(), vec!["f1a".into(), "f1b".into()]),
            ("f2".into(), vec!["f2a".into()])
        ]))
    );
}

#[tokio::test]
async fn test_scenario_get_engines_failed() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_engine_versions()
        .with(eq("aurora-mysql"))
        .return_once(|_| {
            Err(SdkError::service_error(
                DescribeDBEngineVersionsError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe_db_engine_versions error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let scenario = AuroraScenario::new(mock_rds);

    let versions_map = scenario.get_engines().await;
    assert_matches!(
        versions_map,
```

```
        Err(ScenarioError { message, context: _ }) if message == "Failed to
 retrieve DB Engine Versions"
    );
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Rust API 参考》中的 DescribeDBEngineVersions。

有关 AWS SDK 开发人员指南和代码示例的完整列表，请参阅 将此服务与 AWS SDK 结合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

将 `DescribeDBInstances` 与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 DescribeDBInstances。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 开始使用数据库集群

.NET

适用于 .NET 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
    /// <summary>
    /// Returns a list of DB instances.
    /// </summary>
    /// <param name="dbInstanceIdentifier">Optional name of a specific DB
instance.</param>
    /// <returns>List of DB instances.</returns>
    public async Task<List<DBInstance>> DescribeDBInstancesPagedAsync(string?
dbInstanceIdentifier = null)
    {
```

```
        var results = new List<DBInstance>();
        var instancesPaginator = _amazonRDS.Paginators.DescribeDBInstances(
            new DescribeDBInstancesRequest
            {
                DBInstanceIdentifier = dbInstanceIdentifier
            });
        // Get the entire list using the paginator.
        await foreach (var instances in instancesPaginator.DBInstances)
        {
            results.Add(instances);
        }
        return results;
    }
```

- 有关 API 详细信息，请参阅《适用于 .NET 的 AWS SDK API 参考》中的 DescribeDBInstances。

C++

SDK for C++

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);


//! Routine which gets a DB instance description.
/*!
 \sa describeDBCluster()
 \param dbInstanceIdentifier: A DB instance identifier.
 \param instanceResult: The 'DBInstance' object containing the description.
 \param client: 'RDSClient' instance.
```

```
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::describeDBInstance(const Aws::String &dbInstanceIdentifier,
                                        Aws::RDS::Model::DBInstance
 &instanceResult,
                                        const Aws::RDS::RDSClient &client) {
    Aws::RDS::Model::DescribeDBInstancesRequest request;
    request.SetDBInstanceIdentifier(dbInstanceIdentifier);

    Aws::RDS::Model::DescribeDBInstancesOutcome outcome =
            client.DescribeDBInstances(request);

    bool result = true;
    if (outcome.IsSuccess()) {
        instanceResult = outcome.GetResult().GetDBInstances()[0];
    }
    else if (outcome.GetError().GetErrorType() !=
             Aws::RDS::RDSErrors::D_B_INSTANCE_NOT_FOUND_FAULT) {
        result = false;
        std::cerr << "Error with Aurora::DescribeDBInstances. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
    }
        // This example does not log an error if the DB instance does not exist.
        // Instead, instanceResult is set to empty.
    else {
        instanceResult = Aws::RDS::Model::DBInstance();
    }

    return result;
}
```

- 有关 API 详细信息，请参阅《适用于 C++ 的 AWS SDK API 参考》中的 DescribeDBInstances。

Go

适用于 Go V2 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
type DbClusters struct {
 AuroraClient *rds.Client
}



// GetInstance gets data about a DB instance.
func (clusters *DbClusters) GetInstance(instanceName string) (
 *types.DBInstance, error) {
 output, err := clusters.AuroraClient.DescribeDBInstances(context.TODO(),
  &rds.DescribeDBInstancesInput{
   DBInstanceIdentifier: aws.String(instanceName),
  })
 if err != nil {
  var notFoundError *types.DBInstanceNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("DB instance %v does not exist.\n", instanceName)
   err = nil
  } else {
   log.Printf("Couldn't get instance %v: %v\n", instanceName, err)
  }
  return nil, err
 } else {
  return &output.DBInstances[0], nil
 }
}
```

- 有关 API 详细信息，请参阅《适用于 Go 的 AWS SDK API 参考》中的 DescribeDBInstances。

Java

SDK for Java 2.x

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
   // Waits until the database instance is available.
   public static void waitForInstanceReady(RdsClient rdsClient, String
dbClusterIdentifier) {
       boolean instanceReady = false;
       String instanceReadyStr;
       System.out.println("Waiting for instance to become available.");
       try {
           DescribeDbClustersRequest instanceRequest =
DescribeDbClustersRequest.builder()
                   .dbClusterIdentifier(dbClusterIdentifier)
                   .build();

           while (!instanceReady) {
               DescribeDbClustersResponse response =
rdsClient.describeDBClusters(instanceRequest);
               List<DBCluster> clusterList = response.dbClusters();
               for (DBCluster cluster : clusterList) {
                   instanceReadyStr = cluster.status();
                   if (instanceReadyStr.contains("available")) {
                       instanceReady = true;
                   } else {
                       System.out.print(".");
                       Thread.sleep(sleepTime * 1000);
                   }
               }
           }
           System.out.println("Database cluster is available!");

       } catch (RdsException | InterruptedException e) {
           System.err.println(e.getMessage());
           System.exit(1);
       }
```

```
    }
```

- 有关 API 详细信息，请参阅《AWS SDK for Java 2.x API 参考》中的 DescribeDBInstances。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
suspend fun waitDBAuroraInstanceReady(dbInstanceIdentifierVal: String?) {
    var instanceReady = false
    var instanceReadyStr: String
    println("Waiting for instance to become available.")
    val instanceRequest =
        DescribeDbInstancesRequest {
            dbInstanceIdentifier = dbInstanceIdentifierVal
        }

    var endpoint = ""
    RdsClient { region = "us-west-2" }.use { rdsClient ->
        while (!instanceReady) {
            val response = rdsClient.describeDbInstances(instanceRequest)
            response.dbInstances?.forEach { instance ->
                instanceReadyStr = instance.dbInstanceStatus.toString()
                if (instanceReadyStr.contains("available")) {
                    endpoint = instance.endpoint?.address.toString()
                    instanceReady = true
                } else {
                    print(".")
                    delay(sleepTime * 1000)
                }
            }
        }
    }
```

```
    println("Database instance is available! The connection endpoint is
 $endpoint")
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Kotlin API 参考》中的 DescribeDBInstances。

Python

SDK for Python (Boto3)

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def get_db_instance(self, instance_id):
        """
        Gets data about a DB instance.

        :param instance_id: The ID of the DB instance to retrieve.
        :return: The retrieved DB instance.
```

```
        """
        try:
            response = self.rds_client.describe_db_instances(
                DBInstanceIdentifier=instance_id
            )
            db_inst = response["DBInstances"][0]
        except ClientError as err:
            if err.response["Error"]["Code"] == "DBInstanceNotFound":
                logger.info("Instance %s does not exist.", instance_id)
            else:
                logger.error(
                    "Couldn't get DB instance %s. Here's why: %s: %s",
                    instance_id,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            return db_inst
```

- 有关 API 详细信息，请参阅《AWS SDK for Python (Boto3) API 参考》中的 DescribeDBInstances。

Rust

适用于 Rust 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
    pub async fn clean_up(self) -> Result<(), Vec<ScenarioError>> {
        let mut clean_up_errors: Vec<ScenarioError> = vec![];

        // Delete the instance. rds.DeleteDbInstance.
        let delete_db_instance = self
            .rds
```

```
            .delete_db_instance(
                self.db_instance_identifier
                    .as_deref()
                    .expect("instance identifier"),
            )
            .await;
        if let Err(err) = delete_db_instance {
            let identifier = self
                .db_instance_identifier
                .as_deref()
                .unwrap_or("Missing Instance Identifier");
            let message = format!("failed to delete db instance {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance to delete
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_instances =
self.rds.describe_db_instances().await;
                if let Err(err) = describe_db_instances {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check instance state during deletion",
                        &err,
                    ));
                    break;
                }
                let db_instances = describe_db_instances
                    .unwrap()
                    .db_instances()
                    .iter()
                    .filter(|instance| instance.db_cluster_identifier ==
self.db_cluster_identifier)
                    .cloned()
                    .collect::<Vec<DbInstance>>();

                if db_instances.is_empty() {
                    trace!("Delete Instance waited and no instances were found");
                    break;
                }
                match db_instances.first().unwrap().db_instance_status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but instances is in
{status}");
```

```
                        continue;
                    }
                    None => {
                        warn!("No status for DB instance");
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster. rds.DeleteDbCluster.
        let delete_db_cluster = self
            .rds
            .delete_db_cluster(
                self.db_cluster_identifier
                    .as_deref()
                    .expect("cluster identifier"),
            )
            .await;

        if let Err(err) = delete_db_cluster {
            let identifier = self
                .db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing DB Cluster Identifier");
            let message = format!("failed to delete db cluster {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance and cluster to fully delete.
rds.DescribeDbInstances and rds.DescribeDbClusters until both are not found.
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_clusters = self
                    .rds
                    .describe_db_clusters(
                        self.db_cluster_identifier
                            .as_deref()
                            .expect("cluster identifier"),
                    )
                    .await;
                if let Err(err) = describe_db_clusters {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check cluster state during deletion",
                        &err,
```

```
                    ));
                    break;
                }
                let describe_db_clusters = describe_db_clusters.unwrap();
                let db_clusters = describe_db_clusters.db_clusters();
                if db_clusters.is_empty() {
                    trace!("Delete cluster waited and no clusters were found");
                    break;
                }
                match db_clusters.first().unwrap().status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but clusters is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB cluster");
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster parameter group.
rds.DeleteDbClusterParameterGroup.
        let delete_db_cluster_parameter_group = self
            .rds
            .delete_db_cluster_parameter_group(
                self.db_cluster_parameter_group
                    .map(|g| {
                        g.db_cluster_parameter_group_name
                            .unwrap_or_else(||
DB_CLUSTER_PARAMETER_GROUP_NAME.to_string())
                    })
                    .as_deref()
                    .expect("cluster parameter group name"),
            )
            .await;
        if let Err(error) = delete_db_cluster_parameter_group {
            clean_up_errors.push(ScenarioError::new(
                "Failed to delete the db cluster parameter group",
                &error,
            ))
```

```
        }

        if clean_up_errors.is_empty() {
            Ok(())
        } else {
            Err(clean_up_errors)
        }
    }

    pub async fn describe_db_instances(
        &self,
    ) -> Result<DescribeDbInstancesOutput, SdkError<DescribeDBInstancesError>> {
        self.inner.describe_db_instances().send().await
    }

#[tokio::test]
async fn test_scenario_clean_up() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| Ok(DescribeDbInstancesOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster()
```

```
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| Ok(DescribeDbClustersOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
Describe Instances
```

```
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_scenario_clean_up_errors() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| {
            Err(SdkError::service_error(
                DescribeDBInstancesError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db instances error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
```

```
        });

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db clusters error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
```

```
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_err());
        let errs = clean_up.unwrap_err();
        assert_eq!(errs.len(), 2);
        assert_matches!(errs.get(0), Some(ScenarioError {message, context: _}) if
 message == "Failed to check instance state during deletion");
        assert_matches!(errs.get(1), Some(ScenarioError {message, context: _}) if
 message == "Failed to check cluster state during deletion");
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Rust API 参考》中的 DescribeDBInstances。

有关 AWS SDK 开发人员指南和代码示例的完整列表，请参阅 将此服务与 AWS SDK 结合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

将 **DescribeOrderableDBInstanceOptions** 与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 DescribeOrderableDBInstanceOptions。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 开始使用数据库集群

.NET

适用于 .NET 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
    /// <summary>
    /// Get a list of orderable DB instance options for a specific
    /// engine and engine version.
    /// </summary>
    /// <param name="engine">Name of the engine.</param>
    /// <param name="engineVersion">Version of the engine.</param>
    /// <returns>List of OrderableDBInstanceOptions.</returns>
    public async Task<List<OrderableDBInstanceOption>>
DescribeOrderableDBInstanceOptionsPagedAsync(string engine, string
engineVersion)
    {
        // Use a paginator to get a list of DB instance options.
        var results = new List<OrderableDBInstanceOption>();
        var paginateInstanceOptions =
_amazonRDS.Paginators.DescribeOrderableDBInstanceOptions(
            new DescribeOrderableDBInstanceOptionsRequest()
            {
                Engine = engine,
                EngineVersion = engineVersion,
            });
        // Get the entire list using the paginator.
        await foreach (var instanceOptions in
paginateInstanceOptions.OrderableDBInstanceOptions)
        {
            results.Add(instanceOptions);
        }
        return results;
    }
```

- 有关 API 详细信息，请参阅《适用于 .NET 的 AWS SDK API 参考》中的 DescribeOrderableDBInstanceOptions。

C++

SDK for C++

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);


//! Routine which gets available DB instance classes, displays the list
//! to the user, and returns the user selection.
/*!
 \sa chooseDBInstanceClass()
 \param engineName: The DB engine name.
 \param engineVersion: The DB engine version.
 \param dbInstanceClass: String for DB instance class chosen by the user.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::chooseDBInstanceClass(const Aws::String &engine,
                                           const Aws::String &engineVersion,
                                           Aws::String &dbInstanceClass,
                                           const Aws::RDS::RDSClient &client) {
    std::vector<Aws::String> instanceClasses;
    Aws::String marker; // The marker is used for pagination.
    do {
        Aws::RDS::Model::DescribeOrderableDBInstanceOptionsRequest request;
        request.SetEngine(engine);
        request.SetEngineVersion(engineVersion);
        if (!marker.empty()) {
```

```
            request.SetMarker(marker);
        }

        Aws::RDS::Model::DescribeOrderableDBInstanceOptionsOutcome outcome =
                client.DescribeOrderableDBInstanceOptions(request);

        if (outcome.IsSuccess()) {
            const Aws::Vector<Aws::RDS::Model::OrderableDBInstanceOption>
 &options =
                    outcome.GetResult().GetOrderableDBInstanceOptions();
            for (const Aws::RDS::Model::OrderableDBInstanceOption &option:
 options) {
                const Aws::String &instanceClass = option.GetDBInstanceClass();
                if (std::find(instanceClasses.begin(), instanceClasses.end(),
                              instanceClass) == instanceClasses.end()) {
                    instanceClasses.push_back(instanceClass);
                }
            }
            marker = outcome.GetResult().GetMarker();
        }
        else {
            std::cerr << "Error with Aurora::DescribeOrderableDBInstanceOptions.
 "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    } while (!marker.empty());

    std::cout << "The available DB instance classes for your database engine
 are:"
              << std::endl;
    for (int i = 0; i < instanceClasses.size(); ++i) {
        std::cout << "   " << i + 1 << ": " << instanceClasses[i] << std::endl;
    }

    int choice = askQuestionForIntRange(
            "Which DB instance class do you want to use? ",
            1, static_cast<int>(instanceClasses.size()));
    dbInstanceClass = instanceClasses[choice - 1];
    return true;
}
```

- 有关 API 详细信息，请参阅《适用于 C++ 的 AWS SDK API 参考》中的 DescribeOrderableDBInstanceOptions。

Go

适用于 Go V2 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
type DbClusters struct {
 AuroraClient *rds.Client
}



// GetOrderableInstances uses a paginator to get DB instance options that can be
 used to create DB instances that are
// compatible with a set of specifications.
func (clusters *DbClusters) GetOrderableInstances(engine string, engineVersion
 string) (
 []types.OrderableDBInstanceOption, error) {

 var output *rds.DescribeOrderableDBInstanceOptionsOutput
 var instances []types.OrderableDBInstanceOption
 var err error
 orderablePaginator :=
 rds.NewDescribeOrderableDBInstanceOptionsPaginator(clusters.AuroraClient,
  &rds.DescribeOrderableDBInstanceOptionsInput{
   Engine:        aws.String(engine),
   EngineVersion: aws.String(engineVersion),
  })
 for orderablePaginator.HasMorePages() {
  output, err = orderablePaginator.NextPage(context.TODO())
  if err != nil {
   log.Printf("Couldn't get orderable DB instances: %v\n", err)
   break
```

```
  } else {
   instances = append(instances, output.OrderableDBInstanceOptions...)
  }
 }
 return instances, err
}
```

- 有关 API 详细信息，请参阅《适用于 Go 的 AWS SDK API 参考》中的 DescribeOrderableDBInstanceOptions。

Java

SDK for Java 2.x

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
    public static void describeDBEngines(RdsClient rdsClient) {
        try {
            DescribeDbEngineVersionsRequest engineVersionsRequest =
DescribeDbEngineVersionsRequest.builder()
                    .engine("aurora-mysql")
                    .defaultOnly(true)
                    .maxRecords(20)
                    .build();

            DescribeDbEngineVersionsResponse response =
rdsClient.describeDBEngineVersions(engineVersionsRequest);
            List<DBEngineVersion> engines = response.dbEngineVersions();

            // Get all DBEngineVersion objects.
            for (DBEngineVersion engineOb : engines) {
                System.out.println("The name of the DB parameter group family for
the database engine is "
                        + engineOb.dbParameterGroupFamily());
```

```
                System.out.println("The name of the database engine " +
 engineOb.engine());
                System.out.println("The version number of the database engine " +
 engineOb.engineVersion());
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- 有关 API 详细信息，请参阅《AWS SDK for Java 2.x API 参考》中的 DescribeOrderableDBInstanceOptions。

PowerShell

适用于 PowerShell 的工具

示例 1：此示例列出了支持AWS 区域特定数据库实例类的数据库引擎版本。

```
$params = @{
  Engine = 'aurora-postgresql'
  DBInstanceClass = 'db.r5.large'
  Region = 'us-east-1'
}
Get-RDSOrderableDBInstanceOption @params
```

示例 2：此示例列出了AWS 区域特定数据库引擎版本支持的数据库实例类。

```
$params = @{
  Engine = 'aurora-postgresql'
  EngineVersion = '13.6'
  Region = 'us-east-1'
}
Get-RDSOrderableDBInstanceOption @params
```

- 有关 API 详细信息，请参阅《AWS Tools for PowerShell Cmdlet 参考》中的 DescribeOrderableDBInstanceOptions。

Python

SDK for Python (Boto3)

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def get_orderable_instances(self, db_engine, db_engine_version):
        """
        Gets DB instance options that can be used to create DB instances that are
        compatible with a set of specifications.

        :param db_engine: The database engine that must be supported by the DB
 instance.
        :param db_engine_version: The engine version that must be supported by
 the DB instance.
        :return: The list of DB instance options that can be used to create a
 compatible DB instance.
        """
        try:
```

```
            inst_opts = []
            paginator = self.rds_client.get_paginator(
                "describe_orderable_db_instance_options"
            )
            for page in paginator.paginate(
                Engine=db_engine, EngineVersion=db_engine_version
            ):
                inst_opts += page["OrderableDBInstanceOptions"]
        except ClientError as err:
            logger.error(
                "Couldn't get orderable DB instances. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return inst_opts
```

- 有关 API 详细信息，请参阅《AWS SDK for Python (Boto3) API 参考》中的 DescribeOrderableDBInstanceOptions。

Rust

适用于 Rust 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
   pub async fn get_instance_classes(&self) -> Result<Vec<String>,
ScenarioError> {
        let describe_orderable_db_instance_options_items = self
            .rds
            .describe_orderable_db_instance_options(
                DB_ENGINE,
                self.engine_version
                    .as_ref()
```

```
                    .expect("engine version for db instance options")
                    .as_str(),
            )
            .await;

        describe_orderable_db_instance_options_items
            .map(|options| {
                options
                    .iter()
                    .filter(|o| o.storage_type() == Some("aurora"))
                    .map(|o|
 o.db_instance_class().unwrap_or_default().to_string())
                    .collect::<Vec<String>>()
            })
            .map_err(|err| ScenarioError::new("Could not get available instance
 classes", &err))
    }

    pub async fn describe_orderable_db_instance_options(
        &self,
        engine: &str,
        engine_version: &str,
    ) -> Result<Vec<OrderableDbInstanceOption>,
 SdkError<DescribeOrderableDBInstanceOptionsError>>
    {
        self.inner
            .describe_orderable_db_instance_options()
            .engine(engine)
            .engine_version(engine_version)
            .into_paginator()
            .items()
            .send()
            .try_collect()
            .await
    }

#[tokio::test]
async fn test_scenario_get_instance_classes() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .return_once(|_, _, _| {
            Ok(CreateDbClusterParameterGroupOutput::builder()
```

```
 .db_cluster_parameter_group(DbClusterParameterGroup::builder().build())
                .build())
        });

    mock_rds
        .expect_describe_orderable_db_instance_options()
        .with(eq("aurora-mysql"), eq("aurora-mysql8.0"))
        .return_once(|_, _| {
            Ok(vec![
                OrderableDbInstanceOption::builder()
                    .db_instance_class("t1")
                    .storage_type("aurora")
                    .build(),
                OrderableDbInstanceOption::builder()
                    .db_instance_class("t1")
                    .storage_type("aurora-iopt1")
                    .build(),
                OrderableDbInstanceOption::builder()
                    .db_instance_class("t2")
                    .storage_type("aurora")
                    .build(),
                OrderableDbInstanceOption::builder()
                    .db_instance_class("t3")
                    .storage_type("aurora")
                    .build(),
            ])
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario
        .set_engine("aurora-mysql", "aurora-mysql8.0")
        .await
        .expect("set engine");

    let instance_classes = scenario.get_instance_classes().await;

    assert_eq!(
        instance_classes,
        Ok(vec!["t1".into(), "t2".into(), "t3".into()])
    );
}

#[tokio::test]
```

```
async fn test_scenario_get_instance_classes_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_orderable_db_instance_options()
        .with(eq("aurora-mysql"), eq("aurora-mysql8.0"))
        .return_once(|_, _| {
            Err(SdkError::service_error(

 DescribeOrderableDBInstanceOptionsError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe_orderable_db_instance_options_error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_family = Some("aurora-mysql".into());
    scenario.engine_version = Some("aurora-mysql8.0".into());

    let instance_classes = scenario.get_instance_classes().await;

    assert_matches!(
        instance_classes,
        Err(ScenarioError {message, context: _}) if message == "Could not get
 available instance classes"
    );
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Rust API 参考》中的 DescribeOrderableDBInstanceOptions。

有关 AWS SDK 开发人员指南和代码示例的完整列表，请参阅 将此服务与 AWS SDK 结合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

将 **ModifyDBClusterParameterGroup** 与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 ModifyDBClusterParameterGroup。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 开始使用数据库集群

.NET

适用于 .NET 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
    /// <summary>
    /// Modify the specified integer parameters with new values from user input.
    /// </summary>
    /// <param name="groupName">The group name for the parameters.</param>
    /// <param name="parameters">The list of integer parameters to modify.</
param>
    /// <param name="newValue">Optional int value to set for parameters.</param>
    /// <returns>The name of the group that was modified.</returns>
    public async Task<string> ModifyIntegerParametersInGroupAsync(string
 groupName, List<Parameter> parameters, int newValue = 0)
    {
        foreach (var p in parameters)
        {
            if (p.IsModifiable && p.DataType == "integer")
            {
                while (newValue == 0)
                {
                    Console.WriteLine(
                        $"Enter a new value for {p.ParameterName} from the
 allowed values {p.AllowedValues} ");

                    var choice = Console.ReadLine();
                    int.TryParse(choice, out newValue);
                }

                p.ParameterValue = newValue.ToString();
```

```
            }
        }

        var request = new ModifyDBClusterParameterGroupRequest
        {
            Parameters = parameters,
            DBClusterParameterGroupName = groupName,
        };

        var result = await
_amazonRDS.ModifyDBClusterParameterGroupAsync(request);
        return result.DBClusterParameterGroupName;
    }
```

- 有关 API 详细信息，请参阅《适用于 .NET 的 AWS SDK API 参考》中的 ModifyDBClusterParameterGroup。

C++

SDK for C++

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

        Aws::RDS::Model::ModifyDBClusterParameterGroupRequest request;
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);
        request.SetParameters(updateParameters);

        Aws::RDS::Model::ModifyDBClusterParameterGroupOutcome outcome =
                client.ModifyDBClusterParameterGroup(request);
```

```
        if (outcome.IsSuccess()) {
            std::cout << "The DB cluster parameter group was successfully
 modified."
                      << std::endl;
        }
        else {
            std::cerr << "Error with Aurora::ModifyDBClusterParameterGroup. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
        }
```

- 有关 API 详细信息，请参阅《适用于 C++ 的 AWS SDK API 参考》中的 ModifyDBClusterParameterGroup。

Go

适用于 Go V2 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
type DbClusters struct {
 AuroraClient *rds.Client
}



// UpdateParameters updates parameters in a named DB cluster parameter group.
func (clusters *DbClusters) UpdateParameters(parameterGroupName string, params
 []types.Parameter) error {
 _, err := clusters.AuroraClient.ModifyDBClusterParameterGroup(context.TODO(),
  &rds.ModifyDBClusterParameterGroupInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
   Parameters:                  params,
  })
 if err != nil {
```

```
  log.Printf("Couldn't update parameters in %v: %v\n", parameterGroupName, err)
  return err
 } else {
  return nil
 }
}
```

- 有关 API 详细信息，请参阅《适用于 Go 的 AWS SDK API 参考》中的 ModifyDBClusterParameterGroup。

Java

SDK for Java 2.x

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
   public static void describeDbClusterParameterGroups(RdsClient rdsClient,
String dbClusterGroupName) {
       try {
           DescribeDbClusterParameterGroupsRequest groupsRequest =
DescribeDbClusterParameterGroupsRequest.builder()
                   .dbClusterParameterGroupName(dbClusterGroupName)
                   .maxRecords(20)
                   .build();

           List<DBClusterParameterGroup> groups =
rdsClient.describeDBClusterParameterGroups(groupsRequest)
                   .dbClusterParameterGroups();
           for (DBClusterParameterGroup group : groups) {
               System.out.println("The group name is " +
group.dbClusterParameterGroupName());
               System.out.println("The group ARN is " +
group.dbClusterParameterGroupArn());
           }
```

```
        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- 有关 API 详细信息，请参阅《AWS SDK for Java 2.x API 参考》中的 ModifyDBClusterParameterGroup。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
// Modify the auto_increment_offset parameter.
suspend fun modifyDBClusterParas(dClusterGroupName: String?) {
    val parameter1 =
        Parameter {
            parameterName = "auto_increment_offset"
            applyMethod = ApplyMethod.fromValue("immediate")
            parameterValue = "5"
        }

    val paraList = ArrayList<Parameter>()
    paraList.add(parameter1)
    val groupRequest =
        ModifyDbClusterParameterGroupRequest {
            dbClusterParameterGroupName = dClusterGroupName
            parameters = paraList
        }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.modifyDbClusterParameterGroup(groupRequest)
        println("The parameter group ${response.dbClusterParameterGroupName} was
 successfully modified")
```

```
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Kotlin API 参考》中的 ModifyDBClusterParameterGroup。

Python

SDK for Python (Boto3)

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def update_parameters(self, parameter_group_name, update_parameters):
        """
        Updates parameters in a custom DB cluster parameter group.

        :param parameter_group_name: The name of the parameter group to update.
```

```
        :param update_parameters: The parameters to update in the group.
        :return: Data about the modified parameter group.
        """
        try:
            response = self.rds_client.modify_db_cluster_parameter_group(
                DBClusterParameterGroupName=parameter_group_name,
                Parameters=update_parameters,
            )
        except ClientError as err:
            logger.error(
                "Couldn't update parameters in %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response
```

- 有关 API 详细信息，请参阅《AWS SDK for Python (Boto3) API 参考》中的 ModifyDBClusterParameterGroup。

Rust

适用于 Rust 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
   // Modify both the auto_increment_offset and auto_increment_increment
parameters in one call in the custom parameter group. Set their ParameterValue
fields to a new allowable value. rds.ModifyDbClusterParameterGroup.
   pub async fn update_auto_increment(
       &self,
       offset: u8,
       increment: u8,
```

```
    ) -> Result<(), ScenarioError> {
        let modify_db_cluster_parameter_group = self
            .rds
            .modify_db_cluster_parameter_group(
                DB_CLUSTER_PARAMETER_GROUP_NAME,
                vec![
                    Parameter::builder()
                        .parameter_name("auto_increment_offset")
                        .parameter_value(format!("{offset}"))
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                    Parameter::builder()
                        .parameter_name("auto_increment_increment")
                        .parameter_value(format!("{increment}"))
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                ],
            )
            .await;

        if let Err(error) = modify_db_cluster_parameter_group {
            return Err(ScenarioError::new(
                "Failed to modify cluster parameter group",
                &error,
            ));
        }

        Ok(())
    }

    pub async fn modify_db_cluster_parameter_group(
        &self,
        name: &str,
        parameters: Vec<Parameter>,
    ) -> Result<ModifyDbClusterParameterGroupOutput,
SdkError<ModifyDBClusterParameterGroupError>>
    {
        self.inner
            .modify_db_cluster_parameter_group()
            .db_cluster_parameter_group_name(name)
            .set_parameters(Some(parameters))
            .send()
            .await
    }
```

```
#[tokio::test]
async fn test_scenario_update_auto_increment() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_modify_db_cluster_parameter_group()
        .withf(|name, params| {
            assert_eq!(name, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(
                params,
                &vec![
                    Parameter::builder()
                        .parameter_name("auto_increment_offset")
                        .parameter_value("10")
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                    Parameter::builder()
                        .parameter_name("auto_increment_increment")
                        .parameter_value("20")
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                ]
            );
            true
        })
        .return_once(|_, _|
 Ok(ModifyDbClusterParameterGroupOutput::builder().build()));

    let scenario = AuroraScenario::new(mock_rds);

    scenario
        .update_auto_increment(10, 20)
        .await
        .expect("update auto increment");
}

#[tokio::test]
async fn test_scenario_update_auto_increment_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_modify_db_cluster_parameter_group()
        .return_once(|_, _| {
```

```
            Err(SdkError::service_error(

 ModifyDBClusterParameterGroupError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "modify_db_cluster_parameter_group_error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let scenario = AuroraScenario::new(mock_rds);

    let update = scenario.update_auto_increment(10, 20).await;
    assert_matches!(update, Err(ScenarioError { message, context: _}) if message
 == "Failed to modify cluster parameter group");
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Rust API 参考》中的 ModifyDBClusterParameterGroup。

有关 AWS SDK 开发人员指南和代码示例的完整列表，请参阅 将此服务与 AWS SDK 结合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

使用 AWS SDK 的 Aurora 场景

以下代码示例显示如何通过 AWS SDK 实施 Aurora 中的常见场景。这些场景显示了如何通过在 Aurora 中调用多个函数来完成特定任务。每个场景都包含一个指向 GitHub 的链接，其中包含了有关如何设置和运行代码的说明。

示例

- 通过 AWS SDK 开始使用 Aurora 数据库集群

通过 AWS SDK 开始使用 Aurora 数据库集群

以下代码示例演示了如何：

- 创建自定义 Aurora 数据库集群参数组并设置参数值。

- 创建一个使用参数组的数据库集群。
- 创建包含数据库的数据库实例。
- 拍摄数据库集群的快照，然后清理资源。

.NET

适用于 .NET 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

在命令提示符中运行交互式场景。

```
using Amazon.RDS;
using Amazon.RDS.Model;
using AuroraActions;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Logging.Console;
using Microsoft.Extensions.Logging.Debug;

namespace AuroraScenario;

/// <summary>
/// Scenario for Amazon Aurora examples.
/// </summary>
public class AuroraScenario
{
    /*
    Before running this .NET code example, set up your development environment,
 including your credentials.

    This .NET example performs the following tasks:
    1.  Return a list of the available DB engine families for Aurora MySql using
 the DescribeDBEngineVersionsAsync method.
```

```
   2.  Select an engine family and create a custom DB cluster parameter group
using the CreateDBClusterParameterGroupAsync method.
   3.  Get the parameter group using the DescribeDBClusterParameterGroupsAsync
method.
   4.  Get some parameters in the group using the
DescribeDBClusterParametersAsync method.
   5.  Parse and display some parameters in the group.
   6.  Modify the auto_increment_offset and auto_increment_increment parameters
       using the ModifyDBClusterParameterGroupAsync method.
   7.  Get and display the updated parameters using the
DescribeDBClusterParametersAsync method with a source of "user".
   8.  Get a list of allowed engine versions using the
DescribeDBEngineVersionsAsync method.
   9.  Create an Aurora DB cluster that contains a MySql database and uses the
parameter group.
       using the CreateDBClusterAsync method.
   10. Wait for the DB cluster to be ready using the DescribeDBClustersAsync
method.
   11. Display and select from a list of instance classes available for the
selected engine and version
       using the paginated DescribeOrderableDBInstanceOptions method.
   12. Create a database instance in the cluster using the CreateDBInstanceAsync
method.
   13. Wait for the DB instance to be ready using the DescribeDBInstances
method.
   14. Display the connection endpoint string for the new DB cluster.
   15. Create a snapshot of the DB cluster using the
CreateDBClusterSnapshotAsync method.
   16. Wait for DB snapshot to be ready using the
DescribeDBClusterSnapshotsAsync method.
   17. Delete the DB instance using the DeleteDBInstanceAsync method.
   18. Delete the DB cluster using the DeleteDBClusterAsync method.
   19. Wait for DB cluster to be deleted using the DescribeDBClustersAsync
methods.
   20. Delete the cluster parameter group using the
DeleteDBClusterParameterGroupAsync.
   */

   private static readonly string sepBar = new('-', 80);
   private static AuroraWrapper auroraWrapper = null!;
   private static ILogger logger = null!;
   private static readonly string engine = "aurora-mysql";
   static async Task Main(string[] args)
   {
```

```
        // Set up dependency injection for the Amazon Relational Database Service
(Amazon RDS).
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft",
LogLevel.Trace))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonRDS>()
                    .AddTransient<AuroraWrapper>()
            )
            .Build();

        logger = LoggerFactory.Create(builder =>
        {
            builder.AddConsole();
        }).CreateLogger<AuroraScenario>();

        auroraWrapper = host.Services.GetRequiredService<AuroraWrapper>();

        Console.WriteLine(sepBar);
        Console.WriteLine(
            "Welcome to the Amazon Aurora: get started with DB clusters
example.");
        Console.WriteLine(sepBar);

        DBClusterParameterGroup parameterGroup = null!;
        DBCluster? newCluster = null;
        DBInstance? newInstance = null;

        try
        {
            var parameterGroupFamily = await ChooseParameterGroupFamilyAsync();

            parameterGroup = await
CreateDBParameterGroupAsync(parameterGroupFamily);

            var parameters = await
DescribeParametersInGroupAsync(parameterGroup.DBClusterParameterGroupName,
                new List<string> { "auto_increment_offset",
"auto_increment_increment" });
```

```
            await
ModifyParametersAsync(parameterGroup.DBClusterParameterGroupName, parameters);

            await
DescribeUserSourceParameters(parameterGroup.DBClusterParameterGroupName);

            var engineVersionChoice = await
ChooseDBEngineVersionAsync(parameterGroupFamily);

            var newClusterIdentifier = "Example-Cluster-" + DateTime.Now.Ticks;

            newCluster = await CreateNewCluster
            (
                parameterGroup,
                engine,
                engineVersionChoice.EngineVersion,
                newClusterIdentifier
            );

            var instanceClassChoice = await ChooseDBInstanceClass(engine,
engineVersionChoice.EngineVersion);

            var newInstanceIdentifier = "Example-Instance-" + DateTime.Now.Ticks;

            newInstance = await CreateNewInstance(
                newClusterIdentifier,
                engine,
                engineVersionChoice.EngineVersion,
                instanceClassChoice.DBInstanceClass,
                newInstanceIdentifier
            );

            DisplayConnectionString(newCluster!);
            await CreateSnapshot(newCluster!);
            await CleanupResources(newInstance, newCluster, parameterGroup);

            Console.WriteLine("Scenario complete.");
            Console.WriteLine(sepBar);
        }
        catch (Exception ex)

        {
            await CleanupResources(newInstance, newCluster, parameterGroup);
            logger.LogError(ex, "There was a problem executing the scenario.");
```

```
        }
    }

    /// <summary>
    /// Choose the Aurora DB parameter group family from a list of available
options.
    /// </summary>
    /// <returns>The selected parameter group family.</returns>
    public static async Task<string> ChooseParameterGroupFamilyAsync()
    {
        Console.WriteLine(sepBar);
        // 1. Get a list of available engines.
        var engines = await
auroraWrapper.DescribeDBEngineVersionsForEngineAsync(engine);

        Console.WriteLine($"1. The following is a list of available DB parameter
group families for engine {engine}:");

        var parameterGroupFamilies =
            engines.GroupBy(e => e.DBParameterGroupFamily).ToList();
        for (var i = 1; i <= parameterGroupFamilies.Count; i++)
        {
            var parameterGroupFamily = parameterGroupFamilies[i - 1];
            // List the available parameter group families.
            Console.WriteLine(
                $"\t{i}. Family: {parameterGroupFamily.Key}");
        }

        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > parameterGroupFamilies.Count)
        {
            Console.WriteLine("2. Select an available DB parameter group family
by entering a number from the preceding list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }
        var parameterGroupFamilyChoice = parameterGroupFamilies[choiceNumber -
1];
        Console.WriteLine(sepBar);
        return parameterGroupFamilyChoice.Key;
    }

    /// <summary>
    /// Create and get information on a DB parameter group.
```

```
    /// </summary>
    /// <param name="dbParameterGroupFamily">The DBParameterGroupFamily for the
 new DB parameter group.</param>
    /// <returns>The new DBParameterGroup.</returns>
    public static async Task<DBClusterParameterGroup>
 CreateDBParameterGroupAsync(string dbParameterGroupFamily)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine($"2. Create new DB parameter group with family
 {dbParameterGroupFamily}:");

        var parameterGroup = await
 auroraWrapper.CreateCustomClusterParameterGroupAsync(
            dbParameterGroupFamily,
            "ExampleParameterGroup-" + DateTime.Now.Ticks,
            "New example parameter group");

        var groupInfo =
            await
 auroraWrapper.DescribeCustomDBClusterParameterGroupAsync(parameterGroup.DBClusterParamet

        Console.WriteLine(
            $"3. New DB parameter group created: \n\t{groupInfo?.Description}, \n
\tARN {groupInfo?.DBClusterParameterGroupName}");
        Console.WriteLine(sepBar);
        return parameterGroup;
    }

    /// <summary>
    /// Get and describe parameters from a DBParameterGroup.
    /// </summary>
    /// <param name="parameterGroupName">The name of the DBParameterGroup.</
param>
    /// <param name="parameterNames">Optional specific names of parameters to
 describe.</param>
    /// <returns>The list of requested parameters.</returns>
    public static async Task<List<Parameter>>
 DescribeParametersInGroupAsync(string parameterGroupName, List<string>?
 parameterNames = null)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine("4. Get some parameters from the group.");
        Console.WriteLine(sepBar);
```

```
        var parameters =
            await
auroraWrapper.DescribeDBClusterParametersInGroupAsync(parameterGroupName);

        var matchingParameters =
            parameters.Where(p => parameterNames == null ||
parameterNames.Contains(p.ParameterName)).ToList();

        Console.WriteLine("5. Parameter information:");
        matchingParameters.ForEach(p =>
            Console.WriteLine(
                $"\n\tParameter: {p.ParameterName}." +
                $"\n\tDescription: {p.Description}." +
                $"\n\tAllowed Values: {p.AllowedValues}." +
                $"\n\tValue: {p.ParameterValue}."));

        Console.WriteLine(sepBar);

        return matchingParameters;
    }

    /// <summary>
    /// Modify a parameter from a DBParameterGroup.
    /// </summary>
    /// <param name="parameterGroupName">Name of the DBParameterGroup.</param>
    /// <param name="parameters">The parameters to modify.</param>
    /// <returns>Async task.</returns>
    public static async Task ModifyParametersAsync(string parameterGroupName,
List<Parameter> parameters)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine("6. Modify some parameters in the group.");

        await
auroraWrapper.ModifyIntegerParametersInGroupAsync(parameterGroupName,
parameters);

        Console.WriteLine(sepBar);
    }

    /// <summary>
    /// Describe the user source parameters in the group.
    /// </summary>
```

```
    /// <param name="parameterGroupName">The name of the DBParameterGroup.</
param>
    /// <returns>Async task.</returns>
    public static async Task DescribeUserSourceParameters(string
 parameterGroupName)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine("7. Describe updated user source parameters in the
 group.");

        var parameters =
            await
 auroraWrapper.DescribeDBClusterParametersInGroupAsync(parameterGroupName,
 "user");

        parameters.ForEach(p =>
            Console.WriteLine(
                $"\n\tParameter: {p.ParameterName}." +
                $"\n\tDescription: {p.Description}." +
                $"\n\tAllowed Values: {p.AllowedValues}." +
                $"\n\tValue: {p.ParameterValue}."));

        Console.WriteLine(sepBar);
    }

    /// <summary>
    /// Choose a DB engine version.
    /// </summary>
    /// <param name="dbParameterGroupFamily">DB parameter group family for engine
 choice.</param>
    /// <returns>The selected engine version.</returns>
    public static async Task<DBEngineVersion> ChooseDBEngineVersionAsync(string
 dbParameterGroupFamily)
    {
        Console.WriteLine(sepBar);
        // Get a list of allowed engines.
        var allowedEngines =
            await auroraWrapper.DescribeDBEngineVersionsForEngineAsync(engine,
 dbParameterGroupFamily);

        Console.WriteLine($"Available DB engine versions for parameter group
 family {dbParameterGroupFamily}:");
        int i = 1;
        foreach (var version in allowedEngines)
```

```
        {
            Console.WriteLine(
                $"\t{i}. {version.DBEngineVersionDescription}.");
            i++;
        }

        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > allowedEngines.Count)
        {
            Console.WriteLine("8. Select an available DB engine version by
 entering a number from the list above:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }

        var engineChoice = allowedEngines[choiceNumber - 1];
        Console.WriteLine(sepBar);
        return engineChoice;
    }

    /// <summary>
    /// Create a new RDS DB cluster.
    /// </summary>
    /// <param name="parameterGroup">Parameter group to use for the DB cluster.</
param>
    /// <param name="engineName">Engine to use for the DB cluster.</param>
    /// <param name="engineVersion">Engine version to use for the DB cluster.</
param>
    /// <param name="clusterIdentifier">Cluster identifier to use for the DB
 cluster.</param>
    /// <returns>The new DB cluster.</returns>
    public static async Task<DBCluster?> CreateNewCluster(DBClusterParameterGroup
 parameterGroup,
        string engineName, string engineVersion, string clusterIdentifier)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine($"9. Create a new DB cluster with identifier
 {clusterIdentifier}.");

        DBCluster newCluster;
        var clusters = await auroraWrapper.DescribeDBClustersPagedAsync();
        var isClusterCreated = clusters.Any(i => i.DBClusterIdentifier ==
 clusterIdentifier);
```

```
        if (isClusterCreated)
        {
            Console.WriteLine("Cluster already created.");
            newCluster = clusters.First(i => i.DBClusterIdentifier ==
clusterIdentifier);
        }
        else
        {
            Console.WriteLine("Enter an admin username:");
            var username = Console.ReadLine();

            Console.WriteLine("Enter an admin password:");
            var password = Console.ReadLine();

            newCluster = await auroraWrapper.CreateDBClusterWithAdminAsync(
                "ExampleDatabase",
                clusterIdentifier,
                parameterGroup.DBClusterParameterGroupName,
                engineName,
                engineVersion,
                username!,
                password!
            );

            Console.WriteLine("10. Waiting for DB cluster to be ready...");
            while (newCluster.Status != "available")
            {
                Console.Write(".");
                Thread.Sleep(5000);
                clusters = await
auroraWrapper.DescribeDBClustersPagedAsync(clusterIdentifier);
                newCluster = clusters.First();
            }
        }

        Console.WriteLine(sepBar);
        return newCluster;
    }

    /// <summary>
    /// Choose a DB instance class for a particular engine and engine version.
    /// </summary>
    /// <param name="engine">DB engine for DB instance choice.</param>
```

```
    /// <param name="engineVersion">DB engine version for DB instance choice.</
param>
    /// <returns>The selected orderable DB instance option.</returns>
    public static async Task<OrderableDBInstanceOption>
 ChooseDBInstanceClass(string engine, string engineVersion)
    {
        Console.WriteLine(sepBar);
        // Get a list of allowed DB instance classes.
        var allowedInstances =
            await
 auroraWrapper.DescribeOrderableDBInstanceOptionsPagedAsync(engine,
 engineVersion);

        Console.WriteLine($"Available DB instance classes for engine {engine} and
 version {engineVersion}:");
        int i = 1;


        foreach (var instance in allowedInstances)
        {
            Console.WriteLine(
                $"\t{i}. Instance class: {instance.DBInstanceClass} (storage type
 {instance.StorageType})");
            i++;
        }

        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > allowedInstances.Count)
        {
            Console.WriteLine("11. Select an available DB instance class by
 entering a number from the preceding list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }

        var instanceChoice = allowedInstances[choiceNumber - 1];
        Console.WriteLine(sepBar);
        return instanceChoice;
    }

    /// <summary>
    /// Create a new DB instance.
    /// </summary>
    /// <param name="engineName">Engine to use for the DB instance.</param>
```

```
    /// <param name="engineVersion">Engine version to use for the DB instance.</
param>
    /// <param name="instanceClass">Instance class to use for the DB instance.</
param>
    /// <param name="instanceIdentifier">Instance identifier to use for the DB
 instance.</param>
    /// <returns>The new DB instance.</returns>
    public static async Task<DBInstance?> CreateNewInstance(
        string clusterIdentifier,
        string engineName,
        string engineVersion,
        string instanceClass,
        string instanceIdentifier)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine($"12. Create a new DB instance with identifier
 {instanceIdentifier}.");
        bool isInstanceReady = false;
        DBInstance newInstance;
        var instances = await auroraWrapper.DescribeDBInstancesPagedAsync();
        isInstanceReady = instances.FirstOrDefault(i =>
            i.DBInstanceIdentifier == instanceIdentifier)?.DBInstanceStatus ==
 "available";

        if (isInstanceReady)
        {
            Console.WriteLine("Instance already created.");
            newInstance = instances.First(i => i.DBInstanceIdentifier ==
 instanceIdentifier);
        }
        else
        {

            newInstance = await auroraWrapper.CreateDBInstanceInClusterAsync(
                clusterIdentifier,
                instanceIdentifier,
                engineName,
                engineVersion,
                instanceClass
            );

            Console.WriteLine("13. Waiting for DB instance to be ready...");
            while (!isInstanceReady)
            {
```

```
                Console.Write(".");
                Thread.Sleep(5000);
                instances = await
 auroraWrapper.DescribeDBInstancesPagedAsync(instanceIdentifier);
                isInstanceReady = instances.FirstOrDefault()?.DBInstanceStatus ==
 "available";
                newInstance = instances.First();
            }
        }

        Console.WriteLine(sepBar);
        return newInstance;
    }

    /// <summary>
    /// Display a connection string for an Amazon RDS DB cluster.
    /// </summary>
    /// <param name="cluster">The DB cluster to use to get a connection string.</
param>
    public static void DisplayConnectionString(DBCluster cluster)
    {
        Console.WriteLine(sepBar);
        // Display the connection string.
        Console.WriteLine("14. New DB cluster connection string: ");
        Console.WriteLine(
            $"\n{engine} -h {cluster.Endpoint} -P {cluster.Port} "
            + $"-u {cluster.MasterUsername} -p [YOUR PASSWORD]\n");

        Console.WriteLine(sepBar);
    }

    /// <summary>
    /// Create a snapshot from an Amazon RDS DB cluster.
    /// </summary>
    /// <param name="cluster">DB cluster to use when creating a snapshot.</param>
    /// <returns>The snapshot object.</returns>
    public static async Task<DBClusterSnapshot> CreateSnapshot(DBCluster cluster)
    {
        Console.WriteLine(sepBar);
        // Create a snapshot.
        Console.WriteLine($"15. Creating snapshot from DB cluster
 {cluster.DBClusterIdentifier}.");
        var snapshot = await
 auroraWrapper.CreateClusterSnapshotByIdentifierAsync(
```

```
            cluster.DBClusterIdentifier,
            "ExampleSnapshot-" + DateTime.Now.Ticks);

        // Wait for the snapshot to be available.
        bool isSnapshotReady = false;

        Console.WriteLine($"16. Waiting for snapshot to be ready...");
        while (!isSnapshotReady)
        {
            Console.Write(".");
            Thread.Sleep(5000);
            var snapshots =
                await
auroraWrapper.DescribeDBClusterSnapshotsByIdentifierAsync(cluster.DBClusterIdentifier);
            isSnapshotReady = snapshots.FirstOrDefault()?.Status == "available";
            snapshot = snapshots.First();
        }

        Console.WriteLine(
            $"Snapshot {snapshot.DBClusterSnapshotIdentifier} status is
{snapshot.Status}.");
        Console.WriteLine(sepBar);
        return snapshot;
    }

    /// <summary>
    /// Clean up resources from the scenario.
    /// </summary>
    /// <param name="newInstance">The instance to clean up.</param>
    /// <param name="newCluster">The cluster to clean up.</param>
    /// <param name="parameterGroup">The parameter group to clean up.</param>
    /// <returns>Async Task.</returns>
    private static async Task CleanupResources(
        DBInstance? newInstance,
        DBCluster? newCluster,
        DBClusterParameterGroup? parameterGroup)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Clean up resources.");

        if (newInstance is not null && GetYesNoResponse($"\tClean up instance
{newInstance.DBInstanceIdentifier}? (y/n)"))
        {
            // Delete the DB instance.
```

```
            Console.WriteLine($"17. Deleting the DB instance
{newInstance.DBInstanceIdentifier}.");
            await
auroraWrapper.DeleteDBInstanceByIdentifierAsync(newInstance.DBInstanceIdentifier);
        }

        if (newCluster is not null && GetYesNoResponse($"\tClean up cluster
{newCluster.DBClusterIdentifier}? (y/n)"))
        {
            // Delete the DB cluster.
            Console.WriteLine($"18. Deleting the DB cluster
{newCluster.DBClusterIdentifier}.");
            await
auroraWrapper.DeleteDBClusterByIdentifierAsync(newCluster.DBClusterIdentifier);

            // Wait for the DB cluster to delete.
            Console.WriteLine($"19. Waiting for the DB cluster to delete...");
            bool isClusterDeleted = false;

            while (!isClusterDeleted)
            {
                Console.Write(".");
                Thread.Sleep(5000);
                var cluster = await auroraWrapper.DescribeDBClustersPagedAsync();
                isClusterDeleted = cluster.All(i => i.DBClusterIdentifier !=
newCluster.DBClusterIdentifier);
            }

            Console.WriteLine("DB cluster deleted.");
        }

        if (parameterGroup is not null && GetYesNoResponse($"\tClean up parameter
group? (y/n)"))
        {
            Console.WriteLine($"20. Deleting the DB parameter group
{parameterGroup.DBClusterParameterGroupName}.");
            await
auroraWrapper.DeleteClusterParameterGroupByNameAsync(parameterGroup.DBClusterParameterGr
            Console.WriteLine("Parameter group deleted.");
        }

        Console.WriteLine(new string('-', 80));
    }
```

```
    /// <summary>
    /// Get a yes or no response from the user.
    /// </summary>
    /// <param name="question">The question string to print on the console.</
param>
    /// <returns>True if the user responds with a yes.</returns>
    private static bool GetYesNoResponse(string question)
    {
        Console.WriteLine(question);
        var ynResponse = Console.ReadLine();
        var response = ynResponse != null &&
                       ynResponse.Equals("y",
                           StringComparison.InvariantCultureIgnoreCase);
        return response;
    }
```

场景调用以管理 Aurora 操作的包装程序方法。

```
using Amazon.RDS;
using Amazon.RDS.Model;

namespace AuroraActions;

/// <summary>
/// Wrapper for the Amazon Aurora cluster client operations.
/// </summary>
public class AuroraWrapper
{
    private readonly IAmazonRDS _amazonRDS;
    public AuroraWrapper(IAmazonRDS amazonRDS)
    {
        _amazonRDS = amazonRDS;
    }

    /// <summary>
    /// Get a list of DB engine versions for a particular DB engine.
    /// </summary>
    /// <param name="engine">The name of the engine.</param>
    /// <param name="parameterGroupFamily">Optional parameter group family
 name.</param>
    /// <returns>A list of DBEngineVersions.</returns>
```

```
    public async Task<List<DBEngineVersion>>
 DescribeDBEngineVersionsForEngineAsync(string engine,
        string? parameterGroupFamily = null)
    {
        var response = await _amazonRDS.DescribeDBEngineVersionsAsync(
            new DescribeDBEngineVersionsRequest()
            {
                Engine = engine,
                DBParameterGroupFamily = parameterGroupFamily
            });
        return response.DBEngineVersions;
    }

    /// <summary>
    /// Create a custom cluster parameter group.
    /// </summary>
    /// <param name="parameterGroupFamily">The family of the parameter group.</
param>
    /// <param name="groupName">The name for the new parameter group.</param>
    /// <param name="description">A description for the new parameter group.</
param>
    /// <returns>The new parameter group object.</returns>
    public async Task<DBClusterParameterGroup>
 CreateCustomClusterParameterGroupAsync(
        string parameterGroupFamily,
        string groupName,
        string description)
    {
        var request = new CreateDBClusterParameterGroupRequest
        {
            DBParameterGroupFamily = parameterGroupFamily,
            DBClusterParameterGroupName = groupName,
            Description = description,
        };

        var response = await
 _amazonRDS.CreateDBClusterParameterGroupAsync(request);
        return response.DBClusterParameterGroup;
    }

    /// <summary>
    /// Describe the cluster parameters in a parameter group.
    /// </summary>
    /// <param name="groupName">The name of the parameter group.</param>
```

```
    /// <param name="source">The optional name of the source filter.</param>
    /// <returns>The collection of parameters.</returns>
    public async Task<List<Parameter>>
 DescribeDBClusterParametersInGroupAsync(string groupName, string? source = null)
    {
        var paramList = new List<Parameter>();

        DescribeDBClusterParametersResponse response;
        var request = new DescribeDBClusterParametersRequest
        {
            DBClusterParameterGroupName = groupName,
            Source = source,
        };

        // Get the full list if there are multiple pages.
        do
        {
            response = await
 _amazonRDS.DescribeDBClusterParametersAsync(request);
            paramList.AddRange(response.Parameters);

            request.Marker = response.Marker;
        }
        while (response.Marker is not null);

        return paramList;
    }

    /// <summary>
    /// Get the description of a DB cluster parameter group by name.
    /// </summary>
    /// <param name="name">The name of the DB parameter group to describe.</
param>
    /// <returns>The parameter group description.</returns>
    public async Task<DBClusterParameterGroup?>
 DescribeCustomDBClusterParameterGroupAsync(string name)
    {
        var response = await _amazonRDS.DescribeDBClusterParameterGroupsAsync(
            new DescribeDBClusterParameterGroupsRequest()
            {
                DBClusterParameterGroupName = name
            });
        return response.DBClusterParameterGroups.FirstOrDefault();
    }
```

```
    /// <summary>
    /// Modify the specified integer parameters with new values from user input.
    /// </summary>
    /// <param name="groupName">The group name for the parameters.</param>
    /// <param name="parameters">The list of integer parameters to modify.</
param>
    /// <param name="newValue">Optional int value to set for parameters.</param>
    /// <returns>The name of the group that was modified.</returns>
    public async Task<string> ModifyIntegerParametersInGroupAsync(string
 groupName, List<Parameter> parameters, int newValue = 0)
    {
        foreach (var p in parameters)
        {
            if (p.IsModifiable && p.DataType == "integer")
            {
                while (newValue == 0)
                {
                    Console.WriteLine(
                        $"Enter a new value for {p.ParameterName} from the
 allowed values {p.AllowedValues} ");

                    var choice = Console.ReadLine();
                    int.TryParse(choice, out newValue);
                }

                p.ParameterValue = newValue.ToString();
            }
        }

        var request = new ModifyDBClusterParameterGroupRequest
        {
            Parameters = parameters,
            DBClusterParameterGroupName = groupName,
        };

        var result = await
 _amazonRDS.ModifyDBClusterParameterGroupAsync(request);
        return result.DBClusterParameterGroupName;
    }


    /// <summary>
    /// Get a list of orderable DB instance options for a specific
```

```
    /// engine and engine version.
    /// </summary>
    /// <param name="engine">Name of the engine.</param>
    /// <param name="engineVersion">Version of the engine.</param>
    /// <returns>List of OrderableDBInstanceOptions.</returns>
    public async Task<List<OrderableDBInstanceOption>>
DescribeOrderableDBInstanceOptionsPagedAsync(string engine, string
engineVersion)
    {
        // Use a paginator to get a list of DB instance options.
        var results = new List<OrderableDBInstanceOption>();
        var paginateInstanceOptions =
_amazonRDS.Paginators.DescribeOrderableDBInstanceOptions(
            new DescribeOrderableDBInstanceOptionsRequest()
            {
                Engine = engine,
                EngineVersion = engineVersion,
            });
        // Get the entire list using the paginator.
        await foreach (var instanceOptions in
paginateInstanceOptions.OrderableDBInstanceOptions)
        {
            results.Add(instanceOptions);
        }
        return results;
    }

    /// <summary>
    /// Delete a particular parameter group by name.
    /// </summary>
    /// <param name="groupName">The name of the parameter group.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteClusterParameterGroupByNameAsync(string
groupName)
    {
        var request = new DeleteDBClusterParameterGroupRequest
        {
            DBClusterParameterGroupName = groupName,
        };

        var response = await
_amazonRDS.DeleteDBClusterParameterGroupAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

```
    /// <summary>
    /// Create a new cluster and database.
    /// </summary>
    /// <param name="dbName">The name of the new database.</param>
    /// <param name="clusterIdentifier">The identifier of the cluster.</param>
    /// <param name="parameterGroupName">The name of the parameter group.</param>
    /// <param name="dbEngine">The engine to use for the new cluster.</param>
    /// <param name="dbEngineVersion">The version of the engine to use.</param>
    /// <param name="adminName">The admin username.</param>
    /// <param name="adminPassword">The primary admin password.</param>
    /// <returns>The cluster object.</returns>
    public async Task<DBCluster> CreateDBClusterWithAdminAsync(
        string dbName,
        string clusterIdentifier,
        string parameterGroupName,
        string dbEngine,
        string dbEngineVersion,
        string adminName,
        string adminPassword)
    {
        var request = new CreateDBClusterRequest
        {
            DatabaseName = dbName,
            DBClusterIdentifier = clusterIdentifier,
            DBClusterParameterGroupName = parameterGroupName,
            Engine = dbEngine,
            EngineVersion = dbEngineVersion,
            MasterUsername = adminName,
            MasterUserPassword = adminPassword,
        };

        var response = await _amazonRDS.CreateDBClusterAsync(request);
        return response.DBCluster;
    }

    /// <summary>
    /// Returns a list of DB instances.
    /// </summary>
    /// <param name="dbInstanceIdentifier">Optional name of a specific DB
instance.</param>
    /// <returns>List of DB instances.</returns>
    public async Task<List<DBInstance>> DescribeDBInstancesPagedAsync(string?
dbInstanceIdentifier = null)
```

```
    {
        var results = new List<DBInstance>();
        var instancesPaginator = _amazonRDS.Paginators.DescribeDBInstances(
            new DescribeDBInstancesRequest
            {
                DBInstanceIdentifier = dbInstanceIdentifier
            });
        // Get the entire list using the paginator.
        await foreach (var instances in instancesPaginator.DBInstances)
        {
            results.Add(instances);
        }
        return results;
    }

    /// <summary>
    /// Returns a list of DB clusters.
    /// </summary>
    /// <param name="dbInstanceIdentifier">Optional name of a specific DB
cluster.</param>
    /// <returns>List of DB clusters.</returns>
    public async Task<List<DBCluster>> DescribeDBClustersPagedAsync(string?
dbClusterIdentifier = null)
    {
        var results = new List<DBCluster>();

        DescribeDBClustersResponse response;
        DescribeDBClustersRequest request = new DescribeDBClustersRequest
        {
            DBClusterIdentifier = dbClusterIdentifier
        };
        // Get the full list if there are multiple pages.
        do
        {
            response = await _amazonRDS.DescribeDBClustersAsync(request);
            results.AddRange(response.DBClusters);
            request.Marker = response.Marker;
        }
        while (response.Marker is not null);
        return results;
    }

    /// <summary>
    /// Create an Amazon Relational Database Service (Amazon RDS) DB instance
```

```
    /// with a particular set of properties. Use the action
DescribeDBInstancesAsync
    /// to determine when the DB instance is ready to use.
    /// </summary>
    /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <param name="dbEngine">The engine for the DB instance.</param>
    /// <param name="dbEngineVersion">Version for the DB instance.</param>
    /// <param name="instanceClass">Class for the DB instance.</param>
    /// <returns>DB instance object.</returns>
    public async Task<DBInstance> CreateDBInstanceInClusterAsync(
        string dbClusterIdentifier,
        string dbInstanceIdentifier,
        string dbEngine,
        string dbEngineVersion,
        string instanceClass)
    {
        // When creating the instance within a cluster, do not specify the name
or size.
        var response = await _amazonRDS.CreateDBInstanceAsync(
            new CreateDBInstanceRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                DBInstanceIdentifier = dbInstanceIdentifier,
                Engine = dbEngine,
                EngineVersion = dbEngineVersion,
                DBInstanceClass = instanceClass
            });

        return response.DBInstance;
    }

    /// <summary>
    /// Create a snapshot of a cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <param name="snapshotIdentifier">Identifier for the snapshot.</param>
    /// <returns>DB snapshot object.</returns>
    public async Task<DBClusterSnapshot>
CreateClusterSnapshotByIdentifierAsync(string dbClusterIdentifier, string
snapshotIdentifier)
    {
        var response = await _amazonRDS.CreateDBClusterSnapshotAsync(
            new CreateDBClusterSnapshotRequest()
```

```
            {
                DBClusterIdentifier = dbClusterIdentifier,
                DBClusterSnapshotIdentifier = snapshotIdentifier,
            });

        return response.DBClusterSnapshot;
    }

    /// <summary>
    /// Return a list of DB snapshots for a particular DB cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <returns>List of DB snapshots.</returns>
    public async Task<List<DBClusterSnapshot>>
DescribeDBClusterSnapshotsByIdentifierAsync(string dbClusterIdentifier)
    {
        var results = new List<DBClusterSnapshot>();

        DescribeDBClusterSnapshotsResponse response;
        DescribeDBClusterSnapshotsRequest request = new
DescribeDBClusterSnapshotsRequest
        {
            DBClusterIdentifier = dbClusterIdentifier
        };
        // Get the full list if there are multiple pages.
        do
        {
            response = await _amazonRDS.DescribeDBClusterSnapshotsAsync(request);
            results.AddRange(response.DBClusterSnapshots);
            request.Marker = response.Marker;
        }
        while (response.Marker is not null);
        return results;
    }

    /// <summary>
    /// Delete a particular DB cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <returns>DB cluster object.</returns>
    public async Task<DBCluster> DeleteDBClusterByIdentifierAsync(string
dbClusterIdentifier)
    {
        var response = await _amazonRDS.DeleteDBClusterAsync(
```

```
            new DeleteDBClusterRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                SkipFinalSnapshot = true
            });

        return response.DBCluster;
    }

    /// <summary>
    /// Delete a particular DB instance.
    /// </summary>
    /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
    /// <returns>DB instance object.</returns>
    public async Task<DBInstance> DeleteDBInstanceByIdentifierAsync(string
 dbInstanceIdentifier)
    {
        var response = await _amazonRDS.DeleteDBInstanceAsync(
            new DeleteDBInstanceRequest()
            {
                DBInstanceIdentifier = dbInstanceIdentifier,
                SkipFinalSnapshot = true,
                DeleteAutomatedBackups = true
            });

        return response.DBInstance;
    }
}
```

- 有关 API 详细信息，请参阅《适用于 .NET 的 AWS SDK API 参考》中的以下主题。
  - CreateDBCluster
  - CreateDBClusterParameterGroup
  - CreateDBClusterSnapshot
  - CreateDBInstance
  - DeleteDBCluster
  - DeleteDBClusterParameterGroup
  - DeleteDBInstance
  - DescribeDBClusterParameterGroups
  - DescribeDBClusterParameters

- DescribeDBClusterSnapshots
- DescribeDBClusters
- DescribeDBEngineVersions
- DescribeDBInstances
- DescribeOrderableDBInstanceOptions
- ModifyDBClusterParameterGroup

C++

SDK for C++

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

//! Routine which creates an Amazon Aurora DB cluster and demonstrates several
 operations
//! on that cluster.
/*!
 \sa gettingStartedWithDBClusters()
 \param clientConfiguration: AWS client configuration.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::gettingStartedWithDBClusters(
        const Aws::Client::ClientConfiguration &clientConfig) {
    Aws::RDS::RDSClient client(clientConfig);

    printAsterisksLine();
    std::cout << "Welcome to the Amazon Relational Database Service (Amazon
 Aurora)"
              << std::endl;
    std::cout << "get started with DB clusters demo." << std::endl;
    printAsterisksLine();
```

```
    std::cout << "Checking for an existing DB cluster parameter group named '" <<
              CLUSTER_PARAMETER_GROUP_NAME << "'." << std::endl;
    Aws::String dbParameterGroupFamily("Undefined");
    bool parameterGroupFound = true;
    {
        // 1. Check if the DB cluster parameter group already exists.
        Aws::RDS::Model::DescribeDBClusterParameterGroupsRequest request;
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);

        Aws::RDS::Model::DescribeDBClusterParameterGroupsOutcome outcome =
                client.DescribeDBClusterParameterGroups(request);

        if (outcome.IsSuccess()) {
            std::cout << "DB cluster parameter group named '" <<
                      CLUSTER_PARAMETER_GROUP_NAME << "' already exists." <<
 std::endl;
            dbParameterGroupFamily =
 outcome.GetResult().GetDBClusterParameterGroups()
[0].GetDBParameterGroupFamily();
        }
        else if (outcome.GetError().GetErrorType() ==
                 Aws::RDS::RDSErrors::D_B_PARAMETER_GROUP_NOT_FOUND_FAULT) {
            std::cout << "DB cluster parameter group named '" <<
                      CLUSTER_PARAMETER_GROUP_NAME << "' does not exist." <<
 std::endl;
            parameterGroupFound = false;
        }
        else {
            std::cerr << "Error with Aurora::DescribeDBClusterParameterGroups. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    }

    if (!parameterGroupFound) {
        Aws::Vector<Aws::RDS::Model::DBEngineVersion> engineVersions;

        // 2. Get available parameter group families for the specified engine.
        if (!getDBEngineVersions(DB_ENGINE, NO_PARAMETER_GROUP_FAMILY,
                                 engineVersions, client)) {
            return false;
        }
```

```
        std::cout << "Getting available parameter group families for " <<
DB_ENGINE
                  << "."
                  << std::endl;
        std::vector<Aws::String> families;
        for (const Aws::RDS::Model::DBEngineVersion &version: engineVersions) {
            Aws::String family = version.GetDBParameterGroupFamily();
            if (std::find(families.begin(), families.end(), family) ==
                families.end()) {
                families.push_back(family);
                std::cout << "  " << families.size() << ": " << family <<
std::endl;
            }
        }

        int choice = askQuestionForIntRange("Which family do you want to use? ",
1,
                                            static_cast<int>(families.size()));
        dbParameterGroupFamily = families[choice - 1];
    }
    if (!parameterGroupFound) {
        // 3.  Create a DB cluster parameter group.
        Aws::RDS::Model::CreateDBClusterParameterGroupRequest request;
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);
        request.SetDBParameterGroupFamily(dbParameterGroupFamily);
        request.SetDescription("Example cluster parameter group.");

        Aws::RDS::Model::CreateDBClusterParameterGroupOutcome outcome =
                client.CreateDBClusterParameterGroup(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB cluster parameter group was successfully
created."
                      << std::endl;
        }
        else {
            std::cerr << "Error with Aurora::CreateDBClusterParameterGroup. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    }
```

```
   printAsterisksLine();
   std::cout << "Let's set some parameter values in your cluster parameter
group."
             << std::endl;

   Aws::Vector<Aws::RDS::Model::Parameter> autoIncrementParameters;
   // 4.  Get the parameters in the DB cluster parameter group.
   if (!getDBClusterParameters(CLUSTER_PARAMETER_GROUP_NAME,
AUTO_INCREMENT_PREFIX,
                               NO_SOURCE,
                               autoIncrementParameters,
                               client)) {
       cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, "", "", client);
       return false;
   }

   Aws::Vector<Aws::RDS::Model::Parameter> updateParameters;

   for (Aws::RDS::Model::Parameter &autoIncParameter: autoIncrementParameters) {
       if (autoIncParameter.GetIsModifiable() &&
           (autoIncParameter.GetDataType() == "integer")) {
           std::cout << "The " << autoIncParameter.GetParameterName()
                     << " is described as: " <<
                     autoIncParameter.GetDescription() << "." << std::endl;
           if (autoIncParameter.ParameterValueHasBeenSet()) {
               std::cout << "The current value is "
                         << autoIncParameter.GetParameterValue()
                         << "." << std::endl;
           }
           std::vector<int> splitValues = splitToInts(
                   autoIncParameter.GetAllowedValues(), '-');
           if (splitValues.size() == 2) {
               int newValue = askQuestionForIntRange(
                       Aws::String("Enter a new value between ") +
                       autoIncParameter.GetAllowedValues() + ": ",
                       splitValues[0], splitValues[1]);
               autoIncParameter.SetParameterValue(std::to_string(newValue));
               updateParameters.push_back(autoIncParameter);

           }
           else {
               std::cerr << "Error parsing " <<
autoIncParameter.GetAllowedValues()
                         << std::endl;
```

```
            }
        }
    }

    {
        // 5.  Modify the auto increment parameters in the DB cluster parameter
group.
        Aws::RDS::Model::ModifyDBClusterParameterGroupRequest request;
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);
        request.SetParameters(updateParameters);

        Aws::RDS::Model::ModifyDBClusterParameterGroupOutcome outcome =
                client.ModifyDBClusterParameterGroup(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB cluster parameter group was successfully
modified."
                      << std::endl;
        }
        else {
            std::cerr << "Error with Aurora::ModifyDBClusterParameterGroup. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
        }
    }

    std::cout
            << "You can get a list of parameters you've set by specifying a
source of 'user'."
            << std::endl;

    Aws::Vector<Aws::RDS::Model::Parameter> userParameters;
    // 6.  Display the modified parameters in the DB cluster parameter group.
    if (!getDBClusterParameters(CLUSTER_PARAMETER_GROUP_NAME, NO_NAME_PREFIX,
"user",
                                userParameters,
                                client)) {
        cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, "", "", client);
        return false;
    }

    for (const auto &userParameter: userParameters) {
        std::cout << "   " << userParameter.GetParameterName() << ", " <<
                  userParameter.GetDescription() << ", parameter value - "
```

```
                << userParameter.GetParameterValue() << std::endl;
    }

    printAsterisksLine();
    std::cout << "Checking for an existing DB Cluster." << std::endl;

    Aws::RDS::Model::DBCluster dbCluster;
    // 7.  Check if the DB cluster already exists.
    if (!describeDBCluster(DB_CLUSTER_IDENTIFIER, dbCluster, client)) {
        cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, "", "", client);
        return false;
    }

    Aws::String engineVersionName;
    Aws::String engineName;
    if (dbCluster.DBClusterIdentifierHasBeenSet()) {
        std::cout << "The DB cluster already exists." << std::endl;
        engineVersionName = dbCluster.GetEngineVersion();
        engineName = dbCluster.GetEngine();

    }
    else {
        std::cout << "Let's create a DB cluster." << std::endl;
        const Aws::String administratorName = askQuestion(
                "Enter an administrator username for the database: ");
        const Aws::String administratorPassword = askQuestion(
                "Enter a password for the administrator (at least 8 characters):
");
        Aws::Vector<Aws::RDS::Model::DBEngineVersion> engineVersions;

        // 8.  Get a list of engine versions for the parameter group family.
        if (!getDBEngineVersions(DB_ENGINE, dbParameterGroupFamily,
engineVersions,
                                 client)) {
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, "", "", client);
            return false;
        }

        std::cout << "The available engines for your parameter group family are:"
                  << std::endl;

        int index = 1;
        for (const Aws::RDS::Model::DBEngineVersion &engineVersion:
engineVersions) {
```

```
            std::cout << "  " << index << ": " <<
engineVersion.GetEngineVersion()
                      << std::endl;
            ++index;
        }
        int choice = askQuestionForIntRange("Which engine do you want to use? ",
1,

static_cast<int>(engineVersions.size()));
        const Aws::RDS::Model::DBEngineVersion engineVersion =
engineVersions[choice -
                                                                              1];

        engineName = engineVersion.GetEngine();
        engineVersionName = engineVersion.GetEngineVersion();
        std::cout << "Creating a DB cluster named '" << DB_CLUSTER_IDENTIFIER
                  << "' and database '" << DB_NAME << "'.\n"
                  << "The DB cluster is configured to use your custom cluster
parameter group '"
                  << CLUSTER_PARAMETER_GROUP_NAME << "', and \n"
                  << "selected engine version " <<
engineVersion.GetEngineVersion()
                  << ".\nThis typically takes several minutes." << std::endl;

        Aws::RDS::Model::CreateDBClusterRequest request;
        request.SetDBClusterIdentifier(DB_CLUSTER_IDENTIFIER);
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);
        request.SetEngine(engineName);
        request.SetEngineVersion(engineVersionName);
        request.SetMasterUsername(administratorName);
        request.SetMasterUserPassword(administratorPassword);

        Aws::RDS::Model::CreateDBClusterOutcome outcome =
                client.CreateDBCluster(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB cluster creation has started."
                      << std::endl;
        }
        else {
            std::cerr << "Error with Aurora::CreateDBCluster. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, "", "", client);
```

```
            return false;
        }
    }

    std::cout << "Waiting for the DB cluster to become available." << std::endl;

    int counter = 0;
    // 11. Wait for the DB cluster to become available.
    do {
        std::this_thread::sleep_for(std::chrono::seconds(1));
        ++counter;
        if (counter > 900) {
            std::cerr << "Wait for cluster to become available timed out ofter "
                      << counter
                      << " seconds." << std::endl;
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                             DB_CLUSTER_IDENTIFIER, "", client);
            return false;
        }

        dbCluster = Aws::RDS::Model::DBCluster();
        if (!describeDBCluster(DB_CLUSTER_IDENTIFIER, dbCluster, client)) {
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                             DB_CLUSTER_IDENTIFIER, "", client);
            return false;
        }

        if ((counter % 20) == 0) {
            std::cout << "Current DB cluster status is '"
                      << dbCluster.GetStatus()
                      << "' after " << counter << " seconds." << std::endl;
        }
    } while (dbCluster.GetStatus() != "available");

    if (dbCluster.GetStatus() == "available") {
        std::cout << "The DB cluster has been created." << std::endl;
    }

    printAsterisksLine();
    Aws::RDS::Model::DBInstance dbInstance;
    // 11.  Check if the DB instance already exists.
    if (!describeDBInstance(DB_INSTANCE_IDENTIFIER, dbInstance, client)) {
        cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, DB_CLUSTER_IDENTIFIER, "",
                         client);
```

```
        return false;
    }

    if (dbInstance.DbInstancePortHasBeenSet()) {
        std::cout << "The DB instance already exists." << std::endl;
    }
    else {
        std::cout << "Let's create a DB instance." << std::endl;

        Aws::String dbInstanceClass;
        // 12.  Get a list of instance classes.
        if (!chooseDBInstanceClass(engineName,
                                   engineVersionName,
                                   dbInstanceClass,
                                   client)) {
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, DB_CLUSTER_IDENTIFIER,
"",
                             client);
            return false;
        }

        std::cout << "Creating a DB instance named '" << DB_INSTANCE_IDENTIFIER
                  << "' with selected DB instance class '" << dbInstanceClass
                  << "'.\nThis typically takes several minutes." << std::endl;

        // 13. Create a DB instance.
        Aws::RDS::Model::CreateDBInstanceRequest request;
        request.SetDBInstanceIdentifier(DB_INSTANCE_IDENTIFIER);
        request.SetDBClusterIdentifier(DB_CLUSTER_IDENTIFIER);
        request.SetEngine(engineName);
        request.SetDBInstanceClass(dbInstanceClass);

        Aws::RDS::Model::CreateDBInstanceOutcome outcome =
                client.CreateDBInstance(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB instance creation has started."
                      << std::endl;
        }
        else {
            std::cerr << "Error with RDS::CreateDBInstance. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
```

```
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, DB_CLUSTER_IDENTIFIER,
"",
                             client);
            return false;
        }
    }

    std::cout << "Waiting for the DB instance to become available." << std::endl;

    counter = 0;
    // 14. Wait for the DB instance to become available.
    do {
        std::this_thread::sleep_for(std::chrono::seconds(1));
        ++counter;
        if (counter > 900) {
            std::cerr << "Wait for instance to become available timed out ofter "
                      << counter
                      << " seconds." << std::endl;
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                             DB_CLUSTER_IDENTIFIER, DB_INSTANCE_IDENTIFIER,
client);
            return false;
        }

        dbInstance = Aws::RDS::Model::DBInstance();
        if (!describeDBInstance(DB_INSTANCE_IDENTIFIER, dbInstance, client)) {
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                             DB_CLUSTER_IDENTIFIER, DB_INSTANCE_IDENTIFIER,
client);
            return false;
        }

        if ((counter % 20) == 0) {
            std::cout << "Current DB instance status is '"
                      << dbInstance.GetDBInstanceStatus()
                      << "' after " << counter << " seconds." << std::endl;
        }
    } while (dbInstance.GetDBInstanceStatus() != "available");

    if (dbInstance.GetDBInstanceStatus() == "available") {
        std::cout << "The DB instance has been created." << std::endl;
    }
```

```
   // 15. Display the connection string that can be used to connect a 'mysql'
shell to the database.
   displayConnection(dbCluster);

   printAsterisksLine();

   if (askYesNoQuestion(
           "Do you want to create a snapshot of your DB cluster (y/n)? ")) {
       Aws::String snapshotID(DB_CLUSTER_IDENTIFIER + "-" +
                              Aws::String(Aws::Utils::UUID::RandomUUID()));
       {
           std::cout << "Creating a snapshot named " << snapshotID << "." <<
std::endl;
           std::cout << "This typically takes a few minutes." << std::endl;

           // 16. Create a snapshot of the DB cluster. (CreateDBClusterSnapshot)
           Aws::RDS::Model::CreateDBClusterSnapshotRequest request;
           request.SetDBClusterIdentifier(DB_CLUSTER_IDENTIFIER);
           request.SetDBClusterSnapshotIdentifier(snapshotID);

           Aws::RDS::Model::CreateDBClusterSnapshotOutcome outcome =
                   client.CreateDBClusterSnapshot(request);

           if (outcome.IsSuccess()) {
               std::cout << "Snapshot creation has started."
                         << std::endl;
           }
           else {
               std::cerr << "Error with Aurora::CreateDBClusterSnapshot. "
                         << outcome.GetError().GetMessage()
                         << std::endl;
               cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                                DB_CLUSTER_IDENTIFIER, DB_INSTANCE_IDENTIFIER,
client);
               return false;
           }
       }

       std::cout << "Waiting for the snapshot to become available." <<
std::endl;

       Aws::RDS::Model::DBClusterSnapshot snapshot;
       counter = 0;
       do {
```

```
            std::this_thread::sleep_for(std::chrono::seconds(1));
            ++counter;
            if (counter > 600) {
                std::cerr << "Wait for snapshot to be available timed out ofter "
                          << counter
                          << " seconds." << std::endl;
                cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                                 DB_CLUSTER_IDENTIFIER, DB_INSTANCE_IDENTIFIER,
client);
                return false;
            }

            // 17. Wait for the snapshot to become available.
            Aws::RDS::Model::DescribeDBClusterSnapshotsRequest request;
            request.SetDBClusterSnapshotIdentifier(snapshotID);

            Aws::RDS::Model::DescribeDBClusterSnapshotsOutcome outcome =
                    client.DescribeDBClusterSnapshots(request);

            if (outcome.IsSuccess()) {
                snapshot = outcome.GetResult().GetDBClusterSnapshots()[0];
            }
            else {
                std::cerr << "Error with Aurora::DescribeDBClusterSnapshots. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                                 DB_CLUSTER_IDENTIFIER, DB_INSTANCE_IDENTIFIER,
client);
                return false;
            }

            if ((counter % 20) == 0) {
                std::cout << "Current snapshot status is '"
                          << snapshot.GetStatus()
                          << "' after " << counter << " seconds." << std::endl;
            }
        } while (snapshot.GetStatus() != "available");

        if (snapshot.GetStatus() != "available") {
            std::cout << "A snapshot has been created." << std::endl;
        }
    }
```

```
    printAsterisksLine();

    bool result = true;
    if (askYesNoQuestion(
            "Do you want to delete the DB cluster, DB instance, and parameter
 group (y/n)? ")) {
        result = cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                                  DB_CLUSTER_IDENTIFIER, DB_INSTANCE_IDENTIFIER,
                                  client);
    }

    return result;
}

//! Routine which gets a DB cluster description.
/*!
 \sa describeDBCluster()
 \param dbClusterIdentifier: A DB cluster identifier.
 \param clusterResult: The 'DBCluster' object containing the description.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::describeDBCluster(const Aws::String &dbClusterIdentifier,
                                       Aws::RDS::Model::DBCluster &clusterResult,
                                       const Aws::RDS::RDSClient &client) {
    Aws::RDS::Model::DescribeDBClustersRequest request;
    request.SetDBClusterIdentifier(dbClusterIdentifier);

    Aws::RDS::Model::DescribeDBClustersOutcome outcome =
            client.DescribeDBClusters(request);

    bool result = true;
    if (outcome.IsSuccess()) {
        clusterResult = outcome.GetResult().GetDBClusters()[0];
    }
    else if (outcome.GetError().GetErrorType() !=
             Aws::RDS::RDSErrors::D_B_CLUSTER_NOT_FOUND_FAULT) {
        result = false;
        std::cerr << "Error with Aurora::GDescribeDBClusters. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
    }
        // This example does not log an error if the DB cluster does not exist.
        // Instead, clusterResult is set to empty.
```

```
    else {
        clusterResult = Aws::RDS::Model::DBCluster();
    }

    return result;

}


//! Routine which gets DB parameters using the 'DescribeDBClusterParameters' api.
/*!
 \sa getDBCLusterParameters()
 \param parameterGroupName: The name of the cluster parameter group.
 \param namePrefix: Prefix string to filter results by parameter name.
 \param source: A source such as 'user', ignored if empty.
 \param parametersResult: Vector of 'Parameter' objects returned by the routine.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::getDBCLusterParameters(const Aws::String
 &parameterGroupName,
                                            const Aws::String &namePrefix,
                                            const Aws::String &source,

 Aws::Vector<Aws::RDS::Model::Parameter> &parametersResult,
                                            const Aws::RDS::RDSClient &client) {
    Aws::String marker; // The marker is used for pagination.
    do {
        Aws::RDS::Model::DescribeDBClusterParametersRequest request;
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);
        if (!marker.empty()) {
            request.SetMarker(marker);
        }
        if (!source.empty()) {
            request.SetSource(source);
        }

        Aws::RDS::Model::DescribeDBClusterParametersOutcome outcome =
                client.DescribeDBClusterParameters(request);

        if (outcome.IsSuccess()) {
            const Aws::Vector<Aws::RDS::Model::Parameter> &parameters =
                    outcome.GetResult().GetParameters();
            for (const Aws::RDS::Model::Parameter &parameter: parameters) {
```

```
                if (!namePrefix.empty()) {
                    if (parameter.GetParameterName().find(namePrefix) == 0) {
                        parametersResult.push_back(parameter);
                    }
                }
                else {
                    parametersResult.push_back(parameter);
                }
            }

            marker = outcome.GetResult().GetMarker();
        }
        else {
            std::cerr << "Error with Aurora::DescribeDBClusterParameters. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    } while (!marker.empty());

    return true;
}


//! Routine which gets available DB engine versions for an engine name and
//! an optional parameter group family.
/*!
 \sa getDBEngineVersions()
 \param engineName: A DB engine name.
 \param parameterGroupFamily: A parameter group family name, ignored if empty.
 \param engineVersionsResult: Vector of 'DBEngineVersion' objects returned by the
 routine.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::getDBEngineVersions(const Aws::String &engineName,
                                         const Aws::String &parameterGroupFamily,

 Aws::Vector<Aws::RDS::Model::DBEngineVersion> &engineVersionsResult,
                                         const Aws::RDS::RDSClient &client) {
    Aws::RDS::Model::DescribeDBEngineVersionsRequest request;
    request.SetEngine(engineName);
    if (!parameterGroupFamily.empty()) {
        request.SetDBParameterGroupFamily(parameterGroupFamily);
```

```
    }

    engineVersionsResult.clear();
    Aws::String marker; // The marker is used for pagination.
    do {
        if (!marker.empty()) {
            request.SetMarker(marker);
        }

        Aws::RDS::Model::DescribeDBEngineVersionsOutcome outcome =
                client.DescribeDBEngineVersions(request);

        if (outcome.IsSuccess()) {
            const Aws::Vector<Aws::RDS::Model::DBEngineVersion> &engineVersions =
                    outcome.GetResult().GetDBEngineVersions();

            engineVersionsResult.insert(engineVersionsResult.end(),
                                        engineVersions.begin(),
 engineVersions.end());
            marker  = outcome.GetResult().GetMarker();
        }
        else {
            std::cerr << "Error with Aurora::DescribeDBEngineVersionsRequest. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
        }
    } while (!marker.empty());

    return true;
}


//! Routine which gets a DB instance description.
/*!
 \sa describeDBCluster()
 \param dbInstanceIdentifier: A DB instance identifier.
 \param instanceResult: The 'DBInstance' object containing the description.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::describeDBInstance(const Aws::String &dbInstanceIdentifier,
                                        Aws::RDS::Model::DBInstance
 &instanceResult,
                                        const Aws::RDS::RDSClient &client) {
```

```
    Aws::RDS::Model::DescribeDBInstancesRequest request;
    request.SetDBInstanceIdentifier(dbInstanceIdentifier);

    Aws::RDS::Model::DescribeDBInstancesOutcome outcome =
            client.DescribeDBInstances(request);

    bool result = true;
    if (outcome.IsSuccess()) {
        instanceResult = outcome.GetResult().GetDBInstances()[0];
    }
    else if (outcome.GetError().GetErrorType() !=
             Aws::RDS::RDSErrors::D_B_INSTANCE_NOT_FOUND_FAULT) {
        result = false;
        std::cerr << "Error with Aurora::DescribeDBInstances. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
    }
        // This example does not log an error if the DB instance does not exist.
        // Instead, instanceResult is set to empty.
    else {
        instanceResult = Aws::RDS::Model::DBInstance();
    }

    return result;
}


//! Routine which gets available DB instance classes, displays the list
//! to the user, and returns the user selection.
/*!
 \sa chooseDBInstanceClass()
 \param engineName: The DB engine name.
 \param engineVersion: The DB engine version.
 \param dbInstanceClass: String for DB instance class chosen by the user.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::chooseDBInstanceClass(const Aws::String &engine,
                                           const Aws::String &engineVersion,
                                           Aws::String &dbInstanceClass,
                                           const Aws::RDS::RDSClient &client) {
    std::vector<Aws::String> instanceClasses;
    Aws::String marker; // The marker is used for pagination.
    do {
```

```
        Aws::RDS::Model::DescribeOrderableDBInstanceOptionsRequest request;
        request.SetEngine(engine);
        request.SetEngineVersion(engineVersion);
        if (!marker.empty()) {
            request.SetMarker(marker);
        }

        Aws::RDS::Model::DescribeOrderableDBInstanceOptionsOutcome outcome =
                client.DescribeOrderableDBInstanceOptions(request);

        if (outcome.IsSuccess()) {
            const Aws::Vector<Aws::RDS::Model::OrderableDBInstanceOption>
&options =
                    outcome.GetResult().GetOrderableDBInstanceOptions();
            for (const Aws::RDS::Model::OrderableDBInstanceOption &option:
options) {
                const Aws::String &instanceClass = option.GetDBInstanceClass();
                if (std::find(instanceClasses.begin(), instanceClasses.end(),
                              instanceClass) == instanceClasses.end()) {
                    instanceClasses.push_back(instanceClass);
                }
            }
            marker = outcome.GetResult().GetMarker();
        }
        else {
            std::cerr << "Error with Aurora::DescribeOrderableDBInstanceOptions.
"
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    } while (!marker.empty());

    std::cout << "The available DB instance classes for your database engine
are:"
              << std::endl;
    for (int i = 0; i < instanceClasses.size(); ++i) {
        std::cout << "   " << i + 1 << ": " << instanceClasses[i] << std::endl;
    }

    int choice = askQuestionForIntRange(
            "Which DB instance class do you want to use? ",
            1, static_cast<int>(instanceClasses.size()));
    dbInstanceClass = instanceClasses[choice - 1];
```

```
    return true;
}

//! Routine which deletes resources created by the scenario.
/*!
\sa cleanUpResources()
\param parameterGroupName: A parameter group name, this may be empty.
\param dbInstanceIdentifier: A DB instance identifier, this may be empty.
\param client: 'RDSClient' instance.
\return bool: Successful completion.
*/
bool AwsDoc::Aurora::cleanUpResources(const Aws::String &parameterGroupName,
                                      const Aws::String &dbClusterIdentifier,
                                      const Aws::String &dbInstanceIdentifier,
                                      const Aws::RDS::RDSClient &client) {
    bool result = true;
    bool instanceDeleting = false;
    bool clusterDeleting = false;
    if (!dbInstanceIdentifier.empty()) {
        {
            // 18. Delete the DB instance.
            Aws::RDS::Model::DeleteDBInstanceRequest request;
            request.SetDBInstanceIdentifier(dbInstanceIdentifier);
            request.SetSkipFinalSnapshot(true);
            request.SetDeleteAutomatedBackups(true);

            Aws::RDS::Model::DeleteDBInstanceOutcome outcome =
                    client.DeleteDBInstance(request);

            if (outcome.IsSuccess()) {
                std::cout << "DB instance deletion has started."
                          << std::endl;
                instanceDeleting = true;
                std::cout
                        << "Waiting for DB instance to delete before deleting the
 parameter group."
                        << std::endl;
            }
            else {
                std::cerr << "Error with Aurora::DeleteDBInstance. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                result = false;
            }
```

```
        }
    }

    if (!dbClusterIdentifier.empty()) {
        {
            // 19. Delete the DB cluster.
            Aws::RDS::Model::DeleteDBClusterRequest request;
            request.SetDBClusterIdentifier(dbClusterIdentifier);
            request.SetSkipFinalSnapshot(true);

            Aws::RDS::Model::DeleteDBClusterOutcome outcome =
                    client.DeleteDBCluster(request);

            if (outcome.IsSuccess()) {
                std::cout << "DB cluster deletion has started."
                          << std::endl;
                clusterDeleting = true;
                std::cout
                        << "Waiting for DB cluster to delete before deleting the
parameter group."
                        << std::endl;
                std::cout << "This may take a while." << std::endl;
            }
            else {
                std::cerr << "Error with Aurora::DeleteDBCluster. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                result = false;
            }
        }
    }
    int counter = 0;

    while (clusterDeleting || instanceDeleting) {
        // 20. Wait for the DB cluster and instance to be deleted.
        std::this_thread::sleep_for(std::chrono::seconds(1));
        ++counter;
        if (counter > 800) {
            std::cerr << "Wait for instance to delete timed out ofter " <<
counter
                      << " seconds." << std::endl;
            return false;
        }
```

```
        Aws::RDS::Model::DBInstance dbInstance = Aws::RDS::Model::DBInstance();
        if (instanceDeleting) {
            if (!describeDBInstance(dbInstanceIdentifier, dbInstance, client)) {
                return false;
            }
            instanceDeleting = dbInstance.DBInstanceIdentifierHasBeenSet();
        }

        Aws::RDS::Model::DBCluster dbCluster = Aws::RDS::Model::DBCluster();
        if (clusterDeleting) {
            if (!describeDBCluster(dbClusterIdentifier, dbCluster, client)) {
                return false;
            }

            clusterDeleting = dbCluster.DBClusterIdentifierHasBeenSet();
        }

        if ((counter % 20) == 0) {
            if (instanceDeleting) {
                std::cout << "Current DB instance status is '"
                          << dbInstance.GetDBInstanceStatus() << "." <<
std::endl;
            }

            if (clusterDeleting) {
                std::cout << "Current DB cluster status is '"
                          << dbCluster.GetStatus() << "." << std::endl;
            }
        }
    }

    if (!parameterGroupName.empty()) {
        // 21. Delete the DB cluster parameter group.
        Aws::RDS::Model::DeleteDBClusterParameterGroupRequest request;
        request.SetDBClusterParameterGroupName(parameterGroupName);

        Aws::RDS::Model::DeleteDBClusterParameterGroupOutcome outcome =
                client.DeleteDBClusterParameterGroup(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB parameter group was successfully deleted."
                      << std::endl;
        }
        else {
```

```
            std::cerr << "Error with Aurora::DeleteDBClusterParameterGroup. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            result = false;
        }
    }

    return result;
}
```

- 有关 API 详细信息，请参阅《适用于 C++ 的 AWS SDK API 参考》中的以下主题。
  - CreateDBCluster
  - CreateDBClusterParameterGroup
  - CreateDBClusterSnapshot
  - CreateDBInstance
  - DeleteDBCluster
  - DeleteDBClusterParameterGroup
  - DeleteDBInstance
  - DescribeDBClusterParameterGroups
  - DescribeDBClusterParameters
  - DescribeDBClusterSnapshots
  - DescribeDBClusters
  - DescribeDBEngineVersions
  - DescribeDBInstances
  - DescribeOrderableDBInstanceOptions
  - ModifyDBClusterParameterGroup

Go

适用于 Go V2 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

在命令提示符中运行交互式场景。

```
// GetStartedClusters is an interactive example that shows you how to use the AWS
 SDK for Go
// with Amazon Aurora to do the following:
//
// 1. Create a custom DB cluster parameter group and set parameter values.
// 2. Create an Aurora DB cluster that is configured to use the parameter group.
// 3. Create a DB instance in the DB cluster that contains a database.
// 4. Take a snapshot of the DB cluster.
// 5. Delete the DB instance, DB cluster, and parameter group.
type GetStartedClusters struct {
 sdkConfig  aws.Config
 dbClusters actions.DbClusters
 questioner demotools.IQuestioner
 helper     IScenarioHelper
 isTestRun  bool
}

// NewGetStartedClusters constructs a GetStartedClusters instance from a
 configuration.
// It uses the specified config to get an Amazon Relational Database Service
 (Amazon RDS)
// client and create wrappers for the actions used in the scenario.
func NewGetStartedClusters(sdkConfig aws.Config, questioner
 demotools.IQuestioner,
 helper IScenarioHelper) GetStartedClusters {
 auroraClient := rds.NewFromConfig(sdkConfig)
 return GetStartedClusters{
  sdkConfig:  sdkConfig,
  dbClusters: actions.DbClusters{AuroraClient: auroraClient},
  questioner: questioner,
```

```
  helper:     helper,
 }
}

// Run runs the interactive scenario.
func (scenario GetStartedClusters) Run(dbEngine string, parameterGroupName
 string,
 clusterName string, dbName string) {
 defer func() {
  if r := recover(); r != nil {
   log.Println("Something went wrong with the demo.")
  }
 }()

 log.Println(strings.Repeat("-", 88))
 log.Println("Welcome to the Amazon Aurora DB Cluster demo.")
 log.Println(strings.Repeat("-", 88))

 parameterGroup := scenario.CreateParameterGroup(dbEngine, parameterGroupName)
 scenario.SetUserParameters(parameterGroupName)
 cluster := scenario.CreateCluster(clusterName, dbEngine, dbName, parameterGroup)
 scenario.helper.Pause(5)
 dbInstance := scenario.CreateInstance(cluster)
 scenario.DisplayConnection(cluster)
 scenario.CreateSnapshot(clusterName)
 scenario.Cleanup(dbInstance, cluster, parameterGroup)

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}

// CreateParameterGroup shows how to get available engine versions for a
 specified
// database engine and create a DB cluster parameter group that is compatible
 with a
// selected engine family.
func (scenario GetStartedClusters) CreateParameterGroup(dbEngine string,
 parameterGroupName string) *types.DBClusterParameterGroup {

 log.Printf("Checking for an existing DB cluster parameter group named %v.\n",
  parameterGroupName)
 parameterGroup, err := scenario.dbClusters.GetParameterGroup(parameterGroupName)
 if err != nil {
```

```
  panic(err)
 }
 if parameterGroup == nil {
  log.Printf("Getting available database engine versions for %v.\n", dbEngine)
  engineVersions, err := scenario.dbClusters.GetEngineVersions(dbEngine, "")
  if err != nil {
   panic(err)
  }

  familySet := map[string]struct{}{}
  for _, family := range engineVersions {
   familySet[*family.DBParameterGroupFamily] = struct{}{}
  }
  var families []string
  for family := range familySet {
   families = append(families, family)
  }
  sort.Strings(families)
  familyIndex := scenario.questioner.AskChoice("Which family do you want to use?
\n", families)
  log.Println("Creating a DB cluster parameter group.")
  _, err = scenario.dbClusters.CreateParameterGroup(
   parameterGroupName, families[familyIndex], "Example parameter group.")
  if err != nil {
   panic(err)
  }
  parameterGroup, err = scenario.dbClusters.GetParameterGroup(parameterGroupName)
  if err != nil {
   panic(err)
  }
 }
 log.Printf("Parameter group %v:\n", *parameterGroup.DBParameterGroupFamily)
 log.Printf("\tName: %v\n", *parameterGroup.DBClusterParameterGroupName)
 log.Printf("\tARN: %v\n", *parameterGroup.DBClusterParameterGroupArn)
 log.Printf("\tFamily: %v\n", *parameterGroup.DBParameterGroupFamily)
 log.Printf("\tDescription: %v\n", *parameterGroup.Description)
 log.Println(strings.Repeat("-", 88))
 return parameterGroup

}

// SetUserParameters shows how to get the parameters contained in a custom
 parameter
// group and update some of the parameter values in the group.
```

```
func (scenario GetStartedClusters) SetUserParameters(parameterGroupName string) {
 log.Println("Let's set some parameter values in your parameter group.")
 dbParameters, err := scenario.dbClusters.GetParameters(parameterGroupName, "")
 if err != nil {
  panic(err)
 }
 var updateParams []types.Parameter
 for _, dbParam := range dbParameters {
  if strings.HasPrefix(*dbParam.ParameterName, "auto_increment") &&
   dbParam.IsModifiable && *dbParam.DataType == "integer" {
   log.Printf("The %v parameter is described as:\n\t%v",
    *dbParam.ParameterName, *dbParam.Description)
   rangeSplit := strings.Split(*dbParam.AllowedValues, "-")
   lower, _ := strconv.Atoi(rangeSplit[0])
   upper, _ := strconv.Atoi(rangeSplit[1])
   newValue := scenario.questioner.AskInt(
    fmt.Sprintf("Enter a value between %v and %v:", lower, upper),
    demotools.InIntRange{Lower: lower, Upper: upper})
   dbParam.ParameterValue = aws.String(strconv.Itoa(newValue))
   updateParams = append(updateParams, dbParam)
  }
 }
 err = scenario.dbClusters.UpdateParameters(parameterGroupName, updateParams)
 if err != nil {
  panic(err)
 }
 log.Println("You can get a list of parameters you've set by specifying a source
 of 'user'.")
 userParameters, err := scenario.dbClusters.GetParameters(parameterGroupName,
 "user")
 if err != nil {
  panic(err)
 }
 log.Println("Here are the parameters you've set:")
 for _, param := range userParameters {
  log.Printf("\t%v: %v\n", *param.ParameterName, *param.ParameterValue)
 }
 log.Println(strings.Repeat("-", 88))
}

// CreateCluster shows how to create an Aurora DB cluster that contains a
 database
// of a specified type. The database is also configured to use a custom DB
 cluster
```

```
// parameter group.
func (scenario GetStartedClusters) CreateCluster(clusterName string, dbEngine
 string,
 dbName string, parameterGroup *types.DBClusterParameterGroup) *types.DBCluster {

 log.Println("Checking for an existing DB cluster.")
 cluster, err := scenario.dbClusters.GetDbCluster(clusterName)
 if err != nil {
  panic(err)
 }
 if cluster == nil {
  adminUsername := scenario.questioner.Ask(
   "Enter an administrator user name for the database: ", demotools.NotEmpty{})
  adminPassword := scenario.questioner.Ask(
   "Enter a password for the administrator (at least 8 characters): ",
 demotools.NotEmpty{})
  engineVersions, err := scenario.dbClusters.GetEngineVersions(dbEngine,
 *parameterGroup.DBParameterGroupFamily)
  if err != nil {
   panic(err)
  }
  var engineChoices []string
  for _, engine := range engineVersions {
   engineChoices = append(engineChoices, *engine.EngineVersion)
  }
  log.Println("The available engines for your parameter group are:")
  engineIndex := scenario.questioner.AskChoice("Which engine do you want to use?
\n", engineChoices)
  log.Printf("Creating DB cluster %v and database %v.\n", clusterName, dbName)
  log.Printf("The DB cluster is configured to use\nyour custom parameter group %v
\n",
   *parameterGroup.DBClusterParameterGroupName)
  log.Printf("and selected engine %v.\n", engineChoices[engineIndex])
  log.Println("This typically takes several minutes.")
  cluster, err = scenario.dbClusters.CreateDbCluster(
   clusterName, *parameterGroup.DBClusterParameterGroupName, dbName, dbEngine,
   engineChoices[engineIndex], adminUsername, adminPassword)
  if err != nil {
   panic(err)
  }
  for *cluster.Status != "available" {
   scenario.helper.Pause(30)
   cluster, err = scenario.dbClusters.GetDbCluster(clusterName)
   if err != nil {
```

```
    panic(err)
   }
   log.Println("Cluster created and available.")
  }
 }
 log.Println("Cluster data:")
 log.Printf("\tDBClusterIdentifier: %v\n", *cluster.DBClusterIdentifier)
 log.Printf("\tARN: %v\n", *cluster.DBClusterArn)
 log.Printf("\tStatus: %v\n", *cluster.Status)
 log.Printf("\tEngine: %v\n", *cluster.Engine)
 log.Printf("\tEngine version: %v\n", *cluster.EngineVersion)
 log.Printf("\tDBClusterParameterGroup: %v\n", *cluster.DBClusterParameterGroup)
 log.Printf("\tEngineMode: %v\n", *cluster.EngineMode)
 log.Println(strings.Repeat("-", 88))
 return cluster
}

// CreateInstance shows how to create a DB instance in an existing Aurora DB
 cluster.
// A new DB cluster contains no DB instances, so you must add one. The first DB
 instance
// that is added to a DB cluster defaults to a read-write DB instance.
func (scenario GetStartedClusters) CreateInstance(cluster *types.DBCluster)
 *types.DBInstance {
 log.Println("Checking for an existing database instance.")
 dbInstance, err := scenario.dbClusters.GetInstance(*cluster.DBClusterIdentifier)
 if err != nil {
  panic(err)
 }
 if dbInstance == nil {
  log.Println("Let's create a database instance in your DB cluster.")
  log.Println("First, choose a DB instance type:")
  instOpts, err := scenario.dbClusters.GetOrderableInstances(
   *cluster.Engine, *cluster.EngineVersion)
  if err != nil {
   panic(err)
  }
  var instChoices []string
  for _, opt := range instOpts {
   instChoices = append(instChoices, *opt.DBInstanceClass)
  }
  instIndex := scenario.questioner.AskChoice(
   "Which DB instance class do you want to use?\n", instChoices)
```

```
  log.Println("Creating a database instance. This typically takes several
 minutes.")
  dbInstance, err = scenario.dbClusters.CreateInstanceInCluster(
   *cluster.DBClusterIdentifier, *cluster.DBClusterIdentifier, *cluster.Engine,
   instChoices[instIndex])
  if err != nil {
   panic(err)
  }
  for *dbInstance.DBInstanceStatus != "available" {
   scenario.helper.Pause(30)
   dbInstance, err =
 scenario.dbClusters.GetInstance(*cluster.DBClusterIdentifier)
   if err != nil {
    panic(err)
   }
  }
 }
 log.Println("Instance data:")
 log.Printf("\tDBInstanceIdentifier: %v\n", *dbInstance.DBInstanceIdentifier)
 log.Printf("\tARN: %v\n", *dbInstance.DBInstanceArn)
 log.Printf("\tStatus: %v\n", *dbInstance.DBInstanceStatus)
 log.Printf("\tEngine: %v\n", *dbInstance.Engine)
 log.Printf("\tEngine version: %v\n", *dbInstance.EngineVersion)
 log.Println(strings.Repeat("-", 88))
 return dbInstance
}

// DisplayConnection displays connection information about an Aurora DB cluster
 and tips
// on how to connect to it.
func (scenario GetStartedClusters) DisplayConnection(cluster *types.DBCluster) {
 log.Println(
  "You can now connect to your database using your favorite MySql client.\n" +
   "One way to connect is by using the 'mysql' shell on an Amazon EC2 instance\n"
 +
   "that is running in the same VPC as your database cluster. Pass the endpoint,
\n" +
   "port, and administrator user name to 'mysql' and enter your password\n" +
   "when prompted:")
 log.Printf("\n\tmysql -h %v -P %v -u %v -p\n",
  *cluster.Endpoint, *cluster.Port, *cluster.MasterUsername)
 log.Println("For more information, see the User Guide for Aurora:\n" +
  "\thttps://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/
CHAP_GettingStartedAurora.CreatingConnecting.Aurora.html#CHAP_GettingStartedAurora.Aurora
```

```
 log.Println(strings.Repeat("-", 88))
}

// CreateSnapshot shows how to create a DB cluster snapshot and wait until it's
 available.
func (scenario GetStartedClusters) CreateSnapshot(clusterName string) {
 if scenario.questioner.AskBool(
  "Do you want to create a snapshot of your DB cluster (y/n)? ", "y") {
  snapshotId := fmt.Sprintf("%v-%v", clusterName, scenario.helper.UniqueId())
  log.Printf("Creating a snapshot named %v. This typically takes a few minutes.
\n", snapshotId)
  snapshot, err := scenario.dbClusters.CreateClusterSnapshot(clusterName,
 snapshotId)
  if err != nil {
   panic(err)
  }
  for *snapshot.Status != "available" {
   scenario.helper.Pause(30)
   snapshot, err = scenario.dbClusters.GetClusterSnapshot(snapshotId)
   if err != nil {
    panic(err)
   }
  }
  log.Println("Snapshot data:")
  log.Printf("\tDBClusterSnapshotIdentifier: %v\n",
 *snapshot.DBClusterSnapshotIdentifier)
  log.Printf("\tARN: %v\n", *snapshot.DBClusterSnapshotArn)
  log.Printf("\tStatus: %v\n", *snapshot.Status)
  log.Printf("\tEngine: %v\n", *snapshot.Engine)
  log.Printf("\tEngine version: %v\n", *snapshot.EngineVersion)
  log.Printf("\tDBClusterIdentifier: %v\n", *snapshot.DBClusterIdentifier)
  log.Printf("\tSnapshotCreateTime: %v\n", *snapshot.SnapshotCreateTime)
  log.Println(strings.Repeat("-", 88))
 }
}

// Cleanup shows how to clean up a DB instance, DB cluster, and DB cluster
 parameter group.
// Before the DB cluster parameter group can be deleted, all associated DB
 instances and
// DB clusters must first be deleted.
func (scenario GetStartedClusters) Cleanup(dbInstance *types.DBInstance, cluster
 *types.DBCluster,
 parameterGroup *types.DBClusterParameterGroup) {
```

```
 if scenario.questioner.AskBool(
  "\nDo you want to delete the database instance, DB cluster, and parameter group
 (y/n)? ", "y") {
  log.Printf("Deleting database instance %v.\n",
 *dbInstance.DBInstanceIdentifier)
  err := scenario.dbClusters.DeleteInstance(*dbInstance.DBInstanceIdentifier)
  if err != nil {
   panic(err)
  }
  log.Printf("Deleting database cluster %v.\n", *cluster.DBClusterIdentifier)
  err = scenario.dbClusters.DeleteDbCluster(*cluster.DBClusterIdentifier)
  if err != nil {
   panic(err)
  }
  log.Println(
   "Waiting for the DB instance and DB cluster to delete. This typically takes
 several minutes.")
  for dbInstance != nil || cluster != nil {
   scenario.helper.Pause(30)
   if dbInstance != nil {
    dbInstance, err =
 scenario.dbClusters.GetInstance(*dbInstance.DBInstanceIdentifier)
    if err != nil {
     panic(err)
    }
   }
   if cluster != nil {
    cluster, err = scenario.dbClusters.GetDbCluster(*cluster.DBClusterIdentifier)
    if err != nil {
     panic(err)
    }
   }
  }
  log.Printf("Deleting parameter group %v.",
 *parameterGroup.DBClusterParameterGroupName)
  err =
 scenario.dbClusters.DeleteParameterGroup(*parameterGroup.DBClusterParameterGroupName)
  if err != nil {
   panic(err)
  }
 }
}
```

定义场景调用以管理 Aurora 操作的函数。

```
type DbClusters struct {
 AuroraClient *rds.Client
}


// GetParameterGroup gets a DB cluster parameter group by name.
func (clusters *DbClusters) GetParameterGroup(parameterGroupName string) (
 *types.DBClusterParameterGroup, error) {
 output, err := clusters.AuroraClient.DescribeDBClusterParameterGroups(
  context.TODO(), &rds.DescribeDBClusterParameterGroupsInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
  })
 if err != nil {
  var notFoundError *types.DBParameterGroupNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("Parameter group %v does not exist.\n", parameterGroupName)
   err = nil
  } else {
   log.Printf("Error getting parameter group %v: %v\n", parameterGroupName, err)
  }
  return nil, err
 } else {
  return &output.DBClusterParameterGroups[0], err
 }
}


// CreateParameterGroup creates a DB cluster parameter group that is based on the
 specified
// parameter group family.
func (clusters *DbClusters) CreateParameterGroup(
 parameterGroupName string, parameterGroupFamily string, description string) (
 *types.DBClusterParameterGroup, error) {

 output, err :=
 clusters.AuroraClient.CreateDBClusterParameterGroup(context.TODO(),
  &rds.CreateDBClusterParameterGroupInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
```

```
    DBParameterGroupFamily:      aws.String(parameterGroupFamily),
    Description:                 aws.String(description),
   })
 if err != nil {
  log.Printf("Couldn't create parameter group %v: %v\n", parameterGroupName, err)
  return nil, err
 } else {
  return output.DBClusterParameterGroup, err
 }
}



// DeleteParameterGroup deletes the named DB cluster parameter group.
func (clusters *DbClusters) DeleteParameterGroup(parameterGroupName string) error
 {
 _, err := clusters.AuroraClient.DeleteDBClusterParameterGroup(context.TODO(),
  &rds.DeleteDBClusterParameterGroupInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
  })
 if err != nil {
  log.Printf("Couldn't delete parameter group %v: %v\n", parameterGroupName, err)
  return err
 } else {
  return nil
 }
}



// GetParameters gets the parameters that are contained in a DB cluster parameter
 group.
func (clusters *DbClusters) GetParameters(parameterGroupName string, source
 string) (
 []types.Parameter, error) {

 var output *rds.DescribeDBClusterParametersOutput
 var params []types.Parameter
 var err error
 parameterPaginator :=
 rds.NewDescribeDBClusterParametersPaginator(clusters.AuroraClient,
  &rds.DescribeDBClusterParametersInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
   Source:                      aws.String(source),
```

```
  })
 for parameterPaginator.HasMorePages() {
  output, err = parameterPaginator.NextPage(context.TODO())
  if err != nil {
   log.Printf("Couldn't get paramaeters for %v: %v\n", parameterGroupName, err)
   break
  } else {
   params = append(params, output.Parameters...)
  }
 }
 return params, err
}



// UpdateParameters updates parameters in a named DB cluster parameter group.
func (clusters *DbClusters) UpdateParameters(parameterGroupName string, params
 []types.Parameter) error {
 _, err := clusters.AuroraClient.ModifyDBClusterParameterGroup(context.TODO(),
  &rds.ModifyDBClusterParameterGroupInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
   Parameters:                  params,
  })
 if err != nil {
  log.Printf("Couldn't update parameters in %v: %v\n", parameterGroupName, err)
  return err
 } else {
  return nil
 }
}



// GetDbCluster gets data about an Aurora DB cluster.
func (clusters *DbClusters) GetDbCluster(clusterName string) (*types.DBCluster,
 error) {
 output, err := clusters.AuroraClient.DescribeDBClusters(context.TODO(),
  &rds.DescribeDBClustersInput{
   DBClusterIdentifier: aws.String(clusterName),
  })
 if err != nil {
  var notFoundError *types.DBClusterNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("DB cluster %v does not exist.\n", clusterName)
```

```
   err = nil
  } else {
   log.Printf("Couldn't get DB cluster %v: %v\n", clusterName, err)
  }
  return nil, err
 } else {
  return &output.DBClusters[0], err
 }
}



// CreateDbCluster creates a DB cluster that is configured to use the specified
 parameter group.
// The newly created DB cluster contains a database that uses the specified
 engine and
// engine version.
func (clusters *DbClusters) CreateDbCluster(clusterName string,
 parameterGroupName string,
 dbName string, dbEngine string, dbEngineVersion string, adminName string,
 adminPassword string) (
 *types.DBCluster, error) {

 output, err := clusters.AuroraClient.CreateDBCluster(context.TODO(),
 &rds.CreateDBClusterInput{
  DBClusterIdentifier:         aws.String(clusterName),
  Engine:                      aws.String(dbEngine),
  DBClusterParameterGroupName: aws.String(parameterGroupName),
  DatabaseName:                aws.String(dbName),
  EngineVersion:               aws.String(dbEngineVersion),
  MasterUserPassword:          aws.String(adminPassword),
  MasterUsername:              aws.String(adminName),
 })
 if err != nil {
  log.Printf("Couldn't create DB cluster %v: %v\n", clusterName, err)
  return nil, err
 } else {
  return output.DBCluster, err
 }
}



// DeleteDbCluster deletes a DB cluster without keeping a final snapshot.
```

```
func (clusters *DbClusters) DeleteDbCluster(clusterName string) error {
 _, err := clusters.AuroraClient.DeleteDBCluster(context.TODO(),
 &rds.DeleteDBClusterInput{
  DBClusterIdentifier: aws.String(clusterName),
  SkipFinalSnapshot:   true,
 })
 if err != nil {
  log.Printf("Couldn't delete DB cluster %v: %v\n", clusterName, err)
  return err
 } else {
  return nil
 }
}



// CreateClusterSnapshot creates a snapshot of a DB cluster.
func (clusters *DbClusters) CreateClusterSnapshot(clusterName string,
 snapshotName string) (
 *types.DBClusterSnapshot, error) {
 output, err := clusters.AuroraClient.CreateDBClusterSnapshot(context.TODO(),
 &rds.CreateDBClusterSnapshotInput{
  DBClusterIdentifier:         aws.String(clusterName),
  DBClusterSnapshotIdentifier: aws.String(snapshotName),
 })
 if err != nil {
  log.Printf("Couldn't create snapshot %v: %v\n", snapshotName, err)
  return nil, err
 } else {
  return output.DBClusterSnapshot, nil
 }
}



// GetClusterSnapshot gets a DB cluster snapshot.
func (clusters *DbClusters) GetClusterSnapshot(snapshotName string)
 (*types.DBClusterSnapshot, error) {
 output, err := clusters.AuroraClient.DescribeDBClusterSnapshots(context.TODO(),
  &rds.DescribeDBClusterSnapshotsInput{
   DBClusterSnapshotIdentifier: aws.String(snapshotName),
  })
 if err != nil {
  log.Printf("Couldn't get snapshot %v: %v\n", snapshotName, err)
```

```
  return nil, err
 } else {
  return &output.DBClusterSnapshots[0], nil
 }
}



// CreateInstanceInCluster creates a database instance in an existing DB cluster.
 The first database that is
// created defaults to a read-write DB instance.
func (clusters *DbClusters) CreateInstanceInCluster(clusterName string,
 instanceName string,
 dbEngine string, dbInstanceClass string) (*types.DBInstance, error) {
 output, err := clusters.AuroraClient.CreateDBInstance(context.TODO(),
 &rds.CreateDBInstanceInput{
  DBInstanceIdentifier: aws.String(instanceName),
  DBClusterIdentifier:  aws.String(clusterName),
  Engine:               aws.String(dbEngine),
  DBInstanceClass:      aws.String(dbInstanceClass),
 })
 if err != nil {
  log.Printf("Couldn't create instance %v: %v\n", instanceName, err)
  return nil, err
 } else {
  return output.DBInstance, nil
 }
}



// GetInstance gets data about a DB instance.
func (clusters *DbClusters) GetInstance(instanceName string) (
 *types.DBInstance, error) {
 output, err := clusters.AuroraClient.DescribeDBInstances(context.TODO(),
  &rds.DescribeDBInstancesInput{
   DBInstanceIdentifier: aws.String(instanceName),
  })
 if err != nil {
  var notFoundError *types.DBInstanceNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("DB instance %v does not exist.\n", instanceName)
   err = nil
  } else {
```

```
   log.Printf("Couldn't get instance %v: %v\n", instanceName, err)
  }
  return nil, err
 } else {
  return &output.DBInstances[0], nil
 }
}



// DeleteInstance deletes a DB instance.
func (clusters *DbClusters) DeleteInstance(instanceName string) error {
 _, err := clusters.AuroraClient.DeleteDBInstance(context.TODO(),
 &rds.DeleteDBInstanceInput{
  DBInstanceIdentifier:   aws.String(instanceName),
  SkipFinalSnapshot:      true,
  DeleteAutomatedBackups: aws.Bool(true),
 })
 if err != nil {
  log.Printf("Couldn't delete instance %v: %v\n", instanceName, err)
  return err
 } else {
  return nil
 }
}



// GetEngineVersions gets database engine versions that are available for the
 specified engine
// and parameter group family.
func (clusters *DbClusters) GetEngineVersions(engine string, parameterGroupFamily
 string) (
 []types.DBEngineVersion, error) {
 output, err := clusters.AuroraClient.DescribeDBEngineVersions(context.TODO(),
  &rds.DescribeDBEngineVersionsInput{
   Engine:                 aws.String(engine),
   DBParameterGroupFamily: aws.String(parameterGroupFamily),
  })
 if err != nil {
  log.Printf("Couldn't get engine versions for %v: %v\n", engine, err)
  return nil, err
 } else {
  return output.DBEngineVersions, nil
```

```
 }
}



// GetOrderableInstances uses a paginator to get DB instance options that can be
 used to create DB instances that are
// compatible with a set of specifications.
func (clusters *DbClusters) GetOrderableInstances(engine string, engineVersion
 string) (
 []types.OrderableDBInstanceOption, error) {

 var output *rds.DescribeOrderableDBInstanceOptionsOutput
 var instances []types.OrderableDBInstanceOption
 var err error
 orderablePaginator :=
 rds.NewDescribeOrderableDBInstanceOptionsPaginator(clusters.AuroraClient,
  &rds.DescribeOrderableDBInstanceOptionsInput{
   Engine:        aws.String(engine),
   EngineVersion: aws.String(engineVersion),
  })
 for orderablePaginator.HasMorePages() {
  output, err = orderablePaginator.NextPage(context.TODO())
  if err != nil {
   log.Printf("Couldn't get orderable DB instances: %v\n", err)
   break
  } else {
   instances = append(instances, output.OrderableDBInstanceOptions...)
  }
 }
 return instances, err
}
```

- 有关 API 详细信息，请参阅《适用于 Go 的 AWS SDK API 参考》中的以下主题。
  - CreateDBCluster
  - CreateDBClusterParameterGroup
  - CreateDBClusterSnapshot
  - CreateDBInstance
  - DeleteDBCluster

- DeleteDBClusterParameterGroup
- DeleteDBInstance
- DescribeDBClusterParameterGroups
- DescribeDBClusterParameters
- DescribeDBClusterSnapshots
- DescribeDBClusters
- DescribeDBEngineVersions
- DescribeDBInstances
- DescribeOrderableDBInstanceOptions
- ModifyDBClusterParameterGroup

Java

SDK for Java 2.x

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
/**
 * Before running this Java (v2) code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * This example requires an AWS Secrets Manager secret that contains the
 * database credentials. If you do not create a
 * secret, this example will not work. For details, see:
 *
 * https://docs.aws.amazon.com/secretsmanager/latest/userguide/integrating_how-
services-use-secrets_RS.html
 *
```

```
 * This Java example performs the following tasks:
 *
 * 1. Gets available engine families for Amazon Aurora MySQL-Compatible Edition
 * by calling the DescribeDbEngineVersions(Engine='aurora-mysql') method.
 * 2. Selects an engine family and creates a custom DB cluster parameter group
 * by invoking the describeDBClusterParameters method.
 * 3. Gets the parameter groups by invoking the describeDBClusterParameterGroups
 * method.
 * 4. Gets parameters in the group by invoking the describeDBClusterParameters
 * method.
 * 5. Modifies the auto_increment_offset parameter by invoking the
 * modifyDbClusterParameterGroupRequest method.
 * 6. Gets and displays the updated parameters.
 * 7. Gets a list of allowed engine versions by invoking the
 * describeDbEngineVersions method.
 * 8. Creates an Aurora DB cluster database cluster that contains a MySQL
 * database.
 * 9. Waits for DB instance to be ready.
 * 10. Gets a list of instance classes available for the selected engine.
 * 11. Creates a database instance in the cluster.
 * 12. Waits for DB instance to be ready.
 * 13. Creates a snapshot.
 * 14. Waits for DB snapshot to be ready.
 * 15. Deletes the DB cluster.
 * 16. Deletes the DB cluster group.
 */
public class AuroraScenario {
    public static long sleepTime = 20;
    public static final String DASHES = new String(new char[80]).replace("\0",
 "-");

    public static void main(String[] args) throws InterruptedException {
        final String usage = "\n" +
                "Usage:\n" +
                "    <dbClusterGroupName> <dbParameterGroupFamily>
 <dbInstanceClusterIdentifier> <dbInstanceIdentifier> <dbName>
 <dbSnapshotIdentifier><secretName>"
                +
                "Where:\n" +
                "    dbClusterGroupName - The name of the DB cluster parameter
 group. \n" +
                "    dbParameterGroupFamily - The DB cluster parameter group
 family name (for example, aurora-mysql5.7). \n"
                +
```

```
                "    dbInstanceClusterIdentifier - The instance cluster
identifier value.\n" +
                "    dbInstanceIdentifier - The database instance identifier.\n"
+
                "    dbName - The database name.\n" +
                "    dbSnapshotIdentifier - The snapshot identifier.\n" +
                "    secretName - The name of the AWS Secrets Manager secret that
contains the database credentials\"\n";
        ;

        if (args.length != 7) {
            System.out.println(usage);
            System.exit(1);
        }

        String dbClusterGroupName = args[0];
        String dbParameterGroupFamily = args[1];
        String dbInstanceClusterIdentifier = args[2];
        String dbInstanceIdentifier = args[3];
        String dbName = args[4];
        String dbSnapshotIdentifier = args[5];
        String secretName = args[6];

        // Retrieve the database credentials using AWS Secrets Manager.
        Gson gson = new Gson();
        User user = gson.fromJson(String.valueOf(getSecretValues(secretName)),
User.class);
        String username = user.getUsername();
        String userPassword = user.getPassword();

        Region region = Region.US_WEST_2;
        RdsClient rdsClient = RdsClient.builder()
                .region(region)
                .build();

        System.out.println(DASHES);
        System.out.println("Welcome to the Amazon Aurora example scenario.");
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("1. Return a list of the available DB engines");
        describeDBEngines(rdsClient);
        System.out.println(DASHES);
```

```
        System.out.println(DASHES);
        System.out.println("2. Create a custom parameter group");
        createDBClusterParameterGroup(rdsClient, dbClusterGroupName,
dbParameterGroupFamily);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("3. Get the parameter group");
        describeDbClusterParameterGroups(rdsClient, dbClusterGroupName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("4. Get the parameters in the group");
        describeDbClusterParameters(rdsClient, dbClusterGroupName, 0);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("5. Modify the auto_increment_offset parameter");
        modifyDBClusterParas(rdsClient, dbClusterGroupName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("6. Display the updated parameter value");
        describeDbClusterParameters(rdsClient, dbClusterGroupName, -1);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("7. Get a list of allowed engine versions");
        getAllowedEngines(rdsClient, dbParameterGroupFamily);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("8. Create an Aurora DB cluster database");
        String arnClusterVal = createDBCluster(rdsClient, dbClusterGroupName,
dbName, dbInstanceClusterIdentifier,
                username, userPassword);
        System.out.println("The ARN of the cluster is " + arnClusterVal);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("9. Wait for DB instance to be ready");
        waitForInstanceReady(rdsClient, dbInstanceClusterIdentifier);
        System.out.println(DASHES);
```

```
        System.out.println(DASHES);
        System.out.println("10. Get a list of instance classes available for the
selected engine");
        String instanceClass = getListInstanceClasses(rdsClient);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("11. Create a database instance in the cluster.");
        String clusterDBARN = createDBInstanceCluster(rdsClient,
dbInstanceIdentifier, dbInstanceClusterIdentifier,
                instanceClass);
        System.out.println("The ARN of the database is " + clusterDBARN);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("12. Wait for DB instance to be ready");
        waitDBInstanceReady(rdsClient, dbInstanceIdentifier);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("13. Create a snapshot");
        createDBClusterSnapshot(rdsClient, dbInstanceClusterIdentifier,
dbSnapshotIdentifier);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("14. Wait for DB snapshot to be ready");
        waitForSnapshotReady(rdsClient, dbSnapshotIdentifier,
dbInstanceClusterIdentifier);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("14. Delete the DB instance");
        deleteDatabaseInstance(rdsClient, dbInstanceIdentifier);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("15. Delete the DB cluster");
        deleteCluster(rdsClient, dbInstanceClusterIdentifier);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("16. Delete the DB cluster group");
        deleteDBClusterGroup(rdsClient, dbClusterGroupName, clusterDBARN);
```

```
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("The Scenario has successfully completed.");
        System.out.println(DASHES);
        rdsClient.close();
    }

    private static SecretsManagerClient getSecretClient() {
        Region region = Region.US_WEST_2;
        return SecretsManagerClient.builder()
                .region(region)

.credentialsProvider(EnvironmentVariableCredentialsProvider.create())
                .build();
    }

    private static String getSecretValues(String secretName) {
        SecretsManagerClient secretClient = getSecretClient();
        GetSecretValueRequest valueRequest = GetSecretValueRequest.builder()
                .secretId(secretName)
                .build();

        GetSecretValueResponse valueResponse =
secretClient.getSecretValue(valueRequest);
        return valueResponse.secretString();
    }

    public static void deleteDBClusterGroup(RdsClient rdsClient, String
dbClusterGroupName, String clusterDBARN)
            throws InterruptedException {
        try {
            boolean isDataDel = false;
            boolean didFind;
            String instanceARN;

            // Make sure that the database has been deleted.
            while (!isDataDel) {
                DescribeDbInstancesResponse response =
rdsClient.describeDBInstances();
                List<DBInstance> instanceList = response.dbInstances();
                int listSize = instanceList.size();
                didFind = false;
                int index = 1;
```

```
                for (DBInstance instance : instanceList) {
                    instanceARN = instance.dbInstanceArn();
                    if (instanceARN.compareTo(clusterDBARN) == 0) {
                        System.out.println(clusterDBARN + " still exists");
                        didFind = true;
                    }
                    if ((index == listSize) && (!didFind)) {
                        // Went through the entire list and did not find the
database ARN.
                        isDataDel = true;
                    }
                    Thread.sleep(sleepTime * 1000);
                    index++;
                }
            }

            DeleteDbClusterParameterGroupRequest clusterParameterGroupRequest =
DeleteDbClusterParameterGroupRequest
                    .builder()
                    .dbClusterParameterGroupName(dbClusterGroupName)
                    .build();


rdsClient.deleteDBClusterParameterGroup(clusterParameterGroupRequest);
            System.out.println(dbClusterGroupName + " was deleted.");

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    public static void deleteCluster(RdsClient rdsClient, String
dbInstanceClusterIdentifier) {
        try {
            DeleteDbClusterRequest deleteDbClusterRequest =
DeleteDbClusterRequest.builder()
                    .dbClusterIdentifier(dbInstanceClusterIdentifier)
                    .skipFinalSnapshot(true)
                    .build();

            rdsClient.deleteDBCluster(deleteDbClusterRequest);
            System.out.println(dbInstanceClusterIdentifier + " was deleted!");
```

```
        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    public static void deleteDatabaseInstance(RdsClient rdsClient, String
dbInstanceIdentifier) {
        try {
            DeleteDbInstanceRequest deleteDbInstanceRequest =
DeleteDbInstanceRequest.builder()
                    .dbInstanceIdentifier(dbInstanceIdentifier)
                    .deleteAutomatedBackups(true)
                    .skipFinalSnapshot(true)
                    .build();

            DeleteDbInstanceResponse response =
rdsClient.deleteDBInstance(deleteDbInstanceRequest);
            System.out.println("The status of the database is " +
response.dbInstance().dbInstanceStatus());

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    public static void waitForSnapshotReady(RdsClient rdsClient, String
dbSnapshotIdentifier,
            String dbInstanceClusterIdentifier) {
        try {
            boolean snapshotReady = false;
            String snapshotReadyStr;
            System.out.println("Waiting for the snapshot to become available.");

            DescribeDbClusterSnapshotsRequest snapshotsRequest =
DescribeDbClusterSnapshotsRequest.builder()
                    .dbClusterSnapshotIdentifier(dbSnapshotIdentifier)
                    .dbClusterIdentifier(dbInstanceClusterIdentifier)
                    .build();

            while (!snapshotReady) {
                DescribeDbClusterSnapshotsResponse response =
rdsClient.describeDBClusterSnapshots(snapshotsRequest);
```

```
                List<DBClusterSnapshot> snapshotList =
response.dbClusterSnapshots();
                for (DBClusterSnapshot snapshot : snapshotList) {
                    snapshotReadyStr = snapshot.status();
                    if (snapshotReadyStr.contains("available")) {
                        snapshotReady = true;
                    } else {
                        System.out.println(".");
                        Thread.sleep(sleepTime * 5000);
                    }
                }
            }

            System.out.println("The Snapshot is available!");

        } catch (RdsException | InterruptedException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    public static void createDBClusterSnapshot(RdsClient rdsClient, String
dbInstanceClusterIdentifier,
            String dbSnapshotIdentifier) {
        try {
            CreateDbClusterSnapshotRequest snapshotRequest =
CreateDbClusterSnapshotRequest.builder()
                    .dbClusterIdentifier(dbInstanceClusterIdentifier)
                    .dbClusterSnapshotIdentifier(dbSnapshotIdentifier)
                    .build();

            CreateDbClusterSnapshotResponse response =
rdsClient.createDBClusterSnapshot(snapshotRequest);
            System.out.println("The Snapshot ARN is " +
response.dbClusterSnapshot().dbClusterSnapshotArn());

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    public static void waitDBInstanceReady(RdsClient rdsClient, String
dbInstanceIdentifier) {
```

```
        boolean instanceReady = false;
        String instanceReadyStr;
        System.out.println("Waiting for instance to become available.");
        try {
            DescribeDbInstancesRequest instanceRequest =
DescribeDbInstancesRequest.builder()
                    .dbInstanceIdentifier(dbInstanceIdentifier)
                    .build();

            String endpoint = "";
            while (!instanceReady) {
                DescribeDbInstancesResponse response =
rdsClient.describeDBInstances(instanceRequest);
                List<DBInstance> instanceList = response.dbInstances();
                for (DBInstance instance : instanceList) {
                    instanceReadyStr = instance.dbInstanceStatus();
                    if (instanceReadyStr.contains("available")) {
                        endpoint = instance.endpoint().address();
                        instanceReady = true;
                    } else {
                        System.out.print(".");
                        Thread.sleep(sleepTime * 1000);
                    }
                }
            }
            System.out.println("Database instance is available! The connection
endpoint is " + endpoint);

        } catch (RdsException | InterruptedException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static String createDBInstanceCluster(RdsClient rdsClient,
            String dbInstanceIdentifier,
            String dbInstanceClusterIdentifier,
            String instanceClass) {
        try {
            CreateDbInstanceRequest instanceRequest =
CreateDbInstanceRequest.builder()
                    .dbInstanceIdentifier(dbInstanceIdentifier)
                    .dbClusterIdentifier(dbInstanceClusterIdentifier)
                    .engine("aurora-mysql")
```

```
                    .dbInstanceClass(instanceClass)
                    .build();

            CreateDbInstanceResponse response =
rdsClient.createDBInstance(instanceRequest);
            System.out.print("The status is " +
response.dbInstance().dbInstanceStatus());
            return response.dbInstance().dbInstanceArn();

        } catch (RdsException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        return "";
    }

    public static String getListInstanceClasses(RdsClient rdsClient) {
        try {
            DescribeOrderableDbInstanceOptionsRequest optionsRequest =
DescribeOrderableDbInstanceOptionsRequest
                    .builder()
                    .engine("aurora-mysql")
                    .maxRecords(20)
                    .build();

            DescribeOrderableDbInstanceOptionsResponse response = rdsClient
                    .describeOrderableDBInstanceOptions(optionsRequest);
            List<OrderableDBInstanceOption> instanceOptions =
response.orderableDBInstanceOptions();
            String instanceClass = "";
            for (OrderableDBInstanceOption instanceOption : instanceOptions) {
                instanceClass = instanceOption.dbInstanceClass();
                System.out.println("The instance class is " +
instanceOption.dbInstanceClass());
                System.out.println("The engine version is " +
instanceOption.engineVersion());
            }
            return instanceClass;

        } catch (RdsException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        return "";
```

```
    }

    // Waits until the database instance is available.
    public static void waitForInstanceReady(RdsClient rdsClient, String
dbClusterIdentifier) {
        boolean instanceReady = false;
        String instanceReadyStr;
        System.out.println("Waiting for instance to become available.");
        try {
            DescribeDbClustersRequest instanceRequest =
DescribeDbClustersRequest.builder()
                    .dbClusterIdentifier(dbClusterIdentifier)
                    .build();

            while (!instanceReady) {
                DescribeDbClustersResponse response =
rdsClient.describeDBClusters(instanceRequest);
                List<DBCluster> clusterList = response.dbClusters();
                for (DBCluster cluster : clusterList) {
                    instanceReadyStr = cluster.status();
                    if (instanceReadyStr.contains("available")) {
                        instanceReady = true;
                    } else {
                        System.out.print(".");
                        Thread.sleep(sleepTime * 1000);
                    }
                }
            }
            System.out.println("Database cluster is available!");

        } catch (RdsException | InterruptedException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static String createDBCluster(RdsClient rdsClient, String
dbParameterGroupFamily, String dbName,
            String dbClusterIdentifier, String userName, String password) {
        try {
            CreateDbClusterRequest clusterRequest =
CreateDbClusterRequest.builder()
                    .databaseName(dbName)
                    .dbClusterIdentifier(dbClusterIdentifier)
```

```
                    .dbClusterParameterGroupName(dbParameterGroupFamily)
                    .engine("aurora-mysql")
                    .masterUsername(userName)
                    .masterUserPassword(password)
                    .build();

            CreateDbClusterResponse response =
rdsClient.createDBCluster(clusterRequest);
            return response.dbCluster().dbClusterArn();

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
        return "";
    }

    // Get a list of allowed engine versions.
    public static void getAllowedEngines(RdsClient rdsClient, String
dbParameterGroupFamily) {
        try {
            DescribeDbEngineVersionsRequest versionsRequest =
DescribeDbEngineVersionsRequest.builder()
                    .dbParameterGroupFamily(dbParameterGroupFamily)
                    .engine("aurora-mysql")
                    .build();

            DescribeDbEngineVersionsResponse response =
rdsClient.describeDBEngineVersions(versionsRequest);
            List<DBEngineVersion> dbEngines = response.dbEngineVersions();
            for (DBEngineVersion dbEngine : dbEngines) {
                System.out.println("The engine version is " +
dbEngine.engineVersion());
                System.out.println("The engine description is " +
dbEngine.dbEngineDescription());
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    // Modify the auto_increment_offset parameter.
```

```
   public static void modifyDBClusterParas(RdsClient rdsClient, String
dClusterGroupName) {
       try {
           Parameter parameter1 = Parameter.builder()
                   .parameterName("auto_increment_offset")
                   .applyMethod("immediate")
                   .parameterValue("5")
                   .build();

           List<Parameter> paraList = new ArrayList<>();
           paraList.add(parameter1);
           ModifyDbClusterParameterGroupRequest groupRequest =
ModifyDbClusterParameterGroupRequest.builder()
                   .dbClusterParameterGroupName(dClusterGroupName)
                   .parameters(paraList)
                   .build();

           ModifyDbClusterParameterGroupResponse response =
rdsClient.modifyDBClusterParameterGroup(groupRequest);
           System.out.println(
                   "The parameter group " +
response.dbClusterParameterGroupName() + " was successfully modified");

       } catch (RdsException e) {
           System.out.println(e.getLocalizedMessage());
           System.exit(1);
       }
   }

   public static void describeDbClusterParameters(RdsClient rdsClient, String
dbCLusterGroupName, int flag) {
       try {
           DescribeDbClusterParametersRequest dbParameterGroupsRequest;
           if (flag == 0) {
               dbParameterGroupsRequest =
DescribeDbClusterParametersRequest.builder()
                       .dbClusterParameterGroupName(dbCLusterGroupName)
                       .build();
           } else {
               dbParameterGroupsRequest =
DescribeDbClusterParametersRequest.builder()
                       .dbClusterParameterGroupName(dbCLusterGroupName)
                       .source("user")
                       .build();
```

```
            }

            DescribeDbClusterParametersResponse response = rdsClient
                    .describeDBClusterParameters(dbParameterGroupsRequest);
            List<Parameter> dbParameters = response.parameters();
            String paraName;
            for (Parameter para : dbParameters) {
                // Only print out information about either auto_increment_offset
or
                // auto_increment_increment.
                paraName = para.parameterName();
                if ((paraName.compareTo("auto_increment_offset") == 0)
                        || (paraName.compareTo("auto_increment_increment ") ==
0)) {
                    System.out.println("*** The parameter name is  " + paraName);
                    System.out.println("*** The parameter value is  " +
para.parameterValue());
                    System.out.println("*** The parameter data type is " +
para.dataType());
                    System.out.println("*** The parameter description is " +
para.description());
                    System.out.println("*** The parameter allowed values  is " +
para.allowedValues());
                }
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    public static void describeDbClusterParameterGroups(RdsClient rdsClient,
String dbClusterGroupName) {
        try {
            DescribeDbClusterParameterGroupsRequest groupsRequest =
DescribeDbClusterParameterGroupsRequest.builder()
                    .dbClusterParameterGroupName(dbClusterGroupName)
                    .maxRecords(20)
                    .build();

            List<DBClusterParameterGroup> groups =
rdsClient.describeDBClusterParameterGroups(groupsRequest)
                    .dbClusterParameterGroups();
```

```
            for (DBClusterParameterGroup group : groups) {
                System.out.println("The group name is " +
group.dbClusterParameterGroupName());
                System.out.println("The group ARN is " +
group.dbClusterParameterGroupArn());
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    public static void createDBClusterParameterGroup(RdsClient rdsClient, String
dbClusterGroupName,
            String dbParameterGroupFamily) {
        try {
            CreateDbClusterParameterGroupRequest groupRequest =
CreateDbClusterParameterGroupRequest.builder()
                    .dbClusterParameterGroupName(dbClusterGroupName)
                    .dbParameterGroupFamily(dbParameterGroupFamily)
                    .description("Created by using the AWS SDK for Java")
                    .build();

            CreateDbClusterParameterGroupResponse response =
rdsClient.createDBClusterParameterGroup(groupRequest);
            System.out.println("The group name is " +
response.dbClusterParameterGroup().dbClusterParameterGroupName());

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    public static void describeDBEngines(RdsClient rdsClient) {
        try {
            DescribeDbEngineVersionsRequest engineVersionsRequest =
DescribeDbEngineVersionsRequest.builder()
                    .engine("aurora-mysql")
                    .defaultOnly(true)
                    .maxRecords(20)
                    .build();
```

```
            DescribeDbEngineVersionsResponse response =
 rdsClient.describeDBEngineVersions(engineVersionsRequest);
            List<DBEngineVersion> engines = response.dbEngineVersions();

            // Get all DBEngineVersion objects.
            for (DBEngineVersion engineOb : engines) {
                System.out.println("The name of the DB parameter group family for
 the database engine is "
                        + engineOb.dbParameterGroupFamily());
                System.out.println("The name of the database engine " +
 engineOb.engine());
                System.out.println("The version number of the database engine " +
 engineOb.engineVersion());
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Java 2.x API 参考》中的以下主题。
  - CreateDBCluster
  - CreateDBClusterParameterGroup
  - CreateDBClusterSnapshot
  - CreateDBInstance
  - DeleteDBCluster
  - DeleteDBClusterParameterGroup
  - DeleteDBInstance
  - DescribeDBClusterParameterGroups
  - DescribeDBClusterParameters
  - DescribeDBClusterSnapshots
  - DescribeDBClusters
  - DescribeDBEngineVersions
  - DescribeDBInstances

- DescribeOrderableDBInstanceOptions
- ModifyDBClusterParameterGroup

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
/**
Before running this Kotlin code example, set up your development environment,
 including your credentials.

For more information, see the following documentation topic:

https://docs.aws.amazon.com/sdk-for-kotlin/latest/developer-guide/setup.html

This example requires an AWS Secrets Manager secret that contains the database
 credentials. If you do not create a
secret, this example will not work. For more details, see:

https://docs.aws.amazon.com/secretsmanager/latest/userguide/integrating_how-
services-use-secrets_RS.html

This Kotlin example performs the following tasks:

1. Returns a list of the available DB engines.
2. Creates a custom DB parameter group.
3. Gets the parameter groups.
4. Gets the parameters in the group.
5. Modifies the auto_increment_increment parameter.
6. Displays the updated parameter value.
7. Gets a list of allowed engine versions.
8. Creates an Aurora DB cluster database.
9. Waits for DB instance to be ready.
10. Gets a list of instance classes available for the selected engine.
```

```
11. Creates a database instance in the cluster.
12. Waits for the database instance in the cluster to be ready.
13. Creates a snapshot.
14. Waits for DB snapshot to be ready.
15. Deletes the DB instance.
16. Deletes the DB cluster.
17. Deletes the DB cluster group.
 */

var slTime: Long = 20

suspend fun main(args: Array<String>) {
    val usage = """
        Usage:
            <dbClusterGroupName> <dbParameterGroupFamily>
 <dbInstanceClusterIdentifier> <dbName> <dbSnapshotIdentifier> <secretName>
        Where:
            dbClusterGroupName - The database group name.
            dbParameterGroupFamily - The database parameter group name.
            dbInstanceClusterIdentifier - The database instance identifier.
            dbName -  The database name.
            dbSnapshotIdentifier - The snapshot identifier.
            secretName - The name of the AWS Secrets Manager secret that contains
 the database credentials.
    """

    if (args.size != 7) {
        println(usage)
        exitProcess(1)
    }

    val dbClusterGroupName = args[0]
    val dbParameterGroupFamily = args[1]
    val dbInstanceClusterIdentifier = args[2]
    val dbInstanceIdentifier = args[3]
    val dbName = args[4]
    val dbSnapshotIdentifier = args[5]
    val secretName = args[6]

    val gson = Gson()
    val user = gson.fromJson(getSecretValues(secretName).toString(),
 User::class.java)
    val username = user.username
    val userPassword = user.password
```

```
    println("1. Return a list of the available DB engines")
    describeAuroraDBEngines()

    println("2. Create a custom parameter group")
    createDBClusterParameterGroup(dbClusterGroupName, dbParameterGroupFamily)

    println("3. Get the parameter group")
    describeDbClusterParameterGroups(dbClusterGroupName)

    println("4. Get the parameters in the group")
    describeDbClusterParameters(dbClusterGroupName, 0)

    println("5. Modify the auto_increment_offset parameter")
    modifyDBClusterParas(dbClusterGroupName)

    println("6. Display the updated parameter value")
    describeDbClusterParameters(dbClusterGroupName, -1)

    println("7. Get a list of allowed engine versions")
    getAllowedClusterEngines(dbParameterGroupFamily)

    println("8. Create an Aurora DB cluster database")
    val arnClusterVal = createDBCluster(dbClusterGroupName, dbName,
dbInstanceClusterIdentifier, username, userPassword)
    println("The ARN of the cluster is $arnClusterVal")

    println("9. Wait for DB instance to be ready")
    waitForClusterInstanceReady(dbInstanceClusterIdentifier)

    println("10. Get a list of instance classes available for the selected
engine")
    val instanceClass = getListInstanceClasses()

    println("11. Create a database instance in the cluster.")
    val clusterDBARN = createDBInstanceCluster(dbInstanceIdentifier,
dbInstanceClusterIdentifier, instanceClass)
    println("The ARN of the database is $clusterDBARN")

    println("12. Wait for DB instance to be ready")
    waitDBAuroraInstanceReady(dbInstanceIdentifier)

    println("13. Create a snapshot")
    createDBClusterSnapshot(dbInstanceClusterIdentifier, dbSnapshotIdentifier)
```

```
    println("14. Wait for DB snapshot to be ready")
    waitSnapshotReady(dbSnapshotIdentifier, dbInstanceClusterIdentifier)

    println("15. Delete the DB instance")
    deleteDBInstance(dbInstanceIdentifier)

    println("16. Delete the DB cluster")
    deleteCluster(dbInstanceClusterIdentifier)

    println("17. Delete the DB cluster group")
    if (clusterDBARN != null) {
        deleteDBClusterGroup(dbClusterGroupName, clusterDBARN)
    }
    println("The Scenario has successfully completed.")
}

@Throws(InterruptedException::class)
suspend fun deleteDBClusterGroup(
    dbClusterGroupName: String,
    clusterDBARN: String,
) {
    var isDataDel = false
    var didFind: Boolean
    var instanceARN: String

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        // Make sure that the database has been deleted.
        while (!isDataDel) {
            val response = rdsClient.describeDbInstances()
            val instanceList = response.dbInstances
            val listSize = instanceList?.size
            isDataDel = false
            didFind = false
            var index = 1
            if (instanceList != null) {
                for (instance in instanceList) {
                    instanceARN = instance.dbInstanceArn.toString()
                    if (instanceARN.compareTo(clusterDBARN) == 0) {
                        println("$clusterDBARN still exists")
                        didFind = true
                    }
                    if (index == listSize && !didFind) {
```

```
                        // Went through the entire list and did not find the
 database ARN.
                        isDataDel = true
                    }
                    delay(slTime * 1000)
                    index++
                }
            }
        }
        val clusterParameterGroupRequest =
            DeleteDbClusterParameterGroupRequest {
                dbClusterParameterGroupName = dbClusterGroupName
            }

        rdsClient.deleteDbClusterParameterGroup(clusterParameterGroupRequest)
        println("$dbClusterGroupName was deleted.")
    }
}

suspend fun deleteCluster(dbInstanceClusterIdentifier: String) {
    val deleteDbClusterRequest =
        DeleteDbClusterRequest {
            dbClusterIdentifier = dbInstanceClusterIdentifier
            skipFinalSnapshot = true
        }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        rdsClient.deleteDbCluster(deleteDbClusterRequest)
        println("$dbInstanceClusterIdentifier was deleted!")
    }
}

suspend fun deleteDBInstance(dbInstanceIdentifierVal: String) {
    val deleteDbInstanceRequest =
        DeleteDbInstanceRequest {
            dbInstanceIdentifier = dbInstanceIdentifierVal
            deleteAutomatedBackups = true
            skipFinalSnapshot = true
        }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.deleteDbInstance(deleteDbInstanceRequest)
        print("The status of the database is
 ${response.dbInstance?.dbInstanceStatus}")
```

```
    }
}

suspend fun waitSnapshotReady(
    dbSnapshotIdentifier: String?,
    dbInstanceClusterIdentifier: String?,
) {
    var snapshotReady = false
    var snapshotReadyStr: String
    println("Waiting for the snapshot to become available.")

    val snapshotsRequest =
        DescribeDbClusterSnapshotsRequest {
            dbClusterSnapshotIdentifier = dbSnapshotIdentifier
            dbClusterIdentifier = dbInstanceClusterIdentifier
        }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        while (!snapshotReady) {
            val response = rdsClient.describeDbClusterSnapshots(snapshotsRequest)
            val snapshotList = response.dbClusterSnapshots
            if (snapshotList != null) {
                for (snapshot in snapshotList) {
                    snapshotReadyStr = snapshot.status.toString()
                    if (snapshotReadyStr.contains("available")) {
                        snapshotReady = true
                    } else {
                        println(".")
                        delay(slTime * 5000)
                    }
                }
            }
        }
    }
    println("The Snapshot is available!")
}

suspend fun createDBClusterSnapshot(
    dbInstanceClusterIdentifier: String?,
    dbSnapshotIdentifier: String?,
) {
    val snapshotRequest =
        CreateDbClusterSnapshotRequest {
            dbClusterIdentifier = dbInstanceClusterIdentifier
```

```
            dbClusterSnapshotIdentifier = dbSnapshotIdentifier
        }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbClusterSnapshot(snapshotRequest)
        println("The Snapshot ARN is
 ${response.dbClusterSnapshot?.dbClusterSnapshotArn}")
    }
}

suspend fun waitDBAuroraInstanceReady(dbInstanceIdentifierVal: String?) {
    var instanceReady = false
    var instanceReadyStr: String
    println("Waiting for instance to become available.")
    val instanceRequest =
        DescribeDbInstancesRequest {
            dbInstanceIdentifier = dbInstanceIdentifierVal
        }

    var endpoint = ""
    RdsClient { region = "us-west-2" }.use { rdsClient ->
        while (!instanceReady) {
            val response = rdsClient.describeDbInstances(instanceRequest)
            response.dbInstances?.forEach { instance ->
                instanceReadyStr = instance.dbInstanceStatus.toString()
                if (instanceReadyStr.contains("available")) {
                    endpoint = instance.endpoint?.address.toString()
                    instanceReady = true
                } else {
                    print(".")
                    delay(sleepTime * 1000)
                }
            }
        }
    }
    println("Database instance is available! The connection endpoint is
 $endpoint")
}

suspend fun createDBInstanceCluster(
    dbInstanceIdentifierVal: String?,
    dbInstanceClusterIdentifierVal: String?,
    instanceClassVal: String?,
): String? {
```

```
    val instanceRequest =
        CreateDbInstanceRequest {
            dbInstanceIdentifier = dbInstanceIdentifierVal
            dbClusterIdentifier = dbInstanceClusterIdentifierVal
            engine = "aurora-mysql"
            dbInstanceClass = instanceClassVal
        }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbInstance(instanceRequest)
        print("The status is ${response.dbInstance?.dbInstanceStatus}")
        return response.dbInstance?.dbInstanceArn
    }
}

suspend fun getListInstanceClasses(): String {
    val optionsRequest =
        DescribeOrderableDbInstanceOptionsRequest {
            engine = "aurora-mysql"
            maxRecords = 20
        }
    var instanceClass = ""
    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response =
 rdsClient.describeOrderableDbInstanceOptions(optionsRequest)
        response.orderableDbInstanceOptions?.forEach { instanceOption ->
            instanceClass = instanceOption.dbInstanceClass.toString()
            println("The instance class is ${instanceOption.dbInstanceClass}")
            println("The engine version is ${instanceOption.engineVersion}")
        }
    }
    return instanceClass
}

// Waits until the database instance is available.
suspend fun waitForClusterInstanceReady(dbClusterIdentifierVal: String?) {
    var instanceReady = false
    var instanceReadyStr: String
    println("Waiting for instance to become available.")

    val instanceRequest =
        DescribeDbClustersRequest {
            dbClusterIdentifier = dbClusterIdentifierVal
        }
```

```
    RdsClient { region = "us-west-2" }.use { rdsClient ->
        while (!instanceReady) {
            val response = rdsClient.describeDbClusters(instanceRequest)
            response.dbClusters?.forEach { cluster ->
                instanceReadyStr = cluster.status.toString()
                if (instanceReadyStr.contains("available")) {
                    instanceReady = true
                } else {
                    print(".")
                    delay(sleepTime * 1000)
                }
            }
        }
    }
    println("Database cluster is available!")
}

suspend fun createDBCluster(
    dbParameterGroupFamilyVal: String?,
    dbName: String?,
    dbClusterIdentifierVal: String?,
    userName: String?,
    password: String?,
): String? {
    val clusterRequest =
        CreateDbClusterRequest {
            databaseName = dbName
            dbClusterIdentifier = dbClusterIdentifierVal
            dbClusterParameterGroupName = dbParameterGroupFamilyVal
            engine = "aurora-mysql"
            masterUsername = userName
            masterUserPassword = password
        }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbCluster(clusterRequest)
        return response.dbCluster?.dbClusterArn
    }
}

// Get a list of allowed engine versions.
suspend fun getAllowedClusterEngines(dbParameterGroupFamilyVal: String?) {
    val versionsRequest =
```

```
        DescribeDbEngineVersionsRequest {
            dbParameterGroupFamily = dbParameterGroupFamilyVal
            engine = "aurora-mysql"
        }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.describeDbEngineVersions(versionsRequest)
        response.dbEngineVersions?.forEach { dbEngine ->
            println("The engine version is ${dbEngine.engineVersion}")
            println("The engine description is ${dbEngine.dbEngineDescription}")
        }
    }
}

// Modify the auto_increment_offset parameter.
suspend fun modifyDBClusterParas(dClusterGroupName: String?) {
    val parameter1 =
        Parameter {
            parameterName = "auto_increment_offset"
            applyMethod = ApplyMethod.fromValue("immediate")
            parameterValue = "5"
        }

    val paraList = ArrayList<Parameter>()
    paraList.add(parameter1)
    val groupRequest =
        ModifyDbClusterParameterGroupRequest {
            dbClusterParameterGroupName = dClusterGroupName
            parameters = paraList
        }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.modifyDbClusterParameterGroup(groupRequest)
        println("The parameter group ${response.dbClusterParameterGroupName} was
 successfully modified")
    }
}

suspend fun describeDbClusterParameters(
    dbCLusterGroupName: String?,
    flag: Int,
) {
    val dbParameterGroupsRequest: DescribeDbClusterParametersRequest
    dbParameterGroupsRequest =
```

```
        if (flag == 0) {
            DescribeDbClusterParametersRequest {
                dbClusterParameterGroupName = dbCLusterGroupName
            }
        } else {
            DescribeDbClusterParametersRequest {
                dbClusterParameterGroupName = dbCLusterGroupName
                source = "user"
            }
        }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response =
 rdsClient.describeDbClusterParameters(dbParameterGroupsRequest)
        response.parameters?.forEach { para ->
            // Only print out information about either auto_increment_offset or
 auto_increment_increment.
            val paraName = para.parameterName
            if (paraName != null) {
                if (paraName.compareTo("auto_increment_offset") == 0 ||
 paraName.compareTo("auto_increment_increment ") == 0) {
                    println("*** The parameter name is  $paraName")
                    println("*** The parameter value is  ${para.parameterValue}")
                    println("*** The parameter data type is ${para.dataType}")
                    println("*** The parameter description is
 ${para.description}")
                    println("*** The parameter allowed values  is
 ${para.allowedValues}")
                }
            }
        }
    }
}

suspend fun describeDbClusterParameterGroups(dbClusterGroupName: String?) {
    val groupsRequest =
        DescribeDbClusterParameterGroupsRequest {
            dbClusterParameterGroupName = dbClusterGroupName
            maxRecords = 20
        }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.describeDbClusterParameterGroups(groupsRequest)
        response.dbClusterParameterGroups?.forEach { group ->
```

```
            println("The group name is ${group.dbClusterParameterGroupName}")
            println("The group ARN is ${group.dbClusterParameterGroupArn}")
        }
    }
}

suspend fun createDBClusterParameterGroup(
    dbClusterGroupNameVal: String?,
    dbParameterGroupFamilyVal: String?,
) {
    val groupRequest =
        CreateDbClusterParameterGroupRequest {
            dbClusterParameterGroupName = dbClusterGroupNameVal
            dbParameterGroupFamily = dbParameterGroupFamilyVal
            description = "Created by using the AWS SDK for Kotlin"
        }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbClusterParameterGroup(groupRequest)
        println("The group name is
 ${response.dbClusterParameterGroup?.dbClusterParameterGroupName}")
    }
}

suspend fun describeAuroraDBEngines() {
    val engineVersionsRequest =
        DescribeDbEngineVersionsRequest {
            engine = "aurora-mysql"
            defaultOnly = true
            maxRecords = 20
        }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.describeDbEngineVersions(engineVersionsRequest)
        response.dbEngineVersions?.forEach { engineOb ->
            println("The name of the DB parameter group family for the database
 engine is ${engineOb.dbParameterGroupFamily}")
            println("The name of the database engine ${engineOb.engine}")
            println("The version number of the database engine
 ${engineOb.engineVersion}")
        }
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Kotlin API 参考》中的以下主题。
  - CreateDBCluster
  - CreateDBClusterParameterGroup
  - CreateDBClusterSnapshot
  - CreateDBInstance
  - DeleteDBCluster
  - DeleteDBClusterParameterGroup
  - DeleteDBInstance
  - DescribeDBClusterParameterGroups
  - DescribeDBClusterParameters
  - DescribeDBClusterSnapshots
  - DescribeDBClusters
  - DescribeDBEngineVersions
  - DescribeDBInstances
  - DescribeOrderableDBInstanceOptions
  - ModifyDBClusterParameterGroup

Python

SDK for Python (Boto3)

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

在命令提示符中运行交互式场景。

```
class AuroraClusterScenario:
    """Runs a scenario that shows how to get started using Aurora DB clusters."""

    def __init__(self, aurora_wrapper):
```

```
        """
        :param aurora_wrapper: An object that wraps Aurora DB cluster actions.
        """
        self.aurora_wrapper = aurora_wrapper

    def create_parameter_group(self, db_engine, parameter_group_name):
        """
        Shows how to get available engine versions for a specified database
engine and
        create a DB cluster parameter group that is compatible with a selected
engine family.

        :param db_engine: The database engine to use as a basis.
        :param parameter_group_name: The name given to the newly created
parameter group.
        :return: The newly created parameter group.
        """
        print(
            f"Checking for an existing DB cluster parameter group named
{parameter_group_name}."
        )
        parameter_group =
self.aurora_wrapper.get_parameter_group(parameter_group_name)
        if parameter_group is None:
            print(f"Getting available database engine versions for {db_engine}.")
            engine_versions = self.aurora_wrapper.get_engine_versions(db_engine)
            families = list({ver["DBParameterGroupFamily"] for ver in
engine_versions})
            family_index = q.choose("Which family do you want to use? ",
families)
            print(f"Creating a DB cluster parameter group.")
            self.aurora_wrapper.create_parameter_group(
                parameter_group_name, families[family_index], "Example parameter
group."
            )
            parameter_group = self.aurora_wrapper.get_parameter_group(
                parameter_group_name
            )
        print(f"Parameter group
{parameter_group['DBClusterParameterGroupName']}:")
        pp(parameter_group)
        print("-" * 88)
        return parameter_group
```

```
    def set_user_parameters(self, parameter_group_name):
        """
        Shows how to get the parameters contained in a custom parameter group and
        update some of the parameter values in the group.

        :param parameter_group_name: The name of the parameter group to query and
modify.
        """
        print("Let's set some parameter values in your parameter group.")
        auto_inc_parameters = self.aurora_wrapper.get_parameters(
            parameter_group_name, name_prefix="auto_increment"
        )
        update_params = []
        for auto_inc in auto_inc_parameters:
            if auto_inc["IsModifiable"] and auto_inc["DataType"] == "integer":
                print(f"The {auto_inc['ParameterName']} parameter is described
as:")
                print(f"\t{auto_inc['Description']}")
                param_range = auto_inc["AllowedValues"].split("-")
                auto_inc["ParameterValue"] = str(
                    q.ask(
                        f"Enter a value between {param_range[0]} and
{param_range[1]}: ",
                        q.is_int,
                        q.in_range(int(param_range[0]), int(param_range[1])),
                    )
                )
                update_params.append(auto_inc)
        self.aurora_wrapper.update_parameters(parameter_group_name,
update_params)
        print(
            "You can get a list of parameters you've set by specifying a source
of 'user'."
        )
        user_parameters = self.aurora_wrapper.get_parameters(
            parameter_group_name, source="user"
        )
        pp(user_parameters)
        print("-" * 88)

    def create_cluster(self, cluster_name, db_engine, db_name, parameter_group):
        """
        Shows how to create an Aurora DB cluster that contains a database of a
specified
```

```
        type. The database is also configured to use a custom DB cluster
parameter group.

        :param cluster_name: The name given to the newly created DB cluster.
        :param db_engine: The engine of the created database.
        :param db_name: The name given to the created database.
        :param parameter_group: The parameter group that is associated with the
DB cluster.
        :return: The newly created DB cluster.
        """
        print("Checking for an existing DB cluster.")
        cluster = self.aurora_wrapper.get_db_cluster(cluster_name)
        if cluster is None:
            admin_username = q.ask(
                "Enter an administrator user name for the database: ",
q.non_empty
            )
            admin_password = q.ask(
                "Enter a password for the administrator (at least 8 characters):
",
                q.non_empty,
            )
            engine_versions = self.aurora_wrapper.get_engine_versions(
                db_engine, parameter_group["DBParameterGroupFamily"]
            )
            engine_choices = [
                ver["EngineVersionDescription"] for ver in engine_versions
            ]
            print("The available engines for your parameter group are:")
            engine_index = q.choose("Which engine do you want to use? ",
engine_choices)
            print(
                f"Creating DB cluster {cluster_name} and database {db_name}.\n"
                f"The DB cluster is configured to use\n"
                f"your custom parameter group
{parameter_group['DBClusterParameterGroupName']}\n"
                f"and selected engine {engine_choices[engine_index]}.\n"
                f"This typically takes several minutes."
            )
            cluster = self.aurora_wrapper.create_db_cluster(
                cluster_name,
                parameter_group["DBClusterParameterGroupName"],
                db_name,
                db_engine,
```

```
                engine_versions[engine_index]["EngineVersion"],
                admin_username,
                admin_password,
            )
            while cluster.get("Status") != "available":
                wait(30)
                cluster = self.aurora_wrapper.get_db_cluster(cluster_name)
            print("Cluster created and available.\n")
        print("Cluster data:")
        pp(cluster)
        print("-" * 88)
        return cluster

    def create_instance(self, cluster):
        """
        Shows how to create a DB instance in an existing Aurora DB cluster. A new
DB cluster
        contains no DB instances, so you must add one. The first DB instance that
is added
        to a DB cluster defaults to a read-write DB instance.

        :param cluster: The DB cluster where the DB instance is added.
        :return: The newly created DB instance.
        """
        print("Checking for an existing database instance.")
        cluster_name = cluster["DBClusterIdentifier"]
        db_inst = self.aurora_wrapper.get_db_instance(cluster_name)
        if db_inst is None:
            print("Let's create a database instance in your DB cluster.")
            print("First, choose a DB instance type:")
            inst_opts = self.aurora_wrapper.get_orderable_instances(
                cluster["Engine"], cluster["EngineVersion"]
            )
            inst_choices = list(
                {
                    opt["DBInstanceClass"] + ", storage type: " +
opt["StorageType"]
                    for opt in inst_opts
                }
            )
            inst_index = q.choose(
                "Which DB instance class do you want to use? ", inst_choices
            )
            print(
```

```
                f"Creating a database instance. This typically takes several
 minutes."
            )
            db_inst = self.aurora_wrapper.create_instance_in_cluster(
                cluster_name,
                cluster_name,
                cluster["Engine"],
                inst_opts[inst_index]["DBInstanceClass"],
            )
            while db_inst.get("DBInstanceStatus") != "available":
                wait(30)
                db_inst = self.aurora_wrapper.get_db_instance(cluster_name)
        print("Instance data:")
        pp(db_inst)
        print("-" * 88)
        return db_inst

    @staticmethod
    def display_connection(cluster):
        """
        Displays connection information about an Aurora DB cluster and tips on
 how to
        connect to it.

        :param cluster: The DB cluster to display.
        """
        print(
            "You can now connect to your database using your favorite MySql
 client.\n"
            "One way to connect is by using the 'mysql' shell on an Amazon EC2
 instance\n"
            "that is running in the same VPC as your database cluster. Pass the
 endpoint,\n"
            "port, and administrator user name to 'mysql' and enter your password
\n"
            "when prompted:\n"
        )
        print(
            f"\n\tmysql -h {cluster['Endpoint']} -P {cluster['Port']} -u
 {cluster['MasterUsername']} -p\n"
        )
        print(
            "For more information, see the User Guide for Aurora:\n"
```

```
            "\thttps://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/
CHAP_GettingStartedAurora.CreatingConnecting.Aurora.html#CHAP_GettingStartedAurora.Aurora
        )
        print("-" * 88)

    def create_snapshot(self, cluster_name):
        """
        Shows how to create a DB cluster snapshot and wait until it's available.

        :param cluster_name: The name of a DB cluster to snapshot.
        """
        if q.ask(
            "Do you want to create a snapshot of your DB cluster (y/n)? ",
 q.is_yesno
        ):
            snapshot_id = f"{cluster_name}-{uuid.uuid4()}"
            print(
                f"Creating a snapshot named {snapshot_id}. This typically takes a
 few minutes."
            )
            snapshot = self.aurora_wrapper.create_cluster_snapshot(
                snapshot_id, cluster_name
            )
            while snapshot.get("Status") != "available":
                wait(30)
                snapshot = self.aurora_wrapper.get_cluster_snapshot(snapshot_id)
            pp(snapshot)
            print("-" * 88)

    def cleanup(self, db_inst, cluster, parameter_group):
        """
        Shows how to clean up a DB instance, DB cluster, and DB cluster parameter
 group.
        Before the DB cluster parameter group can be deleted, all associated DB
 instances and
        DB clusters must first be deleted.

        :param db_inst: The DB instance to delete.
        :param cluster: The DB cluster to delete.
        :param parameter_group: The DB cluster parameter group to delete.
        """
        cluster_name = cluster["DBClusterIdentifier"]
        parameter_group_name = parameter_group["DBClusterParameterGroupName"]
        if q.ask(
```

```
            "\nDo you want to delete the database instance, DB cluster, and
parameter "
            "group (y/n)? ",
            q.is_yesno,
        ):
            print(f"Deleting database instance
{db_inst['DBInstanceIdentifier']}.")

self.aurora_wrapper.delete_db_instance(db_inst["DBInstanceIdentifier"])
            print(f"Deleting database cluster {cluster_name}.")
            self.aurora_wrapper.delete_db_cluster(cluster_name)
            print(
                "Waiting for the DB instance and DB cluster to delete.\n"
                "This typically takes several minutes."
            )
            while db_inst is not None or cluster is not None:
                wait(30)
                if db_inst is not None:
                    db_inst = self.aurora_wrapper.get_db_instance(
                        db_inst["DBInstanceIdentifier"]
                    )
                if cluster is not None:
                    cluster = self.aurora_wrapper.get_db_cluster(
                        cluster["DBClusterIdentifier"]
                    )
            print(f"Deleting parameter group {parameter_group_name}.")
            self.aurora_wrapper.delete_parameter_group(parameter_group_name)

    def run_scenario(self, db_engine, parameter_group_name, cluster_name,
db_name):
        print("-" * 88)
        print(
            "Welcome to the Amazon Relational Database Service (Amazon RDS) get
started\n"
            "with Aurora DB clusters demo."
        )
        print("-" * 88)

        parameter_group = self.create_parameter_group(db_engine,
parameter_group_name)
        self.set_user_parameters(parameter_group_name)
        cluster = self.create_cluster(cluster_name, db_engine, db_name,
parameter_group)
        wait(5)
```

```
        db_inst = self.create_instance(cluster)
        self.display_connection(cluster)
        self.create_snapshot(cluster_name)
        self.cleanup(db_inst, cluster, parameter_group)

        print("\nThanks for watching!")
        print("-" * 88)


if __name__ == "__main__":
    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")
    try:
        scenario = AuroraClusterScenario(AuroraWrapper.from_client())
        scenario.run_scenario(
            "aurora-mysql",
            "doc-example-cluster-parameter-group",
            "doc-example-aurora",
            "docexampledb",
        )
    except Exception:
        logging.exception("Something went wrong with the demo.")
```

定义场景调用以管理 Aurora 操作的函数。

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)
```

```
   def get_parameter_group(self, parameter_group_name):
       """
       Gets a DB cluster parameter group.

       :param parameter_group_name: The name of the parameter group to retrieve.
       :return: The requested parameter group.
       """
       try:
           response = self.rds_client.describe_db_cluster_parameter_groups(
               DBClusterParameterGroupName=parameter_group_name
           )
           parameter_group = response["DBClusterParameterGroups"][0]
       except ClientError as err:
           if err.response["Error"]["Code"] == "DBParameterGroupNotFound":
               logger.info("Parameter group %s does not exist.",
parameter_group_name)
           else:
               logger.error(
                   "Couldn't get parameter group %s. Here's why: %s: %s",
                   parameter_group_name,
                   err.response["Error"]["Code"],
                   err.response["Error"]["Message"],
               )
               raise
       else:
           return parameter_group


   def create_parameter_group(
       self, parameter_group_name, parameter_group_family, description
   ):
       """
       Creates a DB cluster parameter group that is based on the specified
parameter group
       family.

       :param parameter_group_name: The name of the newly created parameter
group.
       :param parameter_group_family: The family that is used as the basis of
the new
                                      parameter group.
       :param description: A description given to the parameter group.
       :return: Data about the newly created parameter group.
```

```
        """
        try:
            response = self.rds_client.create_db_cluster_parameter_group(
                DBClusterParameterGroupName=parameter_group_name,
                DBParameterGroupFamily=parameter_group_family,
                Description=description,
            )
        except ClientError as err:
            logger.error(
                "Couldn't create parameter group %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response


    def delete_parameter_group(self, parameter_group_name):
        """
        Deletes a DB cluster parameter group.

        :param parameter_group_name: The name of the parameter group to delete.
        :return: Data about the parameter group.
        """
        try:
            response = self.rds_client.delete_db_cluster_parameter_group(
                DBClusterParameterGroupName=parameter_group_name
            )
        except ClientError as err:
            logger.error(
                "Couldn't delete parameter group %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response


    def get_parameters(self, parameter_group_name, name_prefix="", source=None):
        """
```

```
        Gets the parameters that are contained in a DB cluster parameter group.

        :param parameter_group_name: The name of the parameter group to query.
        :param name_prefix: When specified, the retrieved list of parameters is
filtered
                            to contain only parameters that start with this
prefix.
        :param source: When specified, only parameters from this source are
retrieved.
                       For example, a source of 'user' retrieves only parameters
that
                       were set by a user.
        :return: The list of requested parameters.
        """
        try:
            kwargs = {"DBClusterParameterGroupName": parameter_group_name}
            if source is not None:
                kwargs["Source"] = source
            parameters = []
            paginator =
self.rds_client.get_paginator("describe_db_cluster_parameters")
            for page in paginator.paginate(**kwargs):
                parameters += [
                    p
                    for p in page["Parameters"]
                    if p["ParameterName"].startswith(name_prefix)
                ]
        except ClientError as err:
            logger.error(
                "Couldn't get parameters for %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return parameters


    def update_parameters(self, parameter_group_name, update_parameters):
        """
        Updates parameters in a custom DB cluster parameter group.

        :param parameter_group_name: The name of the parameter group to update.
```

```
        :param update_parameters: The parameters to update in the group.
        :return: Data about the modified parameter group.
        """
        try:
            response = self.rds_client.modify_db_cluster_parameter_group(
                DBClusterParameterGroupName=parameter_group_name,
                Parameters=update_parameters,
            )
        except ClientError as err:
            logger.error(
                "Couldn't update parameters in %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response


    def get_db_cluster(self, cluster_name):
        """
        Gets data about an Aurora DB cluster.

        :param cluster_name: The name of the DB cluster to retrieve.
        :return: The retrieved DB cluster.
        """
        try:
            response = self.rds_client.describe_db_clusters(
                DBClusterIdentifier=cluster_name
            )
            cluster = response["DBClusters"][0]
        except ClientError as err:
            if err.response["Error"]["Code"] == "DBClusterNotFoundFault":
                logger.info("Cluster %s does not exist.", cluster_name)
            else:
                logger.error(
                    "Couldn't verify the existence of DB cluster %s. Here's why:
%s: %s",
                    cluster_name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
```

```
        else:
            return cluster


    def create_db_cluster(
        self,
        cluster_name,
        parameter_group_name,
        db_name,
        db_engine,
        db_engine_version,
        admin_name,
        admin_password,
    ):
        """
        Creates a DB cluster that is configured to use the specified parameter
group.
        The newly created DB cluster contains a database that uses the specified
engine and
        engine version.

        :param cluster_name: The name of the DB cluster to create.
        :param parameter_group_name: The name of the parameter group to associate
with
                                     the DB cluster.
        :param db_name: The name of the database to create.
        :param db_engine: The database engine of the database that is created,
such as MySql.
        :param db_engine_version: The version of the database engine.
        :param admin_name: The user name of the database administrator.
        :param admin_password: The password of the database administrator.
        :return: The newly created DB cluster.
        """
        try:
            response = self.rds_client.create_db_cluster(
                DatabaseName=db_name,
                DBClusterIdentifier=cluster_name,
                DBClusterParameterGroupName=parameter_group_name,
                Engine=db_engine,
                EngineVersion=db_engine_version,
                MasterUsername=admin_name,
                MasterUserPassword=admin_password,
            )
            cluster = response["DBCluster"]
```

```
        except ClientError as err:
            logger.error(
                "Couldn't create database %s. Here's why: %s: %s",
                db_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return cluster


    def delete_db_cluster(self, cluster_name):
        """
        Deletes a DB cluster.

        :param cluster_name: The name of the DB cluster to delete.
        """
        try:
            self.rds_client.delete_db_cluster(
                DBClusterIdentifier=cluster_name, SkipFinalSnapshot=True
            )
            logger.info("Deleted DB cluster %s.", cluster_name)
        except ClientError:
            logger.exception("Couldn't delete DB cluster %s.", cluster_name)
            raise


    def create_cluster_snapshot(self, snapshot_id, cluster_id):
        """
        Creates a snapshot of a DB cluster.

        :param snapshot_id: The ID to give the created snapshot.
        :param cluster_id: The DB cluster to snapshot.
        :return: Data about the newly created snapshot.
        """
        try:
            response = self.rds_client.create_db_cluster_snapshot(
                DBClusterSnapshotIdentifier=snapshot_id,
DBClusterIdentifier=cluster_id
            )
            snapshot = response["DBClusterSnapshot"]
        except ClientError as err:
            logger.error(
```

```
                "Couldn't create snapshot of %s. Here's why: %s: %s",
                cluster_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return snapshot


    def get_cluster_snapshot(self, snapshot_id):
        """
        Gets a DB cluster snapshot.

        :param snapshot_id: The ID of the snapshot to retrieve.
        :return: The retrieved snapshot.
        """
        try:
            response = self.rds_client.describe_db_cluster_snapshots(
                DBClusterSnapshotIdentifier=snapshot_id
            )
            snapshot = response["DBClusterSnapshots"][0]
        except ClientError as err:
            logger.error(
                "Couldn't get DB cluster snapshot %s. Here's why: %s: %s",
                snapshot_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return snapshot


    def create_instance_in_cluster(
        self, instance_id, cluster_id, db_engine, instance_class
    ):
        """
        Creates a database instance in an existing DB cluster. The first database
that is
        created defaults to a read-write DB instance.

        :param instance_id: The ID to give the newly created DB instance.
```

```
        :param cluster_id: The ID of the DB cluster where the DB instance is
created.
        :param db_engine: The database engine of a database to create in the DB
instance.
                          This must be compatible with the configured parameter
group
                          of the DB cluster.
        :param instance_class: The DB instance class for the newly created DB
instance.
        :return: Data about the newly created DB instance.
        """
        try:
            response = self.rds_client.create_db_instance(
                DBInstanceIdentifier=instance_id,
                DBClusterIdentifier=cluster_id,
                Engine=db_engine,
                DBInstanceClass=instance_class,
            )
            db_inst = response["DBInstance"]
        except ClientError as err:
            logger.error(
                "Couldn't create DB instance %s. Here's why: %s: %s",
                instance_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return db_inst


    def get_engine_versions(self, engine, parameter_group_family=None):
        """
        Gets database engine versions that are available for the specified engine
        and parameter group family.

        :param engine: The database engine to look up.
        :param parameter_group_family: When specified, restricts the returned
list of
                                       engine versions to those that are
compatible with
                                       this parameter group family.
        :return: The list of database engine versions.
        """
```

```
        try:
            kwargs = {"Engine": engine}
            if parameter_group_family is not None:
                kwargs["DBParameterGroupFamily"] = parameter_group_family
            response = self.rds_client.describe_db_engine_versions(**kwargs)
            versions = response["DBEngineVersions"]
        except ClientError as err:
            logger.error(
                "Couldn't get engine versions for %s. Here's why: %s: %s",
                engine,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return versions


    def get_orderable_instances(self, db_engine, db_engine_version):
        """
        Gets DB instance options that can be used to create DB instances that are
        compatible with a set of specifications.

        :param db_engine: The database engine that must be supported by the DB
instance.
        :param db_engine_version: The engine version that must be supported by
the DB instance.
        :return: The list of DB instance options that can be used to create a
compatible DB instance.
        """
        try:
            inst_opts = []
            paginator = self.rds_client.get_paginator(
                "describe_orderable_db_instance_options"
            )
            for page in paginator.paginate(
                Engine=db_engine, EngineVersion=db_engine_version
            ):
                inst_opts += page["OrderableDBInstanceOptions"]
        except ClientError as err:
            logger.error(
                "Couldn't get orderable DB instances. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
```

```
            )
            raise
        else:
            return inst_opts


    def get_db_instance(self, instance_id):
        """
        Gets data about a DB instance.

        :param instance_id: The ID of the DB instance to retrieve.
        :return: The retrieved DB instance.
        """
        try:
            response = self.rds_client.describe_db_instances(
                DBInstanceIdentifier=instance_id
            )
            db_inst = response["DBInstances"][0]
        except ClientError as err:
            if err.response["Error"]["Code"] == "DBInstanceNotFound":
                logger.info("Instance %s does not exist.", instance_id)
            else:
                logger.error(
                    "Couldn't get DB instance %s. Here's why: %s: %s",
                    instance_id,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            return db_inst


    def delete_db_instance(self, instance_id):
        """
        Deletes a DB instance.

        :param instance_id: The ID of the DB instance to delete.
        :return: Data about the deleted DB instance.
        """
        try:
            response = self.rds_client.delete_db_instance(
                DBInstanceIdentifier=instance_id,
                SkipFinalSnapshot=True,
```

```
                DeleteAutomatedBackups=True,
            )
            db_inst = response["DBInstance"]
        except ClientError as err:
            logger.error(
                "Couldn't delete DB instance %s. Here's why: %s: %s",
                instance_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return db_inst
```

- 有关 API 详细信息，请参阅《AWS SDK for Python (Boto3) API 参考》中的以下主题。
  - CreateDBCluster
  - CreateDBClusterParameterGroup
  - CreateDBClusterSnapshot
  - CreateDBInstance
  - DeleteDBCluster
  - DeleteDBClusterParameterGroup
  - DeleteDBInstance
  - DescribeDBClusterParameterGroups
  - DescribeDBClusterParameters
  - DescribeDBClusterSnapshots
  - DescribeDBClusters
  - DescribeDBEngineVersions
  - DescribeDBInstances
  - DescribeOrderableDBInstanceOptions
  - ModifyDBClusterParameterGroup

Rust

适用于 Rust 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

包含适用于 Aurora 场景的特定场景函数的库。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

use phf::{phf_set, Set};
use secrecy::SecretString;
use std::{collections::HashMap, fmt::Display, time::Duration};

use aws_sdk_rds::{
    error::ProvideErrorMetadata,

 operation::create_db_cluster_parameter_group::CreateDbClusterParameterGroupOutput,
    types::{DbCluster, DbClusterParameterGroup, DbClusterSnapshot, DbInstance,
 Parameter},
};
use sdk_examples_test_utils::waiter::Waiter;
use tracing::{info, trace, warn};

const DB_ENGINE: &str = "aurora-mysql";
const DB_CLUSTER_PARAMETER_GROUP_NAME: &str =
 "RustSDKCodeExamplesDBParameterGroup";
const DB_CLUSTER_PARAMETER_GROUP_DESCRIPTION: &str =
    "Parameter Group created by Rust SDK Code Example";
const DB_CLUSTER_IDENTIFIER: &str = "RustSDKCodeExamplesDBCluster";
const DB_INSTANCE_IDENTIFIER: &str = "RustSDKCodeExamplesDBInstance";

static FILTER_PARAMETER_NAMES: Set<&'static str> = phf_set! {
    "auto_increment_offset",
    "auto_increment_increment",
};

#[derive(Debug, PartialEq, Eq)]
```

```
struct MetadataError {
    message: Option<String>,
    code: Option<String>,
}

impl MetadataError {
    fn from(err: &dyn ProvideErrorMetadata) -> Self {
        MetadataError {
            message: err.message().map(String::from),
            code: err.code().map(String::from),
        }
    }
}

impl Display for MetadataError {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        let display = match (&self.message, &self.code) {
            (None, None) => "Unknown".to_string(),
            (None, Some(code)) => format!("({code})"),
            (Some(message), None) => message.to_string(),
            (Some(message), Some(code)) => format!("{message} ({code})"),
        };
        write!(f, "{display}")
    }
}

#[derive(Debug, PartialEq, Eq)]
pub struct ScenarioError {
    message: String,
    context: Option<MetadataError>,
}

impl ScenarioError {
    pub fn with(message: impl Into<String>) -> Self {
        ScenarioError {
            message: message.into(),
            context: None,
        }
    }

    pub fn new(message: impl Into<String>, err: &dyn ProvideErrorMetadata) ->
 Self {
        ScenarioError {
            message: message.into(),
```

```
            context: Some(MetadataError::from(err)),
        }
    }
}

impl std::error::Error for ScenarioError {}
impl Display for ScenarioError {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        match &self.context {
            Some(c) => write!(f, "{}: {}", self.message, c),
            None => write!(f, "{}", self.message),
        }
    }
}

// Parse the ParameterName, Description, and AllowedValues values and display
 them.
#[derive(Debug)]
pub struct AuroraScenarioParameter {
    name: String,
    allowed_values: String,
    current_value: String,
}

impl Display for AuroraScenarioParameter {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        write!(
            f,
            "{}: {} (allowed: {})",
            self.name, self.current_value, self.allowed_values
        )
    }
}

impl From<aws_sdk_rds::types::Parameter> for AuroraScenarioParameter {
    fn from(value: aws_sdk_rds::types::Parameter) -> Self {
        AuroraScenarioParameter {
            name: value.parameter_name.unwrap_or_default(),
            allowed_values: value.allowed_values.unwrap_or_default(),
            current_value: value.parameter_value.unwrap_or_default(),
        }
    }
}
```

```
pub struct AuroraScenario {
    rds: crate::rds::Rds,
    engine_family: Option<String>,
    engine_version: Option<String>,
    instance_class: Option<String>,
    db_cluster_parameter_group: Option<DbClusterParameterGroup>,
    db_cluster_identifier: Option<String>,
    db_instance_identifier: Option<String>,
    username: Option<String>,
    password: Option<SecretString>,
}

impl AuroraScenario {
    pub fn new(client: crate::rds::Rds) -> Self {
        AuroraScenario {
            rds: client,
            engine_family: None,
            engine_version: None,
            instance_class: None,
            db_cluster_parameter_group: None,
            db_cluster_identifier: None,
            db_instance_identifier: None,
            username: None,
            password: None,
        }
    }

    // snippet-start:[rust.aurora.get_engines.usage]
    // Get available engine families for Aurora MySql.
 rds.DescribeDbEngineVersions(Engine='aurora-mysql') and build a set of the
 'DBParameterGroupFamily' field values. I get {aurora-mysql8.0, aurora-mysql5.7}.
    pub async fn get_engines(&self) -> Result<HashMap<String, Vec<String>>,
 ScenarioError> {
        let describe_db_engine_versions =
 self.rds.describe_db_engine_versions(DB_ENGINE).await;
        trace!(versions=?describe_db_engine_versions, "full list of versions");

        if let Err(err) = describe_db_engine_versions {
            return Err(ScenarioError::new(
                "Failed to retrieve DB Engine Versions",
                &err,
            ));
        };
```

```
        let version_count = describe_db_engine_versions
            .as_ref()
            .map(|o| o.db_engine_versions().len())
            .unwrap_or_default();
        info!(version_count, "got list of versions");

        // Create a map of engine families to their available versions.
        let mut versions = HashMap::<String, Vec<String>>::new();
        describe_db_engine_versions
            .unwrap()
            .db_engine_versions()
            .iter()
            .filter_map(
                |v| match (&v.db_parameter_group_family, &v.engine_version) {
                    (Some(family), Some(version)) => Some((family.clone(),
version.clone())),
                    _ => None,
                },
            )
            .for_each(|(family, version)|
versions.entry(family).or_default().push(version));

        Ok(versions)
    }
    // snippet-end:[rust.aurora.get_engines.usage]

    // snippet-start:[rust.aurora.get_instance_classes.usage]
    pub async fn get_instance_classes(&self) -> Result<Vec<String>,
ScenarioError> {
        let describe_orderable_db_instance_options_items = self
            .rds
            .describe_orderable_db_instance_options(
                DB_ENGINE,
                self.engine_version
                    .as_ref()
                    .expect("engine version for db instance options")
                    .as_str(),
            )
            .await;

        describe_orderable_db_instance_options_items
            .map(|options| {
                options
                    .iter()
```

```
                    .filter(|o| o.storage_type() == Some("aurora"))
                    .map(|o|
o.db_instance_class().unwrap_or_default().to_string())
                    .collect::<Vec<String>>()
            })
            .map_err(|err| ScenarioError::new("Could not get available instance
classes", &err))
    }
    // snippet-end:[rust.aurora.get_instance_classes.usage]

    // snippet-start:[rust.aurora.set_engine.usage]
    // Select an engine family and create a custom DB cluster parameter group.
rds.CreateDbClusterParameterGroup(DBParameterGroupFamily='aurora-mysql8.0')
    pub async fn set_engine(&mut self, engine: &str, version: &str) -> Result<(),
ScenarioError> {
        self.engine_family = Some(engine.to_string());
        self.engine_version = Some(version.to_string());
        let create_db_cluster_parameter_group = self
            .rds
            .create_db_cluster_parameter_group(
                DB_CLUSTER_PARAMETER_GROUP_NAME,
                DB_CLUSTER_PARAMETER_GROUP_DESCRIPTION,
                engine,
            )
            .await;

        match create_db_cluster_parameter_group {
            Ok(CreateDbClusterParameterGroupOutput {
                db_cluster_parameter_group: None,
                ..
            }) => {
                return Err(ScenarioError::with(
                    "CreateDBClusterParameterGroup had empty response",
                ));
            }
            Err(error) => {
                if error.code() == Some("DBParameterGroupAlreadyExists") {
                    info!("Cluster Parameter Group already exists, nothing to
do");
                } else {
                    return Err(ScenarioError::new(
                        "Could not create Cluster Parameter Group",
                        &error,
                    ));
```

```
                }
            }
            _ => {
                info!("Created Cluster Parameter Group");
            }
        }

        Ok(())
    }
    // snippet-end:[rust.aurora.set_engine.usage]

    pub fn set_instance_class(&mut self, instance_class: Option<String>) {
        self.instance_class = instance_class;
    }

    pub fn set_login(&mut self, username: Option<String>, password:
Option<SecretString>) {
        self.username = username;
        self.password = password;
    }

    pub async fn connection_string(&self) -> Result<String, ScenarioError> {
        let cluster = self.get_cluster().await?;
        let endpoint = cluster.endpoint().unwrap_or_default();
        let port = cluster.port().unwrap_or_default();
        let username = cluster.master_username().unwrap_or_default();
        Ok(format!("mysql -h {endpoint} -P {port} -u {username} -p"))
    }

    // snippet-start:[rust.aurora.get_cluster.usage]
    pub async fn get_cluster(&self) -> Result<DbCluster, ScenarioError> {
        let describe_db_clusters_output = self
            .rds
            .describe_db_clusters(
                self.db_cluster_identifier
                    .as_ref()
                    .expect("cluster identifier")
                    .as_str(),
            )
            .await;
        if let Err(err) = describe_db_clusters_output {
            return Err(ScenarioError::new("Failed to get cluster", &err));
        }
```

```
        let db_cluster = describe_db_clusters_output
            .unwrap()
            .db_clusters
            .and_then(|output| output.first().cloned());

        db_cluster.ok_or_else(|| ScenarioError::with("Did not find the cluster"))
    }
    // snippet-end:[rust.aurora.get_cluster.usage]

    // snippet-start:[rust.aurora.cluster_parameters.usage]
    // Get the parameter group. rds.DescribeDbClusterParameterGroups
    // Get parameters in the group. This is a long list so you will have to
paginate. Find the auto_increment_offset and auto_increment_increment parameters
(by ParameterName). rds.DescribeDbClusterParameters
    // Parse the ParameterName, Description, and AllowedValues values and display
them.
    pub async fn cluster_parameters(&self) ->
Result<Vec<AuroraScenarioParameter>, ScenarioError> {
        let parameters_output = self
            .rds
            .describe_db_cluster_parameters(DB_CLUSTER_PARAMETER_GROUP_NAME)
            .await;

        if let Err(err) = parameters_output {
            return Err(ScenarioError::new(
                format!("Failed to retrieve parameters for
{DB_CLUSTER_PARAMETER_GROUP_NAME}"),
                &err,
            ));
        }

        let parameters = parameters_output
            .unwrap()
            .into_iter()
            .flat_map(|p| p.parameters.unwrap_or_default().into_iter())
            .filter(|p|
FILTER_PARAMETER_NAMES.contains(p.parameter_name().unwrap_or_default()))
            .map(AuroraScenarioParameter::from)
            .collect::<Vec<_>>();

        Ok(parameters)
    }
    // snippet-end:[rust.aurora.cluster_parameters.usage]
```

```
    // snippet-start:[rust.aurora.update_auto_increment.usage]
    // Modify both the auto_increment_offset and auto_increment_increment
parameters in one call in the custom parameter group. Set their ParameterValue
fields to a new allowable value. rds.ModifyDbClusterParameterGroup.
    pub async fn update_auto_increment(
        &self,
        offset: u8,
        increment: u8,
    ) -> Result<(), ScenarioError> {
        let modify_db_cluster_parameter_group = self
            .rds
            .modify_db_cluster_parameter_group(
                DB_CLUSTER_PARAMETER_GROUP_NAME,
                vec![
                    Parameter::builder()
                        .parameter_name("auto_increment_offset")
                        .parameter_value(format!("{offset}"))
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                    Parameter::builder()
                        .parameter_name("auto_increment_increment")
                        .parameter_value(format!("{increment}"))
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                ],
            )
            .await;

        if let Err(error) = modify_db_cluster_parameter_group {
            return Err(ScenarioError::new(
                "Failed to modify cluster parameter group",
                &error,
            ));
        }

        Ok(())
    }
    // snippet-end:[rust.aurora.update_auto_increment.usage]

    // snippet-start:[rust.aurora.start_cluster_and_instance.usage]
    // Get a list of allowed engine versions.
rds.DescribeDbEngineVersions(Engine='aurora-mysql', DBParameterGroupFamily=<the
family used to create your parameter group in step 2>)
```

```
   // Create an Aurora DB cluster database cluster that contains a MySql
database and uses the parameter group you created.
   // Wait for DB cluster to be ready. Call rds.DescribeDBClusters and check for
Status == 'available'.
   // Get a list of instance classes available for the selected engine
and engine version. rds.DescribeOrderableDbInstanceOptions(Engine='mysql',
EngineVersion=).

   // Create a database instance in the cluster.
   // Wait for DB instance to be ready. Call rds.DescribeDbInstances and check
for DBInstanceStatus == 'available'.
   pub async fn start_cluster_and_instance(&mut self) -> Result<(),
ScenarioError> {
       if self.password.is_none() {
           return Err(ScenarioError::with(
               "Must set Secret Password before starting a cluster",
           ));
       }
       let create_db_cluster = self
           .rds
           .create_db_cluster(
               DB_CLUSTER_IDENTIFIER,
               DB_CLUSTER_PARAMETER_GROUP_NAME,
               DB_ENGINE,
               self.engine_version.as_deref().expect("engine version"),
               self.username.as_deref().expect("username"),
               self.password
                   .replace(SecretString::new("".to_string()))
                   .expect("password"),
           )
           .await;
       if let Err(err) = create_db_cluster {
           return Err(ScenarioError::new(
               "Failed to create DB Cluster with cluster group",
               &err,
           ));
       }

       self.db_cluster_identifier = create_db_cluster
           .unwrap()
           .db_cluster
           .and_then(|c| c.db_cluster_identifier);

       if self.db_cluster_identifier.is_none() {
```

```
            return Err(ScenarioError::with("Created DB Cluster missing
Identifier"));
        }

        info!(
            "Started a db cluster: {}",
            self.db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing ARN")
        );

        let create_db_instance = self
            .rds
            .create_db_instance(
                self.db_cluster_identifier.as_deref().expect("cluster name"),
                DB_INSTANCE_IDENTIFIER,
                self.instance_class.as_deref().expect("instance class"),
                DB_ENGINE,
            )
            .await;
        if let Err(err) = create_db_instance {
            return Err(ScenarioError::new(
                "Failed to create Instance in DB Cluster",
                &err,
            ));
        }

        self.db_instance_identifier = create_db_instance
            .unwrap()
            .db_instance
            .and_then(|i| i.db_instance_identifier);

        // Cluster creation can take up to 20 minutes to become available
        let cluster_max_wait = Duration::from_secs(20 * 60);
        let waiter = Waiter::builder().max(cluster_max_wait).build();
        while waiter.sleep().await.is_ok() {
            let cluster = self
                .rds
                .describe_db_clusters(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;
```

```
            if let Err(err) = cluster {
                warn!(?err, "Failed to describe cluster while waiting for
ready");
                continue;
            }

            let instance = self
                .rds
                .describe_db_instance(
                    self.db_instance_identifier
                        .as_deref()
                        .expect("instance identifier"),
                )
                .await;
            if let Err(err) = instance {
                return Err(ScenarioError::new(
                    "Failed to find instance for cluster",
                    &err,
                ));
            }

            let instances_available = instance
                .unwrap()
                .db_instances()
                .iter()
                .all(|instance| instance.db_instance_status() ==
Some("Available"));

            let endpoints = self
                .rds
                .describe_db_cluster_endpoints(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = endpoints {
                return Err(ScenarioError::new(
                    "Failed to find endpoint for cluster",
                    &err,
                ));
            }
```

```
            let endpoints_available = endpoints
                .unwrap()
                .db_cluster_endpoints()
                .iter()
                .all(|endpoint| endpoint.status() == Some("available"));

            if instances_available && endpoints_available {
                return Ok(());
            }
        }

        Err(ScenarioError::with("timed out waiting for cluster"))
    }
    // snippet-end:[rust.aurora.start_cluster_and_instance.usage]

    // snippet-start:[rust.aurora.snapshot.usage]
    // Create a snapshot of the DB cluster. rds.CreateDbClusterSnapshot.
    // Wait for the snapshot to create. rds.DescribeDbClusterSnapshots until
Status == 'available'.
    pub async fn snapshot(&self, name: &str) -> Result<DbClusterSnapshot,
ScenarioError> {
        let id = self.db_cluster_identifier.as_deref().unwrap_or_default();
        let snapshot = self
            .rds
            .snapshot_cluster(id, format!("{id}_{name}").as_str())
            .await;
        match snapshot {
            Ok(output) => match output.db_cluster_snapshot {
                Some(snapshot) => Ok(snapshot),
                None => Err(ScenarioError::with("Missing Snapshot")),
            },
            Err(err) => Err(ScenarioError::new("Failed to create snapshot",
&err)),
        }
    }
    // snippet-end:[rust.aurora.snapshot.usage]

    // snippet-start:[rust.aurora.clean_up.usage]
    pub async fn clean_up(self) -> Result<(), Vec<ScenarioError>> {
        let mut clean_up_errors: Vec<ScenarioError> = vec![];

        // Delete the instance. rds.DeleteDbInstance.
        let delete_db_instance = self
```

```
            .rds
            .delete_db_instance(
                self.db_instance_identifier
                    .as_deref()
                    .expect("instance identifier"),
            )
            .await;
        if let Err(err) = delete_db_instance {
            let identifier = self
                .db_instance_identifier
                .as_deref()
                .unwrap_or("Missing Instance Identifier");
            let message = format!("failed to delete db instance {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance to delete
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_instances =
self.rds.describe_db_instances().await;
                if let Err(err) = describe_db_instances {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check instance state during deletion",
                        &err,
                    ));
                    break;
                }
                let db_instances = describe_db_instances
                    .unwrap()
                    .db_instances()
                    .iter()
                    .filter(|instance| instance.db_cluster_identifier ==
self.db_cluster_identifier)
                    .cloned()
                    .collect::<Vec<DbInstance>>();

                if db_instances.is_empty() {
                    trace!("Delete Instance waited and no instances were found");
                    break;
                }
                match db_instances.first().unwrap().db_instance_status() {
                    Some("Deleting") => continue,
                    Some(status) => {
```

```
                        info!("Attempting to delete but instances is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB instance");
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster. rds.DeleteDbCluster.
        let delete_db_cluster = self
            .rds
            .delete_db_cluster(
                self.db_cluster_identifier
                    .as_deref()
                    .expect("cluster identifier"),
            )
            .await;

        if let Err(err) = delete_db_cluster {
            let identifier = self
                .db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing DB Cluster Identifier");
            let message = format!("failed to delete db cluster {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance and cluster to fully delete.
rds.DescribeDbInstances and rds.DescribeDbClusters until both are not found.
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_clusters = self
                    .rds
                    .describe_db_clusters(
                        self.db_cluster_identifier
                            .as_deref()
                            .expect("cluster identifier"),
                    )
                    .await;
                if let Err(err) = describe_db_clusters {
                    clean_up_errors.push(ScenarioError::new(
```

```
                        "Failed to check cluster state during deletion",
                        &err,
                    ));
                    break;
                }
                let describe_db_clusters = describe_db_clusters.unwrap();
                let db_clusters = describe_db_clusters.db_clusters();
                if db_clusters.is_empty() {
                    trace!("Delete cluster waited and no clusters were found");
                    break;
                }
                match db_clusters.first().unwrap().status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but clusters is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB cluster");
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster parameter group.
rds.DeleteDbClusterParameterGroup.
        let delete_db_cluster_parameter_group = self
            .rds
            .delete_db_cluster_parameter_group(
                self.db_cluster_parameter_group
                    .map(|g| {
                        g.db_cluster_parameter_group_name
                            .unwrap_or_else(||
DB_CLUSTER_PARAMETER_GROUP_NAME.to_string())
                    })
                    .as_deref()
                    .expect("cluster parameter group name"),
            )
            .await;
        if let Err(error) = delete_db_cluster_parameter_group {
            clean_up_errors.push(ScenarioError::new(
                "Failed to delete the db cluster parameter group",
```

```
                &error,
            ))
        }

        if clean_up_errors.is_empty() {
            Ok(())
        } else {
            Err(clean_up_errors)
        }
    }
    // snippet-end:[rust.aurora.clean_up.usage]
}

#[cfg(test)]
pub mod tests;
```

围绕 RDS 客户端包装器使用自动模拟来对该库进行测试。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

use crate::rds::MockRdsImpl;

use super::*;

use std::io::{Error, ErrorKind};

use assert_matches::assert_matches;
use aws_sdk_rds::{
    error::SdkError,
    operation::{
        create_db_cluster::{CreateDBClusterError, CreateDbClusterOutput},
        create_db_cluster_parameter_group::CreateDBClusterParameterGroupError,
        create_db_cluster_snapshot::{CreateDBClusterSnapshotError,
 CreateDbClusterSnapshotOutput},
        create_db_instance::{CreateDBInstanceError, CreateDbInstanceOutput},
        delete_db_cluster::DeleteDbClusterOutput,
        delete_db_cluster_parameter_group::DeleteDbClusterParameterGroupOutput,
        delete_db_instance::DeleteDbInstanceOutput,
        describe_db_cluster_endpoints::DescribeDbClusterEndpointsOutput,
        describe_db_cluster_parameters::{
            DescribeDBClusterParametersError, DescribeDbClusterParametersOutput,
```

```
        },
        describe_db_clusters::{DescribeDBClustersError,
 DescribeDbClustersOutput},
        describe_db_engine_versions::{
            DescribeDBEngineVersionsError, DescribeDbEngineVersionsOutput,
        },
        describe_db_instances::{DescribeDBInstancesError,
 DescribeDbInstancesOutput},

 describe_orderable_db_instance_options::DescribeOrderableDBInstanceOptionsError,
        modify_db_cluster_parameter_group::{
            ModifyDBClusterParameterGroupError,
 ModifyDbClusterParameterGroupOutput,
        },
    },
    types::{
        error::DbParameterGroupAlreadyExistsFault, DbClusterEndpoint,
 DbEngineVersion,
        OrderableDbInstanceOption,
    },
};
use aws_smithy_runtime_api::http::{Response, StatusCode};
use aws_smithy_types::body::SdkBody;
use mockall::predicate::eq;
use secrecy::ExposeSecret;

// snippet-start:[rust.aurora.set_engine.test]
#[tokio::test]
async fn test_scenario_set_engine() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .with(
            eq("RustSDKCodeExamplesDBParameterGroup"),
            eq("Parameter Group created by Rust SDK Code Example"),
            eq("aurora-mysql"),
        )
        .return_once(|_, _, _| {
            Ok(CreateDbClusterParameterGroupOutput::builder()

 .db_cluster_parameter_group(DbClusterParameterGroup::builder().build())
                .build())
        });
```

```
    let mut scenario = AuroraScenario::new(mock_rds);

    let set_engine = scenario.set_engine("aurora-mysql", "aurora-
mysql8.0").await;

    assert_eq!(set_engine, Ok(()));
    assert_eq!(Some("aurora-mysql"), scenario.engine_family.as_deref());
    assert_eq!(Some("aurora-mysql8.0"), scenario.engine_version.as_deref());
}

#[tokio::test]
async fn test_scenario_set_engine_not_create() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .with(
            eq("RustSDKCodeExamplesDBParameterGroup"),
            eq("Parameter Group created by Rust SDK Code Example"),
            eq("aurora-mysql"),
        )
        .return_once(|_, _, _|
 Ok(CreateDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);

    let set_engine = scenario.set_engine("aurora-mysql", "aurora-
mysql8.0").await;

    assert!(set_engine.is_err());
}

#[tokio::test]
async fn test_scenario_set_engine_param_group_exists() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .withf(|_, _, _| true)
        .return_once(|_, _, _| {
            Err(SdkError::service_error(

 CreateDBClusterParameterGroupError::DbParameterGroupAlreadyExistsFault(
```

```
                    DbParameterGroupAlreadyExistsFault::builder().build(),
                ),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);

    let set_engine = scenario.set_engine("aurora-mysql", "aurora-
mysql8.0").await;

    assert!(set_engine.is_err());
}
// snippet-end:[rust.aurora.set_engine.test]

// snippet-start:[rust.aurora.get_engines.test]
#[tokio::test]
async fn test_scenario_get_engines() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_engine_versions()
        .with(eq("aurora-mysql"))
        .return_once(|_| {
            Ok(DescribeDbEngineVersionsOutput::builder()
                .db_engine_versions(
                    DbEngineVersion::builder()
                        .db_parameter_group_family("f1")
                        .engine_version("f1a")
                        .build(),
                )
                .db_engine_versions(
                    DbEngineVersion::builder()
                        .db_parameter_group_family("f1")
                        .engine_version("f1b")
                        .build(),
                )
                .db_engine_versions(
                    DbEngineVersion::builder()
                        .db_parameter_group_family("f2")
                        .engine_version("f2a")
                        .build(),
                )
```

```
                .db_engine_versions(DbEngineVersion::builder().build())
                .build())
        });

    let scenario = AuroraScenario::new(mock_rds);

    let versions_map = scenario.get_engines().await;

    assert_eq!(
        versions_map,
        Ok(HashMap::from([
            ("f1".into(), vec!["f1a".into(), "f1b".into()]),
            ("f2".into(), vec!["f2a".into()])
        ]))
    );
}

#[tokio::test]
async fn test_scenario_get_engines_failed() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_engine_versions()
        .with(eq("aurora-mysql"))
        .return_once(|_| {
            Err(SdkError::service_error(
                DescribeDBEngineVersionsError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe_db_engine_versions error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let scenario = AuroraScenario::new(mock_rds);

    let versions_map = scenario.get_engines().await;
    assert_matches!(
        versions_map,
        Err(ScenarioError { message, context: _ }) if message == "Failed to
 retrieve DB Engine Versions"
    );
}
```

```
// snippet-end:[rust.aurora.get_engines.test]

// snippet-start:[rust.aurora.get_instance_classes.test]
#[tokio::test]
async fn test_scenario_get_instance_classes() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .return_once(|_, _, _| {
            Ok(CreateDbClusterParameterGroupOutput::builder()

 .db_cluster_parameter_group(DbClusterParameterGroup::builder().build())
                .build())
        });

    mock_rds
        .expect_describe_orderable_db_instance_options()
        .with(eq("aurora-mysql"), eq("aurora-mysql8.0"))
        .return_once(|_, _| {
            Ok(vec![
                OrderableDbInstanceOption::builder()
                    .db_instance_class("t1")
                    .storage_type("aurora")
                    .build(),
                OrderableDbInstanceOption::builder()
                    .db_instance_class("t1")
                    .storage_type("aurora-iopt1")
                    .build(),
                OrderableDbInstanceOption::builder()
                    .db_instance_class("t2")
                    .storage_type("aurora")
                    .build(),
                OrderableDbInstanceOption::builder()
                    .db_instance_class("t3")
                    .storage_type("aurora")
                    .build(),
            ])
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario
        .set_engine("aurora-mysql", "aurora-mysql8.0")
        .await
```

```
        .expect("set engine");

    let instance_classes = scenario.get_instance_classes().await;

    assert_eq!(
        instance_classes,
        Ok(vec!["t1".into(), "t2".into(), "t3".into()])
    );
}

#[tokio::test]
async fn test_scenario_get_instance_classes_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_orderable_db_instance_options()
        .with(eq("aurora-mysql"), eq("aurora-mysql8.0"))
        .return_once(|_, _| {
            Err(SdkError::service_error(

 DescribeOrderableDBInstanceOptionsError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe_orderable_db_instance_options_error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_family = Some("aurora-mysql".into());
    scenario.engine_version = Some("aurora-mysql8.0".into());

    let instance_classes = scenario.get_instance_classes().await;

    assert_matches!(
        instance_classes,
        Err(ScenarioError {message, context: _}) if message == "Could not get
 available instance classes"
    );
}
// snippet-end:[rust.aurora.get_instance_classes.test]

// snippet-start:[rust.aurora.get_cluster.test]
```

```
#[tokio::test]
async fn test_scenario_get_cluster() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(DbCluster::builder().build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());
    let cluster = scenario.get_cluster().await;

    assert!(cluster.is_ok());
}

#[tokio::test]
async fn test_scenario_get_cluster_missing_cluster() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .return_once(|_, _, _| {
            Ok(CreateDbClusterParameterGroupOutput::builder()

 .db_cluster_parameter_group(DbClusterParameterGroup::builder().build())
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| Ok(DescribeDbClustersOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());
    let cluster = scenario.get_cluster().await;

    assert_matches!(cluster, Err(ScenarioError { message, context: _ }) if
 message == "Did not find the cluster");
```

```
}

#[tokio::test]
async fn test_scenario_get_cluster_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .return_once(|_, _, _| {
            Ok(CreateDbClusterParameterGroupOutput::builder()

 .db_cluster_parameter_group(DbClusterParameterGroup::builder().build())
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe_db_clusters_error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());
    let cluster = scenario.get_cluster().await;

    assert_matches!(cluster, Err(ScenarioError { message, context: _ }) if
 message == "Failed to get cluster");
}
// snippet-end:[rust.aurora.get_cluster.test]

#[tokio::test]
async fn test_scenario_connection_string() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_clusters()
```

```
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .endpoint("test_endpoint")
                        .port(3306)
                        .master_username("test_username")
                        .build(),
                )
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());
    let connection_string = scenario.connection_string().await;

    assert_eq!(
        connection_string,
        Ok("mysql -h test_endpoint -P 3306 -u test_username -p".into())
    );
}

// snippet-start:[rust.aurora.cluster_parameters.test]
#[tokio::test]
async fn test_scenario_cluster_parameters() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_cluster_parameters()
        .with(eq("RustSDKCodeExamplesDBParameterGroup"))
        .return_once(|_| {
            Ok(vec![DescribeDbClusterParametersOutput::builder()
                .parameters(Parameter::builder().parameter_name("a").build())
                .parameters(Parameter::builder().parameter_name("b").build())
                .parameters(
                    Parameter::builder()
                        .parameter_name("auto_increment_offset")
                        .build(),
                )
                .parameters(Parameter::builder().parameter_name("c").build())
                .parameters(
                    Parameter::builder()
                        .parameter_name("auto_increment_increment")
```

```
                        .build(),
                )
                .parameters(Parameter::builder().parameter_name("d").build())
                .build()])
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());

    let params = scenario.cluster_parameters().await.expect("cluster params");
    let names: Vec<String> = params.into_iter().map(|p| p.name).collect();
    assert_eq!(
        names,
        vec!["auto_increment_offset", "auto_increment_increment"]
    );
}

#[tokio::test]
async fn test_scenario_cluster_parameters_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_cluster_parameters()
        .with(eq("RustSDKCodeExamplesDBParameterGroup"))
        .return_once(|_| {
            Err(SdkError::service_error(
                DescribeDBClusterParametersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe_db_cluster_parameters_error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());
    let params = scenario.cluster_parameters().await;
    assert_matches!(params, Err(ScenarioError { message, context: _ }) if message
 == "Failed to retrieve parameters for RustSDKCodeExamplesDBParameterGroup");
}
// snippet-end:[rust.aurora.cluster_parameters.test]

// snippet-start:[rust.aurora.update_auto_increment.test]
```

```
#[tokio::test]
async fn test_scenario_update_auto_increment() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_modify_db_cluster_parameter_group()
        .withf(|name, params| {
            assert_eq!(name, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(
                params,
                &vec![
                    Parameter::builder()
                        .parameter_name("auto_increment_offset")
                        .parameter_value("10")
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                    Parameter::builder()
                        .parameter_name("auto_increment_increment")
                        .parameter_value("20")
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                ]
            );
            true
        })
        .return_once(|_, _|
 Ok(ModifyDbClusterParameterGroupOutput::builder().build()));

    let scenario = AuroraScenario::new(mock_rds);

    scenario
        .update_auto_increment(10, 20)
        .await
        .expect("update auto increment");
}

#[tokio::test]
async fn test_scenario_update_auto_increment_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_modify_db_cluster_parameter_group()
        .return_once(|_, _| {
            Err(SdkError::service_error(
```

```
 ModifyDBClusterParameterGroupError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "modify_db_cluster_parameter_group_error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let scenario = AuroraScenario::new(mock_rds);

    let update = scenario.update_auto_increment(10, 20).await;
    assert_matches!(update, Err(ScenarioError { message, context: _}) if message
 == "Failed to modify cluster parameter group");
}
// snippet-end:[rust.aurora.update_auto_increment.test]

// snippet-start:[rust.aurora.start_cluster_and_instance.test]
#[tokio::test]
async fn test_start_cluster_and_instance() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
```

```
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|id| {
            Ok(DescribeDbClustersOutput::builder()

.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_describe_db_instance()
        .with(eq("RustSDKCodeExamplesDBInstance"))
        .return_once(|name| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_instance_identifier(name)
                        .db_instance_status("Available")
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_cluster_endpoints()
```

```
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

 .db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
        assert!(scenario
            .password
            .replace(SecretString::new("BAD SECRET".into()))
            .unwrap()
            .expose_secret()
            .is_empty());
        assert_eq!(
            scenario.db_cluster_identifier,
            Some("RustSDKCodeExamplesDBCluster".into())
        );
    });
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Err(SdkError::service_error(
                CreateDBClusterError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
```

```
                    "create db cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _}) if message
 == "Failed to create DB Cluster with cluster group")
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_missing_id() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()
                .db_cluster(DbCluster::builder().build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context:_ }) if message
 == "Created DB Cluster missing Identifier");
}

#[tokio::test]
async fn test_start_cluster_and_instance_instance_create_error() {
    let mut mock_rds = MockRdsImpl::default();
```

```
    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .return_once(|_, _, _, _| {
            Err(SdkError::service_error(
                CreateDBInstanceError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db instance error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _ }) if message
 == "Failed to create Instance in DB Cluster")
}

#[tokio::test]
```

```
async fn test_start_cluster_and_instance_wait_hiccup() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
```

```
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        })
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()

.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds.expect_describe_db_instance().return_once(|name| {
        Ok(DescribeDbInstancesOutput::builder()
            .db_instances(
                DbInstance::builder()
                    .db_instance_identifier(name)
                    .db_instance_status("Available")
                    .build(),
            )
            .build())
    });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

.db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
```

```
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}
// snippet-end:[rust.aurora.start_cluster_and_instance.test]

// snippet-start:[rust.aurora.clean_up.test]
#[tokio::test]
async fn test_scenario_clean_up() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| Ok(DescribeDbInstancesOutput::builder().build()));
```

```
    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| Ok(DescribeDbClustersOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_ok());
    });
```

```
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_scenario_clean_up_errors() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| {
            Err(SdkError::service_error(
                DescribeDBInstancesError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db instances error",
                ))),
```

```
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db clusters error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
```

```
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_err());
        let errs = clean_up.unwrap_err();
        assert_eq!(errs.len(), 2);
        assert_matches!(errs.get(0), Some(ScenarioError {message, context: _}) if
 message == "Failed to check instance state during deletion");
        assert_matches!(errs.get(1), Some(ScenarioError {message, context: _}) if
 message == "Failed to check cluster state during deletion");
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}
// snippet-end:[rust.aurora.clean_up.test]

// snippet-start:[rust.aurora.snapshot.test]
#[tokio::test]
async fn test_scenario_snapshot() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_snapshot_cluster()
        .with(eq("MockCluster"), eq("MockCluster_MockSnapshot"))
        .times(1)
        .return_once(|_, _| {
            Ok(CreateDbClusterSnapshotOutput::builder()
```

```
                .db_cluster_snapshot(
                    DbClusterSnapshot::builder()
                        .db_cluster_identifier("MockCluster")

 .db_cluster_snapshot_identifier("MockCluster_MockSnapshot")
                        .build(),
                )
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("MockCluster".into());
    let create_snapshot = scenario.snapshot("MockSnapshot").await;
    assert!(create_snapshot.is_ok());
}

#[tokio::test]
async fn test_scenario_snapshot_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_snapshot_cluster()
        .with(eq("MockCluster"), eq("MockCluster_MockSnapshot"))
        .times(1)
        .return_once(|_, _| {
            Err(SdkError::service_error(
                CreateDBClusterSnapshotError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create snapshot error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("MockCluster".into());
    let create_snapshot = scenario.snapshot("MockSnapshot").await;
    assert_matches!(create_snapshot, Err(ScenarioError { message, context: _}) if
 message == "Failed to create snapshot");
}

#[tokio::test]
async fn test_scenario_snapshot_invalid() {
```

```
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_snapshot_cluster()
        .with(eq("MockCluster"), eq("MockCluster_MockSnapshot"))
        .times(1)
        .return_once(|_, _|
 Ok(CreateDbClusterSnapshotOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("MockCluster".into());
    let create_snapshot = scenario.snapshot("MockSnapshot").await;
    assert_matches!(create_snapshot, Err(ScenarioError { message, context: _}) if
 message == "Missing Snapshot");
}
// snippet-end:[rust.aurora.snapshot.test]
```

一个从前端到末端运行场景的二进制文件，使用查询器，以便用户可以做出一些决定。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

use std::fmt::Display;

use anyhow::anyhow;
use aurora_code_examples::{
    aurora_scenario::{AuroraScenario, ScenarioError},
    rds::Rds as RdsClient,
};
use aws_sdk_rds::Client;
use inquire::{validator::StringValidator, CustomUserError};
use secrecy::SecretString;
use tracing::warn;

#[derive(Default, Debug)]
struct Warnings(Vec<String>);

impl Warnings {
    fn new() -> Self {
        Warnings(Vec::with_capacity(5))
    }
```

```
    fn push(&mut self, warning: &str, error: ScenarioError) {
        let formatted = format!("{warning}: {error}");
        warn!("{formatted}");
        self.0.push(formatted);
    }

    fn is_empty(&self) -> bool {
        self.0.is_empty()
    }
}

impl Display for Warnings {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        writeln!(f, "Warnings:")?;
        for warning in &self.0 {
            writeln!(f, "{: >4}- {warning}", "")?;
        }
        Ok(())
    }
}

fn select(
    prompt: &str,
    choices: Vec<String>,
    error_message: &str,
) -> Result<String, anyhow::Error> {
    inquire::Select::new(prompt, choices)
        .prompt()
        .map_err(|error| anyhow!("{error_message}: {error}"))
}

// Prepare the Aurora Scenario. Prompt for several settings that are optional to
 the Scenario, but that the user should choose for the demo.
// This includes the engine, engine version, and instance class.
async fn prepare_scenario(rds: RdsClient) -> Result<AuroraScenario,
 anyhow::Error> {
    let mut scenario = AuroraScenario::new(rds);

    // Get available engine families for Aurora MySql.
 rds.DescribeDbEngineVersions(Engine='aurora-mysql') and build a set of the
 'DBParameterGroupFamily' field values. I get {aurora-mysql8.0, aurora-mysql5.7}.
    let available_engines = scenario.get_engines().await;
    if let Err(error) = available_engines {
        return Err(anyhow!("Failed to get available engines: {}", error));
```

```
    }
    let available_engines = available_engines.unwrap();

    // Select an engine family and create a custom DB cluster parameter group.
 rds.CreateDbClusterParameterGroup(DBParameterGroupFamily='aurora-mysql8.0')
    let engine = select(
        "Select an Aurora engine family",
        available_engines.keys().cloned().collect::<Vec<String>>(),
        "Invalid engine selection",
    )?;

    let version = select(
        format!("Select an Aurora engine version for {engine}").as_str(),
        available_engines.get(&engine).cloned().unwrap_or_default(),
        "Invalid engine version selection",
    )?;

    let set_engine = scenario.set_engine(engine.as_str(),
 version.as_str()).await;
    if let Err(error) = set_engine {
        return Err(anyhow!("Could not set engine: {}", error));
    }

    let instance_classes = scenario.get_instance_classes().await;
    match instance_classes {
        Ok(classes) => {
            let instance_class = select(
                format!("Select an Aurora instance class for {engine}").as_str(),
                classes,
                "Invalid instance class selection",
            )?;
            scenario.set_instance_class(Some(instance_class))
        }
        Err(err) => return Err(anyhow!("Failed to get instance classes for
 engine: {err}")),
    }

    Ok(scenario)
}

// Prepare the cluster, creating a custom parameter group overriding some group
 parameters based on user input.
async fn prepare_cluster(scenario: &mut AuroraScenario, warnings: &mut Warnings)
 -> Result<(), ()> {
```

```
   show_parameters(scenario, warnings).await;

   let offset = prompt_number_or_default(warnings, "auto_increment_offset", 5);
   let increment = prompt_number_or_default(warnings,
"auto_increment_increment", 3);

   // Modify both the auto_increment_offset and auto_increment_increment
parameters in one call in the custom parameter group. Set their ParameterValue
fields to a new allowable value. rds.ModifyDbClusterParameterGroup.
   let update_auto_increment = scenario.update_auto_increment(offset,
increment).await;

   if let Err(error) = update_auto_increment {
       warnings.push("Failed to update auto increment", error);
       return Err(());
   }

   // Get and display the updated parameters. Specify Source of 'user' to get
just the modified parameters. rds.DescribeDbClusterParameters(Source='user')
   show_parameters(scenario, warnings).await;

   let username = inquire::Text::new("Username for the database (default
'testuser')")
       .with_default("testuser")
       .with_initial_value("testuser")
       .prompt();

   if let Err(error) = username {
       warnings.push(
           "Failed to get username, using default",
           ScenarioError::with(format!("Error from inquirer: {error}")),
       );
       return Err(());
   }
   let username = username.unwrap();

   let password = inquire::Text::new("Password for the database (minimum 8
characters)")
       .with_validator(|i: &str| {
           if i.len() >= 8 {
               Ok(inquire::validator::Validation::Valid)
           } else {
               Ok(inquire::validator::Validation::Invalid(
                   "Password must be at least 8 characters".into(),
```

```
                ))
            }
        })
        .prompt();

    let password: Option<SecretString> = match password {
        Ok(password) => Some(SecretString::from(password)),
        Err(error) => {
            warnings.push(
                "Failed to get password, using none (and not starting a DB)",
                ScenarioError::with(format!("Error from inquirer: {error}")),
            );
            return Err(());
        }
    };

    scenario.set_login(Some(username), password);

    Ok(())
}

// Start a single instance in the cluster,
async fn run_instance(scenario: &mut AuroraScenario) -> Result<(), ScenarioError>
 {
    // Create an Aurora DB cluster database cluster that contains a MySql
 database and uses the parameter group you created.
    // Create a database instance in the cluster.
    // Wait for DB instance to be ready. Call rds.DescribeDbInstances and check
 for DBInstanceStatus == 'available'.
    scenario.start_cluster_and_instance().await?;

    let connection_string = scenario.connection_string().await?;

    println!("Database ready: {connection_string}",);

    let _ = inquire::Text::new("Use the database with the connection string. When
 you're finished, press enter key to continue.").prompt();

    // Create a snapshot of the DB cluster. rds.CreateDbClusterSnapshot.
    // Wait for the snapshot to create. rds.DescribeDbClusterSnapshots until
 Status == 'available'.
    let snapshot_name = inquire::Text::new("Provide a name for the snapshot")
        .prompt()
        .unwrap_or(String::from("ScenarioRun"));
```

```
    let snapshot = scenario.snapshot(snapshot_name.as_str()).await?;
    println!(
        "Snapshot is available: {}",
        snapshot.db_cluster_snapshot_arn().unwrap_or("Missing ARN")
    );

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), anyhow::Error> {
    tracing_subscriber::fmt::init();
    let sdk_config = aws_config::from_env().load().await;
    let client = Client::new(&sdk_config);
    let rds = RdsClient::new(client);
    let mut scenario = prepare_scenario(rds).await?;

    // At this point, the scenario has things in AWS and needs to get cleaned up.
    let mut warnings = Warnings::new();

    if prepare_cluster(&mut scenario, &mut warnings).await.is_ok() {
        println!("Configured database cluster, starting an instance.");
        if let Err(err) = run_instance(&mut scenario).await {
            warnings.push("Problem running instance", err);
        }
    }

    // Clean up the instance, cluster, and parameter group, waiting for the
 instance and cluster to delete before moving on.
    let clean_up = scenario.clean_up().await;
    if let Err(errors) = clean_up {
        for error in errors {
            warnings.push("Problem cleaning up scenario", error);
        }
    }

    if warnings.is_empty() {
        Ok(())
    } else {
        println!("There were problems running the scenario:");
        println!("{warnings}");
        Err(anyhow!("There were problems running the scenario"))
    }
}
```

```
#[derive(Clone)]
struct U8Validator {}
impl StringValidator for U8Validator {
    fn validate(&self, input: &str) -> Result<inquire::validator::Validation,
 CustomUserError> {
        if input.parse::<u8>().is_err() {
            Ok(inquire::validator::Validation::Invalid(
                "Can't parse input as number".into(),
            ))
        } else {
            Ok(inquire::validator::Validation::Valid)
        }
    }
}

async fn show_parameters(scenario: &AuroraScenario, warnings: &mut Warnings) {
    let parameters = scenario.cluster_parameters().await;

    match parameters {
        Ok(parameters) => {
            println!("Current parameters");
            for parameter in parameters {
                println!("\t{parameter}");
            }
        }
        Err(error) => warnings.push("Could not find cluster parameters", error),
    }
}

fn prompt_number_or_default(warnings: &mut Warnings, name: &str, default: u8) ->
 u8 {
    let input = inquire::Text::new(format!("Updated {name}:").as_str())
        .with_validator(U8Validator {})
        .prompt();

    match input {
        Ok(increment) => match increment.parse::<u8>() {
            Ok(increment) => increment,
            Err(error) => {
                warnings.push(
                    format!("Invalid updated {name} (using {default}
 instead)").as_str(),
                    ScenarioError::with(format!("{error}")),
```

```
                );
                default
            }
        },
        Err(error) => {
            warnings.push(
                format!("Invalid updated {name} (using {default}
 instead)").as_str(),
                ScenarioError::with(format!("{error}")),
            );
            default
        }
    }
}
```

围绕 Amazon RDS 服务的包装器，允许自动对测试进行模拟。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

use aws_sdk_rds::{
    error::SdkError,
    operation::{
        create_db_cluster::{CreateDBClusterError, CreateDbClusterOutput},
        create_db_cluster_parameter_group::CreateDBClusterParameterGroupError,
        create_db_cluster_parameter_group::CreateDbClusterParameterGroupOutput,
        create_db_cluster_snapshot::{CreateDBClusterSnapshotError,
 CreateDbClusterSnapshotOutput},
        create_db_instance::{CreateDBInstanceError, CreateDbInstanceOutput},
        delete_db_cluster::{DeleteDBClusterError, DeleteDbClusterOutput},
        delete_db_cluster_parameter_group::{
            DeleteDBClusterParameterGroupError,
 DeleteDbClusterParameterGroupOutput,
        },
        delete_db_instance::{DeleteDBInstanceError, DeleteDbInstanceOutput},
        describe_db_cluster_endpoints::{
            DescribeDBClusterEndpointsError, DescribeDbClusterEndpointsOutput,
        },
        describe_db_cluster_parameters::{
            DescribeDBClusterParametersError, DescribeDbClusterParametersOutput,
        },
```

```
        describe_db_clusters::{DescribeDBClustersError,
 DescribeDbClustersOutput},
        describe_db_engine_versions::{
            DescribeDBEngineVersionsError, DescribeDbEngineVersionsOutput,
        },
        describe_db_instances::{DescribeDBInstancesError,
 DescribeDbInstancesOutput},

 describe_orderable_db_instance_options::DescribeOrderableDBInstanceOptionsError,
        modify_db_cluster_parameter_group::{
            ModifyDBClusterParameterGroupError,
 ModifyDbClusterParameterGroupOutput,
        },
    },
    types::{OrderableDbInstanceOption, Parameter},
    Client as RdsClient,
};
use secrecy::{ExposeSecret, SecretString};

#[cfg(test)]
use mockall::automock;

#[cfg(test)]
pub use MockRdsImpl as Rds;
#[cfg(not(test))]
pub use RdsImpl as Rds;

pub struct RdsImpl {
    pub inner: RdsClient,
}

#[cfg_attr(test, automock)]
impl RdsImpl {
    pub fn new(inner: RdsClient) -> Self {
        RdsImpl { inner }
    }

    // snippet-start:[rust.aurora.describe_db_engine_versions.wrapper]
    pub async fn describe_db_engine_versions(
        &self,
        engine: &str,
    ) -> Result<DescribeDbEngineVersionsOutput,
 SdkError<DescribeDBEngineVersionsError>> {
        self.inner
```

```
            .describe_db_engine_versions()
            .engine(engine)
            .send()
            .await
    }
    // snippet-end:[rust.aurora.describe_db_engine_versions.wrapper]

    // snippet-start:[rust.aurora.describe_orderable_db_instance_options.wrapper]
    pub async fn describe_orderable_db_instance_options(
        &self,
        engine: &str,
        engine_version: &str,
    ) -> Result<Vec<OrderableDbInstanceOption>,
SdkError<DescribeOrderableDBInstanceOptionsError>>
    {
        self.inner
            .describe_orderable_db_instance_options()
            .engine(engine)
            .engine_version(engine_version)
            .into_paginator()
            .items()
            .send()
            .try_collect()
            .await
    }
    // snippet-end:[rust.aurora.describe_orderable_db_instance_options.wrapper]

    // snippet-start:[rust.aurora.create_db_cluster_parameter_group.wrapper]
    pub async fn create_db_cluster_parameter_group(
        &self,
        name: &str,
        description: &str,
        family: &str,
    ) -> Result<CreateDbClusterParameterGroupOutput,
SdkError<CreateDBClusterParameterGroupError>>
    {
        self.inner
            .create_db_cluster_parameter_group()
            .db_cluster_parameter_group_name(name)
            .description(description)
            .db_parameter_group_family(family)
            .send()
            .await
    }
```

```
    // snippet-end:[rust.aurora.create_db_cluster_parameter_group.wrapper]

    // snippet-start:[rust.aurora.describe_db_clusters.wrapper]
    pub async fn describe_db_clusters(
        &self,
        id: &str,
    ) -> Result<DescribeDbClustersOutput, SdkError<DescribeDBClustersError>> {
        self.inner
            .describe_db_clusters()
            .db_cluster_identifier(id)
            .send()
            .await
    }
    // snippet-end:[rust.aurora.describe_db_clusters.wrapper]

    // snippet-start:[rust.aurora.describe_db_cluster_parameters.wrapper]
    pub async fn describe_db_cluster_parameters(
        &self,
        name: &str,
    ) -> Result<Vec<DescribeDbClusterParametersOutput>,
SdkError<DescribeDBClusterParametersError>>
    {
        self.inner
            .describe_db_cluster_parameters()
            .db_cluster_parameter_group_name(name)
            .into_paginator()
            .send()
            .try_collect()
            .await
    }
    // snippet-end:[rust.aurora.describe_db_cluster_parameters.wrapper]

    // snippet-start:[rust.aurora.modify_db_cluster_parameter_group.wrapper]
    pub async fn modify_db_cluster_parameter_group(
        &self,
        name: &str,
        parameters: Vec<Parameter>,
    ) -> Result<ModifyDbClusterParameterGroupOutput,
SdkError<ModifyDBClusterParameterGroupError>>
    {
        self.inner
            .modify_db_cluster_parameter_group()
            .db_cluster_parameter_group_name(name)
            .set_parameters(Some(parameters))
```

```
            .send()
            .await
    }
    // snippet-end:[rust.aurora.modify_db_cluster_parameter_group.wrapper]

    // snippet-start:[rust.aurora.create_db_cluster.wrapper]
    pub async fn create_db_cluster(
        &self,
        name: &str,
        parameter_group: &str,
        engine: &str,
        version: &str,
        username: &str,
        password: SecretString,
    ) -> Result<CreateDbClusterOutput, SdkError<CreateDBClusterError>> {
        self.inner
            .create_db_cluster()
            .db_cluster_identifier(name)
            .db_cluster_parameter_group_name(parameter_group)
            .engine(engine)
            .engine_version(version)
            .master_username(username)
            .master_user_password(password.expose_secret())
            .send()
            .await
    }
    // snippet-end:[rust.aurora.create_db_cluster.wrapper]

    // snippet-start:[rust.aurora.create_db_instance.wrapper]
    pub async fn create_db_instance(
        &self,
        cluster_name: &str,
        instance_name: &str,
        instance_class: &str,
        engine: &str,
    ) -> Result<CreateDbInstanceOutput, SdkError<CreateDBInstanceError>> {
        self.inner
            .create_db_instance()
            .db_cluster_identifier(cluster_name)
            .db_instance_identifier(instance_name)
            .db_instance_class(instance_class)
            .engine(engine)
            .send()
            .await
```

```
    }
    // snippet-end:[rust.aurora.create_db_instance.wrapper]

    // snippet-start:[rust.aurora.describe_db_instance.wrapper]
    pub async fn describe_db_instance(
        &self,
        instance_identifier: &str,
    ) -> Result<DescribeDbInstancesOutput, SdkError<DescribeDBInstancesError>> {
        self.inner
            .describe_db_instances()
            .db_instance_identifier(instance_identifier)
            .send()
            .await
    }
    // snippet-end:[rust.aurora.describe_db_instance.wrapper]

    // snippet-start:[rust.aurora.create_db_cluster_snapshot.wrapper]
    pub async fn snapshot_cluster(
        &self,
        db_cluster_identifier: &str,
        snapshot_name: &str,
    ) -> Result<CreateDbClusterSnapshotOutput,
SdkError<CreateDBClusterSnapshotError>> {
        self.inner
            .create_db_cluster_snapshot()
            .db_cluster_identifier(db_cluster_identifier)
            .db_cluster_snapshot_identifier(snapshot_name)
            .send()
            .await
    }
    // snippet-end:[rust.aurora.create_db_cluster_snapshot.wrapper]

    // snippet-start:[rust.aurora.describe_db_instances.wrapper]
    pub async fn describe_db_instances(
        &self,
    ) -> Result<DescribeDbInstancesOutput, SdkError<DescribeDBInstancesError>> {
        self.inner.describe_db_instances().send().await
    }
    // snippet-end:[rust.aurora.describe_db_instances.wrapper]

    // snippet-start:[rust.aurora.describe_db_cluster_endpoints.wrapper]
    pub async fn describe_db_cluster_endpoints(
        &self,
        cluster_identifier: &str,
```

```
    ) -> Result<DescribeDbClusterEndpointsOutput,
SdkError<DescribeDBClusterEndpointsError>> {
        self.inner
            .describe_db_cluster_endpoints()
            .db_cluster_identifier(cluster_identifier)
            .send()
            .await
    }
    // snippet-end:[rust.aurora.describe_db_cluster_endpoints.wrapper]

    // snippet-start:[rust.aurora.delete_db_instance.wrapper]
    pub async fn delete_db_instance(
        &self,
        instance_identifier: &str,
    ) -> Result<DeleteDbInstanceOutput, SdkError<DeleteDBInstanceError>> {
        self.inner
            .delete_db_instance()
            .db_instance_identifier(instance_identifier)
            .skip_final_snapshot(true)
            .send()
            .await
    }
    // snippet-end:[rust.aurora.delete_db_instance.wrapper]

    // snippet-start:[rust.aurora.delete_db_cluster.wrapper]
    pub async fn delete_db_cluster(
        &self,
        cluster_identifier: &str,
    ) -> Result<DeleteDbClusterOutput, SdkError<DeleteDBClusterError>> {
        self.inner
            .delete_db_cluster()
            .db_cluster_identifier(cluster_identifier)
            .skip_final_snapshot(true)
            .send()
            .await
    }
    // snippet-end:[rust.aurora.delete_db_cluster.wrapper]

    // snippet-start:[rust.aurora.delete_db_cluster_parameter_group.wrapper]
    pub async fn delete_db_cluster_parameter_group(
        &self,
        name: &str,
    ) -> Result<DeleteDbClusterParameterGroupOutput,
SdkError<DeleteDBClusterParameterGroupError>>
```

```
    {
        self.inner
            .delete_db_cluster_parameter_group()
            .db_cluster_parameter_group_name(name)
            .send()
            .await
    }
    // snippet-end:[rust.aurora.delete_db_cluster_parameter_group.wrapper]
}
```

在这种情况下使用的带依赖项的 Cargo.toml。

```
[package]
name = "aurora-code-examples"
authors = [
  "David Souther <dpsouth@amazon.com>",
]
edition = "2021"
version = "0.1.0"

# See more keys and their definitions at https://doc.rust-lang.org/cargo/
reference/manifest.html

[dependencies]
anyhow = "1.0.75"
assert_matches = "1.5.0"
aws-config = { version = "1.0.1", features = ["behavior-version-latest"] }
aws-smithy-types = { version = "1.0.1" }
aws-smithy-runtime-api = { version = "1.0.1" }
aws-sdk-rds = { version = "1.3.0" }
inquire = "0.6.2"
mockall = "0.11.4"
phf = { version = "0.11.2", features = ["std", "macros"] }
sdk-examples-test-utils = { path = "../../test-utils" }
secrecy = "0.8.0"
tokio = { version = "1.20.1", features = ["full", "test-util"] }
tracing = "0.1.37"
tracing-subscriber = { version = "0.3.15", features = ["env-filter"] }
```

- 有关 API 详细信息，请参阅《AWS SDK for Rust API 参考》中的以下主题。
  - CreateDBCluster

- CreateDBClusterParameterGroup
- CreateDBClusterSnapshot
- CreateDBInstance
- DeleteDBCluster
- DeleteDBClusterParameterGroup
- DeleteDBInstance
- DescribeDBClusterParameterGroups
- DescribeDBClusterParameters
- DescribeDBClusterSnapshots
- DescribeDBClusters
- DescribeDBEngineVersions
- DescribeDBInstances
- DescribeOrderableDBInstanceOptions
- ModifyDBClusterParameterGroup

有关 AWS SDK 开发人员指南和代码示例的完整列表，请参阅 将此服务与 AWS SDK 结合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

使用 AWS SDK 的 Aurora 跨服务示例

以下示例应用程序使用 AWS SDK 将 Aurora 与其他 AWS 服务相组合。每个示例都包含一个指向 GitHub 的链接，其中包含了有关如何设置和运行应用程序的说明。

示例

- 创建借阅图书馆 REST API
- 创建 Aurora Serverless 工作项跟踪器

创建借阅图书馆 REST API

以下代码示例显示如何创建借阅图书馆，其中顾客可以使用由 Amazon Aurora 数据库支持的 REST API 借阅和归还图书。

Python

SDK for Python (Boto3)

展示如何将 适用于 Python (Boto3) 的 AWS SDK 和 Amazon Relational Database Service (Amazon RDS) API 与 AWS Chalice 结合使用，以创建由 Amazon Aurora 数据库支持的 REST API。此 Web 服务是完全无服务器的，代表简单的借阅图书馆，其中顾客可以借阅和归还图书。了解如何：

- 创建和管理无服务器 Aurora 数据库集群。
- 使用 AWS Secrets Manager 管理数据库凭证。
- 实施一个数据存储层，该层使用 Amazon RDS 将数据移入和移出数据库。
- 使用 AWS Chalice 将无服务器 REST API 部署到 Amazon API Gateway 和 AWS Lambda。
- 使用请求软件包向 Web 服务发送请求。

有关完整的源代码以及如何设置和运行的说明，请参阅 GitHub 上的完整示例。

本示例中使用的服务

- API Gateway
- Aurora
- Lambda
- Secrets Manager

有关 AWS SDK 开发人员指南和代码示例的完整列表，请参阅 将此服务与 AWS SDK 结合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

创建 Aurora Serverless 工作项跟踪器

以下代码示例演示了如何创建用于跟踪 Amazon Aurora Serverless 数据库中的工作项，以及使用 Amazon Simple Email Service (Amazon SES) 发送报告的 Web 应用程序。

.NET

适用于 .NET 的 SDK

展示如何使用 适用于 .NET 的 AWS SDK 通过 Amazon Simple Email Service (Amazon SES) 创建 Web 应用程序来跟踪 Amazon Aurora 数据库中的工作项和电子邮件报告。此示例使用由 React.js 构建的前端与 RESTful .NET 后端进行交互。

- 将 React Web 应用程序与 AWS 服务集成。
- 列出、添加、更新和删除 Aurora 表中的项目。
- 使用 Amazon SES 以电子邮件形式发送已筛选工作项的报告。
- 使用随附的 AWS CloudFormation 脚本部署与管理示例资源。

有关完整的源代码以及如何设置和运行的说明，请参阅 GitHub 上的完整示例。

本示例中使用的服务

- Aurora
- Amazon RDS
- Amazon RDS 数据服务
- Amazon SES

C++

SDK for C++

演示了如何创建用于跟踪和报告存储在 Amazon Aurora Serverless 数据库中的工作项的 Web 应用程序。

有关完整源代码以及如何设置可查询 Amazon Aurora Serverless 数据以及供 React 应用程序使用的 C++ REST API 的说明，请参阅 GitHub 上的完整示例。

本示例中使用的服务

- Aurora
- Amazon RDS
- Amazon RDS 数据服务
- Amazon SES

Java

适用于 Java 的 SDK 2.x

展示如何创建 Web 应用程序来跟踪与报告存储与 Amazon RDS 数据库的工作项。

有关完整源代码以及如何设置查询 Amazon Aurora 无服务器数据以及供 React 应用程序使用的 Spring REST API 的说明，请参阅 GitHub 上的完整示例。

有关完整的源代码以及如何设置和运行使用 JDBC API 的示例的说明，请参阅 GitHub 上的完整示例。

本示例中使用的服务

- Aurora
- Amazon RDS
- Amazon RDS 数据服务
- Amazon SES

JavaScript

SDK for JavaScript (v3)

展示了如何使用 适用于 JavaScript 的 AWS SDK (v3) 创建可跟踪 Amazon Aurora 数据库中的工作项，以及使用 Amazon Simple Email Service (Amazon SES) 通过电子邮件发送报告的 Web 应用程序。此示例使用由 React.js 构建的前端与 Express Node.js 后端进行交互。

- 将 React.js Web 应用程序与 AWS 服务 集成。
- 列出、添加以及更新 Aurora 表中的项目。
- 使用 Amazon SES 以电子邮件形式发送已筛选工作项的报告。
- 使用随附的 AWS CloudFormation 脚本部署与管理示例资源。

有关完整的源代码以及如何设置和运行的说明，请参阅 GitHub 上的完整示例。

本示例中使用的服务

- Aurora
- Amazon RDS
- Amazon RDS 数据服务
- Amazon SES

Kotlin

适用于 Kotlin 的 SDK

展示如何创建 Web 应用程序来跟踪与报告存储与 Amazon RDS 数据库的工作项。

有关完整源代码以及如何设置查询 Amazon Aurora 无服务器数据以及供 React 应用程序使用的 Spring REST API 的说明，请参阅 GitHub 上的完整示例。

本示例中使用的服务

- Aurora
- Amazon RDS
- Amazon RDS 数据服务
- Amazon SES

PHP

适用于 PHP 的 SDK

展示了如何使用 适用于 PHP 的 AWS SDK 来创建用于跟踪 Amazon RDS 数据库中的工作项，并通过 Amazon Simple Email Service (Amazon SES) 以电子邮件发送报告的 Web 应用程序。此示例使用由 React.js 构建的前端与 RESTful PHP 后端进行交互。

- 将 React.js Web 应用程序与 AWS 服务集成。
- 列出、添加、更新和删除 Amazon RDS 表中的项目。
- 使用 Amazon SES 以电子邮件发送已筛选工作项的报告。
- 使用随附的 AWS CloudFormation 脚本部署与管理示例资源。

有关完整的源代码以及如何设置和运行的说明，请参阅 GitHub 上的完整示例。

本示例中使用的服务

- Aurora
- Amazon RDS
- Amazon RDS 数据服务
- Amazon SES

Python

SDK for Python (Boto3)

演示了如何使用 适用于 Python (Boto3) 的 AWS SDK 创建可跟踪 Amazon Aurora Serverless 数据库中的工作项，以及使用 Amazon Simple Email Service (Amazon SES) 通过电子邮件发送报告的 REST 服务。此示例使用 Flask Web 框架处理 HTTP 路由，并与 React 网页集成以呈现功能齐全的 Web 应用程序。

- 构建与 AWS 服务 集成的 Flask REST 服务。

- 读取、写入和更新存储在 Aurora Serverless 数据库中的工作项。
- 创建包含数据库凭证的 AWS Secrets Manager 密钥，并使用它来验证对数据库的调用。
- 使用 Amazon SES 发送工作项的电子邮件报告。

有关完整的源代码以及如何设置和运行的说明，请参阅 GitHub 上的完整示例。

本示例中使用的服务

- Aurora
- Amazon RDS
- Amazon RDS 数据服务
- Amazon SES

有关 AWS SDK 开发人员指南和代码示例的完整列表，请参阅 将此服务与 AWS SDK 结合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

Amazon Aurora 的最佳实践

接下来，您可以找到有关使用 Amazon Aurora 数据库集群或向其迁移数据的一般最佳实践和选项的信息。

一些 Amazon Aurora 的最佳实践是特定于特殊数据库引擎的。有关特定于数据库引擎的 Aurora 最佳实践的更多信息，请参阅以下内容。

数据库引擎	最佳实践
Amazon Aurora MySQL	请参阅 Amazon Aurora MySQL 的最佳实践
Amazon Aurora PostgreSQL	请参阅 Amazon Aurora PostgreSQL 的最佳实践

> **Note**
>
> 有关 Aurora 的常见建议，请参阅 来自 Amazon Aurora 的建议。

主题

- Amazon Aurora 的基本操作指导方针
- 数据库实例 RAM 建议
- AWS 数据库驱动程序
- 监控 Amazon Aurora
- 使用数据库参数组和数据库集群参数组
- Amazon Aurora 最佳实践视频

Amazon Aurora 的基本操作指导方针

以下是使用 Amazon Aurora 时每个人都应遵循的基本操作指导方针。Amazon RDS 服务等级协议要求您遵循以下指导方针：

- 监控您的内存、CPU 和存储空间的使用情况。可将 Amazon CloudWatch 设置为在使用模式发生更改或接近部署容量时向您发送通知。这样，您就可以保持系统的性能和可用性。

- 如果您的客户端应用程序正在缓存数据库实例的域名服务 (DNS) 数据，请将生存时间 (TTL) 值设置为小于 30 秒。数据库实例的底层 IP 地址在故障转移后可能会发生变化。因此，如果您的应用程序试图连接到不再使用的 IP 地址，那么缓存 DNS 数据达较长时间可能导致连接失败。如果连接使用读取器端点，并且只读副本实例之一处于维护状态或已删除，那么具有多个只读副本的 Aurora 数据库集群也会遭遇连接失败。
- 测试数据库集群的故障转移，以了解对于您的使用案例而言，该过程需要多长时间。测试故障转移可帮助您确保访问数据库集群的应用程序能够在故障转移后自动连接到新的数据库集群。

数据库实例 RAM 建议

要获得最佳性能，请分配足够多的 RAM，以便您的工作集几乎完全位于内存中。要确定您的工作集是否几乎完全位于内存中，请在 Amazon CloudWatch 中检查以下指标：

- `VolumeReadIOPS` – 此指标测量集群卷中的读取 I/O 操作的平均数量，每隔 5 分钟报告一次。`VolumeReadIOPS` 值应该是一个较小且稳定的值。在某些情况下，您可能发现读取 I/O 处于高峰或高于平常。如果是这样，请调查数据库集群中的数据库实例，以查看哪些数据库实例正在导致 I/O 增加。

  > **Tip**
  >
  > 如果您的 Aurora MySQL 集群使用并行查询，您可能会看到 `VolumeReadIOPS` 值出现增长。并行查询不使用缓冲池。因此，尽管查询速度很快，但这种优化的处理可能会导致读取操作和相关费用的增加。

- `BufferCacheHitRatio` – 该指标测量数据库集群中的数据库实例的缓冲区缓存处理的请求百分比。通过使用该指标，您可以深入了解从内存中处理的数据量。

  高命中率表明您的数据库实例有足够的可用内存。低命中率表示您对这个数据库实例的查询经常转向磁盘。调查工作负载以查看哪些查询正在导致该行为。

在调查工作负载后，如果发现需要更多内存，则考虑将数据库实例类扩展为具有更多 RAM 的类。在执行该操作后，您可以调查前面讨论的指标，并继续根据需要进行扩展。如果您的 Aurora 集群大于 40 TB，请勿使用 db.t2、db.t3 或 db.t4g 实例类。

有关更多信息，请参阅 Amazon Aurora 的 Amazon CloudWatch 指标。

AWS 数据库驱动程序

推荐使用 AWS 驱动程序套件来建立应用程序连接。借助这些驱动程序可显著缩短切换和故障转移时间，并支持使用 AWS Secrets Manager、AWS Identity and Access Management（IAM）和联合身份进行身份验证。AWS 驱动程序依靠监控数据库集群状态和了解集群拓扑，来确定新的写入器。这种方法将切换和故障转移时间缩短到几秒钟，而开源驱动程序的切换和故障转移时间则为几十秒。

随着新服务功能的推出，使用 AWS 驱动程序套件可为这些服务功能提供内置支持。

有关更多信息，请参阅 使用 AWS 驱动程序连接到 Aurora 数据库集群。

监控 Amazon Aurora

Amazon Aurora 提供了各种指标和洞察，您可以监控它们以确定 Aurora 数据库集群的运行状况和性能。您可以使用各种工具（例如 AWS Management Console、AWS CLI 和 CloudWatch API）查看 Aurora 指标。您可以在 Performance Insights 控制面板中查看组合的 Performance Insights 和 CloudWatch 指标，并监控您的数据库实例。为使用此监控视图，必须为您的数据库实例开启 Performance Insights。有关监控视图的信息，请参阅使用性能详情控制面板查看组合指标。

您可以创建特定时间段的性能分析报告，并查看已确定的见解和解决问题的建议。有关更多信息，请参阅在性能详情中创建性能分析报告。

使用数据库参数组和数据库集群参数组

在将参数组更改应用于生产数据库集群之前，我们建议您在测试数据库集群上试验数据库参数组和数据库集群参数组更改。不正确地设置数据库引擎参数可能会产生意外的不利影响，包括性能下降和系统不稳定。

在修改数据库引擎参数时，请务必谨慎，并在修改数据库参数组之前备份数据库集群。有关备份数据库集群的信息，请参阅备份和还原 Amazon Aurora 数据库集群。

Amazon Aurora 最佳实践视频

YouTube 上的 AWS 在线技术谈话频道包括一个视频演示，旨在介绍创建和配置更安全且可用性更高的 Amazon Aurora 数据库集群的最佳实践。请参阅 Amazon Aurora 高可用性的最佳实践。

执行 Amazon Aurora 概念验证

在下文中，您可以找到有关如何为 Aurora 设置和运行概念验证的说明。概念验证是一种调查，执行此调查可了解 Aurora 是否适用于您的应用程序。概念验证可帮助您了解您自己的数据库应用程序环境中的 Aurora 功能，以及 Aurora 与您当前数据库环境相比较的情况。它还可显示移动数据、移植 SQL 代码、调整性能以及适应您当前管理过程所需完成的工作量。

在本主题中，您可以找到运行概念验证所涉高级过程和决策的概述和分步大纲，如下所示。如需详细说明，您可以使用相关链接转到特定主题的完整文档。

Aurora 概念验证概述

执行 Amazon Aurora 概念验证时，可了解将现有数据和 SQL 应用程序移植到 Aurora 需采取的措施。您可使用代表生产环境的大量数据和活动大规模练习使用 Aurora 的重要内容。这样，旨在确信 Aurora 是否能够完美应对您之前的数据库基础设施无法应对的挑战。在概念验证结束时，您可制订可靠的计划进行更大规模的性能基准测试和应用程序测试。此时，您可了解进行生产部署时所需完成的最重要的工作项。

以下最佳实践建议可帮助您避免导致基准测试出现问题的常见错误。但本主题不介绍执行基准测试和优化性能的分步过程。这些过程因工作负载和所用 Aurora 功能而异。有关详细信息，请参阅与性能相关的文档，比如管理 Aurora 数据库集群的性能和扩展、Amazon Aurora MySQL 性能增强、Amazon Aurora PostgreSQL 的性能和扩展 和在 Amazon Aurora 上使用性能详情监控数据库负载。

本主题的信息主要适用于组织写入代码、设计架构所使用的应用程序以及支持 MySQL 和 PostgreSQL 开源数据库引擎的应用程序。如果是测试商业应用程序或使用应用程序框架生成的代码，则可能无法灵活应用这些指导原则。在此类情况下，请咨询您的AWS代表，了解是否有适合您的应用程序类型的 Aurora 最佳实践或案例研究。

1. 确定目标

作为概念验证的一部分评估 Aurora 时，您可以选择要测出哪些测量值以及如何评估练习是否成功。

您必须确保应用程序的所有功能均与 Aurora 兼容。由于 Aurora 主要版本与 MySQL 和 PostgreSQL 的相应主要版本兼容，因此针对这些引擎开发的大多数应用程序也与 Aurora 兼容。不过，您仍须验证每个应用程序的兼容性。

例如，您设置 Aurora 集群时做出的某些配置选择会影响您是否可以或应该使用特定数据库功能。您可以先确定一种最通用的 Aurora 集群，即预配置集群。然后可以确定无服务器或并行查询等专用配置是否有益于您的工作负载。

使用以下问题帮助确定并量化您的目标：

- Aurora 是否支持工作负载的所有功能使用案例？
- 您需要多大的数据集或负载量？ 是否可以调整到该水平？
- 您对特定查询有哪些吞吐量或延迟要求？ 您是否可以达到这些要求？
- 对于计划内或计划外工作负载停机时间，您可接受的最短时间是多久？ 您是否可以达到此目标？
- 哪些运营效率指标是必需的指标？ 您是否可以正常监控这些指标？
- Aurora 是否支持您的特定业务目标，比如降低成本、扩大部署或预置速度？ 您是否可以量化这些目标？
- 您是否可以达到工作负载的所有安全合规要求？

花时间积累有关 Aurora 数据库引擎和平台功能的知识，并参阅本服务文档。注意有助于您达到预期结果的所有功能。其中一个功能可能是工作负载整合，有关说明信息，请参阅 AWS 数据库博客文章如何计划和优化 Amazon Aurora 与 MySQL 的兼容性以实现工作负载整合。另一个功能可能是按需扩展，有关说明信息，请参阅 Amazon Aurora 用户指南中的Amazon Aurora Auto Scaling 与 Aurora 副本结合使用。其他功能可能是性能改善或简化数据库操作等功能。

2. 了解工作负载特性

在预期的使用案例环境下评估 Aurora。Aurora 非常适用于线上事务处理 (OLTP) 工作负载。您还可以对支持实时 OLTP 数据的集群运行报告，而无需预置单独的数据仓库集群。通过核对以下特性，您可以识别您的使用案例是否属于这些类别：

- 高并发性，具有数十、数百或数千个并行客户端。
- 大量低延迟查询（几毫秒到几秒）。
- 实时短事务。
- 高选择性查询模式，基于索引查找。
- 可利用 Aurora 并行查询的分析查询（对于 HTAP）。

影响数据库选择的一个关键因素是数据的速度。高速度包括非常频繁地插入和更新数据。这样的系统可能涉及数千个连接，数十万项并行的数据库读写查询。高速度系统中的查询所影响的行数通常比较少，并且通常是访问同一行中的多个列。

Aurora 经设计可处理高速度数据。根据工作负载，具有单个 r4.16xlarge 数据库实例的 Aurora 集群每秒可以处理 600,000 多个 SELECT 语句。这样的集群可以每秒处理 200,000 INSERT、UPDATE 和 DELETE 语句，具体同样取决于工作负载。Aurora 是一种行存储数据库，非常适用于高容量、高吞吐量和高度并行化的 OLTP 工作负载。

Aurora 还可以对处理 OLTP 工作负载的同一集群运行报告查询。Aurora 支持多达 15 个副本，每个副本平均花费不到主实例 10 - 20 毫秒的时间。分析师可以实时查询 OLTP 数据，无需将数据复制到单独的数据仓库集群。鉴于 Aurora 集群应用并行查询功能，您可以将大量处理、筛选和聚合工作分载到大规模分布式 Aurora 存储子系统。

使用此计划阶段可让自己熟悉 Aurora 的功能、其他AWS服务、AWS Management Console和 AWS CLI。此外，还可以了解这些工具如何与您打算在概念验证中使用的其他工具结合使用。

3. 使用 AWS Management Console或 AWS CLI 进行练习

下一步是使用AWS Management Console或 AWS CLI 进行练习，以熟悉这些工具和 Aurora。

使用 AWS Management Console进行练习

下面主要介绍最初使用 Aurora 数据库集群的活动，以便您熟悉AWS Management Console环境，练习设置和修改 Aurora 集群。如果将 MySQL 兼容版和 PostgreSQL 兼容版数据库引擎与 Amazon RDS 结合使用，可以在使用 Aurora 时根据这类知识执行操作。

通过利用 Aurora 共享存储模式和功能（比如复制和快照），您可以将整个数据库集群视为另一种可随意操作的对象。您可以在概念验证过程中频繁设置、拆分和更改 Aurora 集群的容量。而不会因早期做出的容量、数据库设置和物理数据布局等相关选择受到限制。

要开始使用，请设置一个空 Aurora 集群。最初试验时，可选择 provisioned (预配置) 容量类型和 regional (区域) 位置。

使用 SQL 命令行应用程序等客户端程序连接到该集群。首先，使用集群终端节点连接。连接到该终端节点可执行任何写入操作，比如数据定义语言 (DDL) 语句以及提取、转换、加载 (ETL) 流程。随后在概念验证中，可使用读取器终端节点连接查询密集型会话，该终端节点将查询工作负载分配到集群的多个数据库实例中。

通过添加更多 Aurora 副本扩展集群。有关这些步骤的信息，请参阅使用 Amazon Aurora 进行复制。通过更改 AWS 实例类扩展或缩减数据库实例。了解 Aurora 如何简化这些类型的操作，以便在初次估计的系统容量不正确时，可以随后做出调整，而无需从头开始操作。

创建快照，并将其还原到其他集群中。

检查集群指标以了解一段时间的活动，以及这些指标如何应用于集群中的数据库实例。

刚开始时，熟悉如何通过 AWS Management Console执行这些操作很有用。在了解可以使用 Aurora 执行的操作后，可以继续使用 AWS CLI 自动执行这些操作。在以下部分，您可找到有关概念验证期间这些活动的过程和最佳实践的更多详细信息。

使用 AWS CLI进行练习

我们建议自动执行部署和管理步骤，即使是在概念验证设置时也是如此。为此，请尚未熟悉 AWS CLI 的用户先熟悉相关操作。如果将 MySQL 兼容版和 PostgreSQL 兼容版数据库引擎与 Amazon RDS 结合使用，可以在使用 Aurora 时根据这类知识执行操作。

Aurora 通常包含多个排列在集群中的数据库实例组。因此，许多操作都包括确定与特定集群关联的数据库实例，然后对这些实例循环执行管理操作。

例如，您可以自动执行多个步骤，比如创建 Aurora 集群，然后使用更多实例类扩展这些集群，或使用其他数据库实例扩展这些集群。这样可帮助您在概念验证中重复任何阶段，以及使用不同种类或配置的 Aurora 集群了解假设场景。

了解 AWS CloudFormation 等基础设施部署工具的功能和限制。您可能会发现，在概念验证环境下完成的活动不适用于生产环境。例如，AWS CloudFormation 的修改行为将创建新实例，并删除当前实例，包括其数据。有关此行为的更多详细信息，请参阅 AWS CloudFormation 用户指南中的堆栈资源的更新行为。

4. 创建 Aurora 集群

使用 Aurora，可通过将数据库实例添加到集群，并将数据库实例扩展为更强大的实例类来了解假设场景。您还可以使用不同配置设置创建集群，以并行运行相同的工作负载。使用 Aurora，您可灵活设置、拆分和重新配置数据库集群。因此，在概念验证过程的早期阶段练习这些方法很有用。有关创建 Aurora 集群的常规过程，请参阅创建 Amazon Aurora 数据库集群。

如果可行，请先使用以下设置创建集群。仅在考虑使用某些特定使用案例时才跳过此步骤。例如，如果您的使用案例需要使用一种专用 Aurora 集群，您可以跳过此步骤。或者，如果您需要特定的数据库引擎和版本组合，也可以跳过此步骤。

- 关闭 Easy create (轻松创建)。对于概念验证，建议您了解选择的所有设置，以便随后创建相同或略微不同的集群。
- 使用最新的数据库引擎版本。数据库引擎和版本的这些组合与其它 Aurora 功能具有广泛的兼容性，并且有大量的客户使用生产应用程序。
  - Aurora MySQL 版本 3.x（与 MySQL 8.0 兼容）
  - Aurora PostgreSQL 版本 15.x 或 16.x
- 选择 Dev/Test (开发/测试) 模板。对于概念验证活动，此选择并不重要。
- 对于 DB instance class (数据库实例类)，选择 Memory optimized classes (内存优化类) 和其中一个 xlarge 实例类。您稍后可以向上或向下调整实例类。
- 在 Multi-AZ Deployment (多可用区部署) 下，选择 Create an Aurora Replica or Reader node in a different AZ (在不同可用区中创建 Aurora 副本或读取器节点)。许多极有用的 Aurora 内容都涉及多数据库实例集群。先在新集群中添加至少两个数据库实例是明智之举。对第二个数据库实例使用不同的可用区，有助于测试不同的高可用性场景。
- 在为数据库实例选择名称时，需使用通用命名约定。不能将任何集群数据库实例称为“写入器”，因为其它数据库实例会根据需要充当这些角色。我们建议使用诸如 `clustername-az-serialnumber` 之类的名称，例如 `myprodappdb-a-01`。这些代码段可唯一标识数据库实例及其位置。
- 为 Aurora 集群设置较长的备份保留期。使用较长的保留期，可在长达 35 天的时间内执行时间点恢复 (PITR) 操作。在运行包含 DDL 和数据操作语言 (DML) 语句的测试后，可将数据库重置为已知状态。您还可以在误删除或误更改数据的情况下恢复这些数据。
- 在创建集群时，启用其他恢复、记录和监控功能。开启回溯、性能详情、监控和日志导出下提供的所有选项。启用这些功能后，您可以测试回溯、增强监控或工作负载的 Performance Insights 等功能的适用性。您还可以在概念验证期间轻松调查性能，执行故障排除操作。

5. 设置架构

在 Aurora 集群上，可为您的应用程序设置数据库、表、索引、外键和其他架构对象。如果是从其他 MySQL 兼容版或 PostgreSQL 兼容版数据库系统进行迁移，则要求此阶段的操作简单直接。您可以使用相同的 SQL 语法和命令行或您熟悉的适用于数据库引擎的其他客户端应用程序。

要在您的集群上发送 SQL 语句，请找到其集群终端节点，并提供该值作为连接到客户端应用程序的参数值。您可以在集群详细信息页面的 Connectivity (连接) 选项卡上找到集群终端节点。此集群终端节点是标记为 Writer 的终端节点。其他标记为 Reader 的终端节点表示只读连接，您可向运行报告或其他只读查询的最终用户提供这类连接。有关连接到集群时出现问题的帮助信息，请参阅连接到 Amazon Aurora 数据库集群。

如果是从其他数据库系统移植架构和数据，此时要求做出某些架构更改。这些架构更改要与 Aurora 中的可用 SQL 语法和功能相匹配。此时，您可能要舍弃某些列、约束、触发器或其他架构对象。如果这些对象要求改动 Aurora 兼容性，并且不能帮助您实现概念验证的目标，则这样做特别有用。

如果是从其他数据库系统迁移，且此系统使用的底层引擎与 Aurora 的不同，则需考虑使用 AWS Schema Conversion Tool (AWS SCT) 简化此过程。有关详细信息，请参阅AWS Schema Conversion Tool 用户指南。有关迁移和移植活动的一般详细信息，请参阅 AWS 白皮书《将数据库迁移到 Amazon Aurora》。

在此阶段，您可以评估您的架构设置是否存在效率低下的问题，例如在对策略或分区表等其他表结构建立索引时。在具有多数据库实例和极大工作负载的集群上部署应用程序时，这种效率低下的问题会被放大。请考虑是立即微调这类性能问题，还是在稍后的完整基准测试等活动中微调。

6. 导入数据

在概念验证期间，您需要从以前的数据库系统导入数据或代表示例。如果可行，至少在每个表中设置一些数据。这样，有助于测试所有数据类型和架构功能的兼容性。在练习使用基本 Aurora 功能后，可扩展数据量。在完成概念验证时，您应使用大得足以得出准确结论的数据集测试您的 ETL 工具、查询和全部工作负载。

您可以使用多种方法将物理或逻辑备份数据导入 Aurora。有关详细信息，请根据您在概念验证中使用的数据库引擎参阅将数据迁移到 Amazon Aurora MySQL 数据库集群或将数据迁移到与 PostgreSQL 兼容的 Amazon Aurora。

使用您考虑使用的 ETL 工具和方法试验。了解哪种工具和方法最符合您的需求。您需要考虑吞吐量和灵活性。例如，某些 ETL 工具是执行一次性传输，而其他工具是将原来系统中的数据持续复制到 Aurora。

如果是从 MySQL 兼容版系统迁移到 Aurora MySQL，可以使用本机数据传输工具。如果是从 PostgreSQL 兼容版系统迁移到 Aurora PostgreSQL，同样可应用本机数据传输工具。如果是从其他数据库系统迁移，且此系统使用的底层引擎与 Aurora 的不同，则可以使用 AWS Database Migration Service (AWS DMS) 进行试验。有关 AWS DMS 的详细信息，请参阅 AWS Database Migration Service 用户指南。

有关迁移和移植活动的详细信息，请参阅AWS白皮书的 Aurora 迁移手册。

7. 移植 SQL 代码

试验 SQL 和相关应用程序需要完成不同的工作量，具体取决于不同的案例。具体而言，工作量取决于是从 MySQL 兼容版系统还是 PostgreSQL 兼容版系统，抑或是其他类型的系统迁移。

- 如果是从 RDS for MySQL 或 RDS for PostgreSQL 迁移，则 SQL 更改量很少，以致于您可以使用 Aurora 尝试执行原始 SQL 代码，并手动加入所需更改。
- 同样，如果是从与 MySQL 或 PostgreSQL 兼容的本地数据库迁移，则可以尝试执行原始 SQL 代码，并手动加入更改。
- 如果是来自其他商用数据库，则必需的 SQL 更改量会增大。在这种情况下，需考虑使用 AWS SCT。

在此阶段，您可以评估您的架构设置是否存在效率低下的问题，例如在对策略或分区表等其他表结构建立索引时。请考虑是立即微调这类性能问题，还是在稍后的完整基准测试等活动中微调。

您可以验证应用程序的数据库连接逻辑。为利用 Aurora 分布式处理，您可能需要使用单独的连接来执行读写操作，并使用相对较短的会话来完成查询操作。有关连接的信息，请参阅 9. 连接到 Aurora。

考虑是否必须妥协和取舍才能解决生产数据库中的问题。花时间研究概念验证计划，以改进您的架构设计和查询。为判定是否可以轻松提高性能、降低运营成本和提高可扩展性，可尝试在不同 Aurora 集群上并行运行原始应用程序和修改的应用程序。

有关迁移和移植活动的详细信息，请参阅AWS白皮书的 Aurora 迁移手册。

8. 指定配置设置

作为 Aurora 概念验证练习的一部分，您还可以检查您的数据库配置参数。您可能已针对性能和可扩展性在当前环境下优化了 MySQL 或 PostgreSQL 配置设置。此 Aurora 存储子系统是针对带有高速存储子系统的基于云的分布式环境进行调整和优化的。因此，许多以前的数据库引擎设置都不适用了。我们建议使用默认的 Aurora 配置设置进行初步试验。仅当您遇到性能和可扩展性瓶颈时，才重新应用您当前环境中的设置。如果您有兴趣，可在 AWS 数据库博客上的 Aurora 存储引擎简介中更深入地了解此主题。

Aurora 可让您对特定应用或使用案例轻松重用最佳配置设置。您无需为每个数据库实例编辑单独的配置文件，只需管理分配给整个集群或特定数据库实例的参数集。例如，时区设置适用于集群中的所有数据库实例，而您可以调整每个数据库实例的页面缓存大小设置。

您可先应用默认参数集之一，然后仅对需要微调的参数进行更改。有关使用参数组的详细信息，请参阅 Amazon Aurora 数据库集群和数据库实例参数。有关是否适用于 Aurora 集群的配置设置，请根据您的数据库引擎参阅 Aurora MySQL 配置参数或 Amazon Aurora PostgreSQL 参数。

9. 连接到 Aurora

您会发现，在进行初始架构和数据设置并运行示例查询时，您可以连接到 Aurora 集群中的不同终端节点。使用的终端节点取决于操作是读取（比如 SELECT 语句）还是写入（比如 CREATE 或 INSERT 语句）。在增加 Aurora 集群上的工作负载和使用 Aurora 功能试验时，请务必让应用程序向相应的终端节点分配每个操作。

通过对写入操作使用集群终端节点，您可始终连接到集群中具有读/写功能的数据库实例。默认情况下，Aurora 集群中仅一个数据库实例具有读/写功能。此数据库实例被称为主实例。如果原始主实例不可用，则 Aurora 会激活故障转移机制，并且其他数据库实例会接任主实例。

同样，通过将 SELECT 语句定向到读取器终端节点，可将处理查询的工作分布到集群的各数据库实例中。通过轮询 DNS 解析可将每个读取器连接分配给不同的数据库实例。在只读数据库 Aurora 副本上完成大部分查询工作，可减少主实例的负载，让其能够处理 DDL 和 DML 语句。

使用这些终端节点可减少对硬编码主机名的依赖，并可帮助您的应用程序更快速地从数据库实例故障中恢复。

> **Note**
>
> Aurora 还具有您创建的自定义终端节点。在概念验证期间通常不需要使用这些终端节点。

Aurora 副本会受副本滞后影响，即使该滞后通常是 10 到 20 毫秒。您可以监控复制滞后，并确定该值是否在数据一致性要求的范围内。在某些情况下，读取查询可能要求严格的读取一致性（先写后读一致性）。在这些情况下，您可以继续对其使用集群终端节点，而不是读取器终端节点。

为充分利用 Aurora 功能实现分布式并行执行，您可能需要更改连接逻辑。目的是避免将所有读取请求发送到主实例。只读 Aurora 副本将使用所有相同的数据待命，准备处理 SELECT 语句。对应用逻辑编码可对每种操作使用相应的终端节点。请遵循以下一般准则：

- 避免对所有数据库会话都使用仅有的一个硬编码连接字符串。
- 如果可行，请将写入操作（比如 DDL 和 DML 语句）包括在客户端应用程序代码的函数中。这样，您可以使用特定连接执行不同种类的操作。
- 为查询操作创建单独的函数。Aurora 会将读取器终端节点的各新连接分配给不同的 Aurora 副本，以使读取密集型应用程序的负载均衡。
- 对于涉及多组查询的操作，在完成每组相关查询时，先断开与读取器终端节点的连接，然后再重新连接。如果软件堆栈提供连接池，请使用该功能。将查询定向到不同连接可帮助 Aurora 将读取工作负载分布到集群的各数据库实例中。

有关 Aurora 连接管理和终端节点的一般信息，请参阅连接到 Amazon Aurora 数据库集群。有关此主题的深入分析，请参阅 Aurora MySQL 数据库管理员手册 – 连接管理。

10. 运行工作负载

在准备好架构、数据和配置设置后，可以通过运行工作负载开始练习使用集群。在概念验证中使用反映生产工作负载主要内容的工作负载。我们建议始终使用实际测试和工作负载而非 sysbench 或 TPC-C 等合成基准做出性能相关决策。如果可行，请根据自己的架构、查询模式和用量收集测量值。

只要可行，请复制应用程序运行所依据的实际条件。例如，您通常在与 Aurora 集群相同的AWS区域和相同的 Virtual Private Cloud (VPC) 中的 Amazon EC2 实例中运行应用程序代码。如果您的生产应用程序是在跨多可用区的多个 EC2 实例中运行，请按同样的条件设置概念验证环境。有关AWS区域的更多信息，请参阅 Amazon RDS 用户指南中的区域和可用区。要了解有关 Amazon VPC 服务的更多信息，请参阅 Amazon VPC 用户指南 中的什么是 Amazon VPC？

在验证应用程序的基本功能是否有效并且是否可以通过 Aurora 访问数据后，可以练习使用 Aurora 集群的内容。您可能要尝试使用的某些功能包括具有负载均衡功能的并发连接、并发事务和自动复制。

现在，您应该熟悉数据传输机制了，因此可以使用更大比例的示例数据运行测试。

在此阶段，您可以看到更改配置设置（比如内存限制和连接限制）的效果。您可以重新访问 8. 指定配置设置中所探讨的过程。

您还可以使用创建和还原快照等机制试验。例如，您可以使用不同 AWS 实例类、AWS 副本编号等创建集群。然后在每个集群上，您可以还原相同的快照，其中包含架构和所有数据。有关该循环操作的详细信息，请参阅创建数据库集群快照和从数据库集群快照还原。

11. 衡量性能

此操作范围的最佳实践可确保设置所有适用工具和流程，以便在工作负载操作期间快速隔离异常行为。此外，您还可通过设置这些对象来了解是否可以可靠地确定所有适用原因。

您始终可以通过检查 Monitoring (监控) 选项卡，了解集群的当前状态，或检查一段时间的趋势。此选项卡位于每个 Aurora 集群或数据库实例的控制台详细信息页面中。其中以图表形式显示 Amazon CloudWatch 监控服务中的指标。您可以按名称、数据库实例和时间段筛选这些指标。

要在 Monitoring (监控) 选项卡上使用更多选项，请在集群设置中启用“Enhanced Monitoring (增强监控)”和 Performance Insights。如果在设置集群时未选择这些选项，您也可以以后启用这些选项。

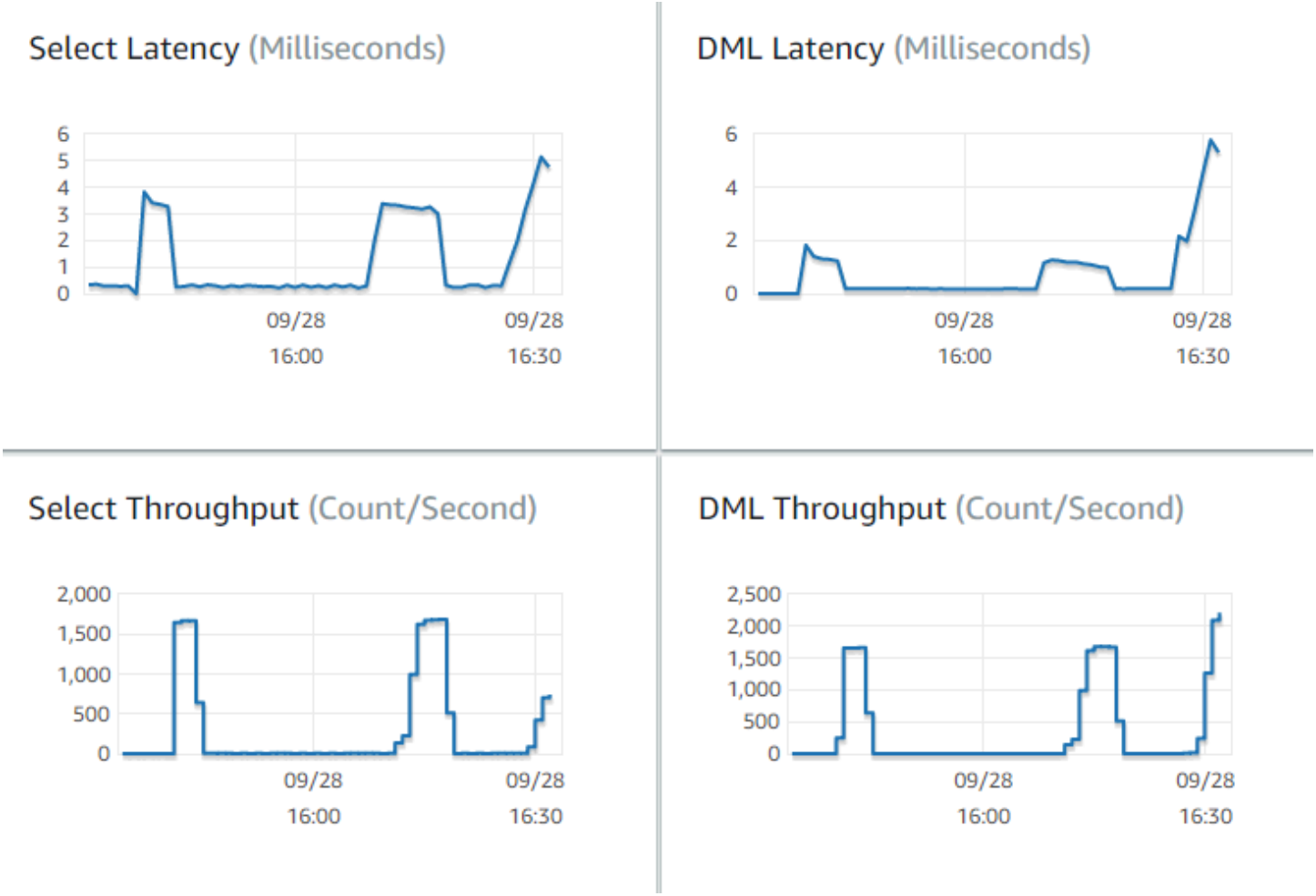
为测量性能，通常需依赖显示全部 Aurora 集群活动的图表。您可以验证 Aurora 副本的负载和响应时间是否相似。您还可以了解读写主实例和只读 Aurora 副本之间的工作是如何划分的。如果数据库实例

之间的负载不均衡，或存在仅影响一个数据库实例的问题，您可以针对该特定实例检查 Monitoring (监控) 选项卡。

在设置环境和实际工作负载以模拟生产应用程序后，可以测量 Aurora 的运行性能。要了解的最重要的问题如下：

- Aurora 每秒处理多少个查询？ 您可以检查 Throughput (吞吐量) 指标，以了解各种操作的数据。
- Aurora 处理给定查询平均需要多长时间？ 您可以检查 Latency (延迟) 指标，以了解各种操作的数据。

要查看吞吐量和延迟指标，请在 Amazon RDS 控制台中查看给定 Aurora 集群的监控选项卡。以下屏幕截图显示了监控选项卡上选择延迟、DML 延迟、选择吞吐量和 DML 吞吐量指标的示例。



如果可以创建这些指标的基准值，请在当前环境下执行此操作。如果不可行，可通过执行等同于生产应用程序的工作负载来构建 Aurora 集群的相关基准。例如，运行并行用户数量和查询数量相当的 Aurora 工作负载。然后观察使用不同实例类、集群大小、配置设置等试验时这些值的变化情况。

如果吞吐量数值低于预期值，可针对工作负载进一步调查影响数据库性能的因素。同样，如果延迟数值高于预期值，也可以进一步调查。为此，需监控数据库服务器的二级指标（CPU、内存等）。您可以了解这些数据库实例是否接近其限制。您还可以了解数据库实例可使用多少附加容量来处理更多并行查询，对更大的表运行查询等。

> **Tip**
>
> 要检测超出预期范围的指标值，请设置 CloudWatch 警报。

在评估理想的 Aurora 集群大小和容量时，您可以找到既能达到应用程序性能峰值又不会超额配置资源的配置。其中一个重要因素是，为 Aurora 集群中的数据库实例找到适宜的大小。首先，选择一个实例大小，其 CPU 和内存容量与您当前生产环境的配置相似。为该实例大小的工作负载收集吞吐量和延迟数值。然后，将实例扩展到更大规模。了解吞吐量数值是否会增长，延迟数值是否会降低。您也可以缩减实例大小，然后了解延迟和吞吐量数值是否保持不变。目的是在可能最小的实例中实现最高的吞吐量和最短的延迟。

> **Tip**
>
> 使用足够的现有容量调整 Aurora 集群和相关数据库实例的大小，以应对突发的不可预测的流量高峰。对于任务关键型数据库，保留至少 20% 的备用 CPU 和内存容量。

确保性能测试运行时间足够长，以便在热稳定状态下测量数据库性能。您可能需要运行工作负载几分钟，甚至几小时才能达到此稳定状态。运行开始时出现一些差异是正常的。出现这样的差异是因为每个 Aurora 副本都会根据所处理的 SELECT 查询预热其缓存。

Aurora 最适合处理涉及多个并行用户和查询的事务型工作负载。为确保以最佳性能驱动足够多的负载，您可运行使用多线程的基准测试，或同时运行多个实例的性能测试。测量使用数百个，甚至数千个并行客户端线程时的性能。模拟您希望在生产环境下使用的并行线程的数量。您还可以使用更多线程执行额外的压力测试，以测量 Aurora 可扩展性。

12. 练习使用 Aurora 高可用性

许多主要的 Aurora 功能都涉及高可用性。这些功能包括自动复制、自动故障转移、自动备份及时间点还原，以及将数据库实例添加到集群的功能。这类功能的安全性与可靠性对任务关键型应用程序极重要。

评估这些功能需要应用特定思维模式。在性能测量等早期活动中，您已观察了系统正常运行时的性能。测试高可用性要求您全面考虑最糟糕情况下的行为。您必须考虑到各种故障，即使很少出现这样的情况。您可以故意引发问题，以确保系统快速准确地恢复正常。

> **Tip**
>
> 对于概念验证，使用相同的AWS实例类设置 Aurora 集群中的所有数据库实例。这样，可在脱机使用数据库实例模拟故障时，试验 Aurora 可用性功能，并且不需要对性能和可扩展性进行过多更改。

我们建议在每个 Aurora 集群中至少使用两个实例。Aurora 集群中的数据库实例最多可跨三个可用区 (AZ)。前两个或前三个数据库实例各自位于不同的 AZ 中。在开始使用更大的集群时，可将数据库实例分布到 AWS 区域的所有 AZ 中。这样可提高容错能力。即使某问题影响到某个 AZ，Aurora 也可以将故障转移到其他 AZ 的数据库实例。如果您运行的集群所包含的实例超过三个，请将这些数据库实例尽可能均匀地分布到三个 AZ 上。

> **Tip**
>
> Aurora 集群存储独立于数据库实例存储。每个 Aurora 集群存储始终跨越三个 AZ。
> 在测试高可用性功能时，可始终在测试集群中使用容量相同的数据库实例。这样可在一个数据库实例接管另一个数据库实例的任务时，避免出现不可预测的性能、延迟等变化。

要了解如何模拟失败条件以测试高可用性功能，请参阅使用错误注入查询测试 Amazon Aurora MySQL。

作为概念验证练习的一部分，其中一个目标是找到理想的数据库实例数量，并为这些数据库实例找到最佳实例类。这样需要平衡高可用性和性能要求。

对于 Aurora，集群中的数据库实例越多，高可用性的优势越大。增加数据库实例，还可提高读取密集型应用程序的可扩展性。Aurora 可以在只读的 Aurora 副本中分发多个 SELECT 连接查询。

而另一方面，限制数据库实例的数量会减少主节点的复制流量。复制流量会消耗网络带宽，这涉及总体性能和可扩展性问题。因此，对于写入密集型 OLTP 应用程序，宁可使用较少数量的大数据库实例，也不要使用大量小数据库实例。

在典型的 Aurora 集群中，一个数据库实例（主实例）处理所有 DDL 和 DML 语句。而其他数据库实例（Aurora 副本）仅处理 SELECT 语句。尽管这些数据库实例的工作量并不完全相同，我们还是建议对

集群中的所有数据库实例使用相同的实例类。这样，在发生故障并且 Aurora 将其中一个只读数据库实例提升为新的主实例时，此主实例的容量才会与之前的容量相同。

如果需要在同一集群中使用不同容量的数据库实例，请为这些数据库实例设置故障转移层。这些故障转移层可确定根据故障转移机制提升 Aurora 副本的顺序。将比其他数据库实例大得多或小得多的数据库实例放到较低的故障转移层。这样可确保提升时最后选择这些数据库实例。

练习使用 Aurora 的数据恢复功能，比如自动执行时间点还原、手动快照和还原以及集群回溯。如果适用，可将快照复制到其他 AWS 区域，还原到其他 AWS 区域以模拟 DR 场景。

调查您所在组织的还原时间目标 (RTO)、还原点目标 (RPO) 和地理冗余的要求。大多数组织将这些项目分类到灾难恢复这一大类下。您可在灾难恢复过程中评估本节介绍的 Aurora 高可用性功能，以确保满足您的 RTO 和 RPO 要求。

13. 后续操作

在概念验证过程成功结束时，您可根据预期工作负载确认 Aurora 是否是适合您的解决方案。在前面的过程中，您已确定 Aurora 在实际操作环境中的工作情况，并根据成功标准对其进行评估。

在构建数据库环境并使用 Aurora 运行后，您可以继续执行更详细的评估步骤，从而实现最终迁移和生产部署。根据个人情况，其他步骤可能包含也可能包含在概念验证过程中。有关迁移和移植活动的详细信息，请参阅AWS白皮书的 Aurora 迁移手册。

在下一步中，需考虑工作负载的相关安全配置，并设法满足生产环境的安全要求。计划要采用的控制措施，以保护对 Aurora 集群主用户凭证的访问权限。定义数据库用户的角色和责任，以控制对 Aurora 集群中存储数据的访问权限。考虑数据库访问应用程序、脚本和第三方工具或服务的要求。了解AWS服务和功能，比如 AWS Secrets Manager 和 AWS Identity and Access Management（IAM）身份验证。

现在，您应该了解使用 Aurora 运行基准测试的过程和最佳实践。您可能发现需要执行更多性能优化操作。有关详细信息，请参阅管理 Aurora 数据库集群的性能和扩展、Amazon Aurora MySQL 性能增强、Amazon Aurora PostgreSQL 的性能和扩展 和在 Amazon Aurora 上使用性能详情监控数据库负载。如果执行更多优化操作，请确保熟悉您在概念验证期间收集的指标。对于下一步，您可能要使用不同的配置设置、数据库引擎和数据库版本选项创建新集群。或者，您可能要创建多种专用的 Aurora 集群来满足特定使用案例的需求。

例如，您可以针对混合事务/分析处理 (HTAP) 应用程序探索 Aurora 并行查询集群。如果广泛的地理分布对灾难恢复至关重要或者可最大程度缩短延迟，您可探索 Aurora 全局数据库。如果工作负载是间歇性的，或者您是在开发/测试场景中使用 Aurora，则可探索 Aurora Serverless 集群。

此外，生产集群也可能需要处理大量传入连接。要了解这些技术，请参阅AWS白皮书 Aurora MySQL 数据库管理员手册 - 连接管理。

如果在概念验证后确定您的使用案例不适合使用 Aurora，请考虑使用以下AWS服务：

- 对于纯粹用于分析的使用案例，工作负载受益于列式存储格式和其他更适合 OLAP 工作负载的功能。解决此类使用案例的 AWS 服务包括：
  - Amazon Redshift
  - Amazon EMR
  - Amazon Athena
- 许多工作负载都可从 Aurora 与以下一种或多种服务的组合中获益。您可以使用以下服务在这些服务之间迁移数据：
  - AWS Glue
  - AWS DMS
  - 从 Amazon S3 导入，如 Amazon Aurora 用户指南所述
  - 导出到 Amazon S3，如 Amazon Aurora 用户指南所述
  - 许多其他热门 ETL 工具

Amazon Aurora 中的安全性

AWS 十分重视云安全性。作为 AWS 客户，您将从专为满足大多数安全敏感型组织的要求而打造的数据中心和网络架构中受益。

安全性是 AWS 和您的共同责任。责任共担模式将其描述为云的安全性和云中的安全性：

- 云的安全性 – AWS负责保护在AWS云中运行AWS服务的基础架构。AWS还向您提供可安全使用的服务。作为 AWS 合规性计划的一部分，第三方审核人员将定期测试和验证安全性的有效性。要了解适用于 Amazon Aurora（Aurora）的合规性计划，请参阅 AWS 按合规性计划提供的范围内服务。
- 云中的安全性：您的责任由您使用的 AWS 服务决定。您还需要对其他因素负责，包括您的数据的敏感性、您组织的要求以及适用的法律法规。

此文档将帮助您了解如何在使用 Amazon Aurora 时应用责任共担模式。以下主题说明如何配置 Amazon Aurora 以实现您的安全性和合规性目标。您还将了解如何使用其他 AWS 服务来帮助您监控和保护 Amazon Aurora 资源。

您可以管理对数据库集群上的 Amazon Aurora 资源和数据库的访问。用来管理访问的方法取决于用户需要对 Amazon Aurora 执行的任务类型：

- 在基于 Amazon VPC 服务的虚拟私有云（VPC）中运行数据库集群以获得尽可能大的网络访问控制能力。有关在 VPC 中创建数据库集群的更多信息，请参阅 Amazon VPC 和 Amazon Aurora。
- 使用 AWS Identity and Access Management（IAM）策略分配权限，来决定谁可以管理 Amazon Aurora 资源。例如，您可以使用 IAM 确定哪些用户可以创建、描述、修改和删除数据库集群、为资源添加标签或修改安全组。

  要查看 IAM 策略示例，请参阅 Amazon Aurora 的基于身份的策略示例。
- 使用安全组可以控制哪些 IP 地址或 Amazon EC2 实例可连接到数据库集群上的数据库。首次创建数据库集群时，除非通过关联安全组指定的规则进行访问，否则实例防火墙会阻止任何数据库访问。
- 将安全套接字层（SSL）或传输层安全性协议（TLS）连接用于运行 Aurora MySQL 或 Aurora PostgreSQL 的数据库集群。有关将 SSL/TLS 用于数据库集群的更多信息，请参阅 使用 SSL/TLS 加密与数据库集群的连接。
- 使用 Amazon Aurora 加密来保护您的数据库数据库集群和静态快照。Amazon Aurora 加密使用行业标准 AES-256 加密算法，来对托管您的数据库集群的服务器上的数据进行加密。有关更多信息，请参阅 加密 Amazon Aurora 资源。

- 使用数据库引擎的安全功能，控制哪些用户可以登录数据库集群上的数据库。这些功能就像本地网络上的数据库一样工作。

  有关 Aurora MySQL 安全性的信息，请参阅 使用 Amazon Aurora MySQL 实现高安全性。有关 Aurora PostgreSQL 安全性的信息，请参阅 使用 Amazon Aurora PostgreSQL 实现高安全性。

Aurora 是托管式数据库服务 Amazon Relational Database Service（Amazon RDS）的一部分。Amazon RDS 是一项 Web 服务，让用户能够在云中更轻松地设置、操作和扩展关系数据库。如果您还不熟悉 Amazon RDS，请参阅 Amazon RDS 用户指南。

Aurora 包括一个高性能的存储子系统。已自定义其 MySQL 和 PostgreSQL 兼容数据库引擎以利用该快速分布式存储。Aurora 还会自动执行和标准化数据库集群和复制，这通常是数据库配置和管理方面的最大问题。

对于 Amazon RDS 和 Aurora，您可以编程方式访问 RDS API，并且可以使用 AWS CLI 以交互方式访问 RDS API。一些 RDS API 操作和 AWS CLI 命令适用于 Amazon RDS 和 Aurora，而其他一些适用于 Amazon RDS 或 Aurora。有关 RDS API 操作的信息，请参阅 Amazon RDS API 参考。有关 AWS CLI 的更多信息，请参阅适用于 Amazon RDS 的 AWS Command Line Interface 参考。

> **Note**
>
> 您必须仅为您的使用案例配置安全性。您无需为 Amazon Aurora 管理的过程配置安全访问。这些过程包括创建备份、自动失效转移和其他过程。

有关管理对 Amazon Aurora 资源和您的数据库集群上的数据库的访问的更多信息，请参阅以下主题。

主题

- Amazon Aurora 的数据库身份验证
- 使用 Amazon Aurora 和 AWS Secrets Manager 管理密码
- Amazon RDS 中的数据保护
- Amazon Aurora 的 Identity and Access Management
- Amazon Aurora 中的日志记录和监控
- Amazon Aurora 的合规性验证
- Amazon Aurora 中的弹性
- Amazon Aurora 中的基础设施安全性

- Amazon RDS API 和接口 VPC 终端节点 (AWS PrivateLink)
- Amazon Aurora 的安全最佳实践
- 使用安全组控制访问权限
- 主用户账户权限
- 将服务相关角色用于 Amazon Aurora
- Amazon VPC 和 Amazon Aurora

Amazon Aurora 的数据库身份验证

Amazon Aurora 支持多种对数据库用户进行身份验证的方法。

默认情况下，密码身份验证适用于所有数据库集群 对于 Aurora MySQL 和 Aurora PostgreSQL，您还可以为同一个数据库集群添加 IAM 数据库身份验证和 Kerberos 身份验证之一或这两者。

密码、Kerberos 和 IAM 数据库身份验证使用不同的方法对数据库进行身份验证。因此，特定用户只能使用一种身份验证方法登录到数据库。

对于 PostgreSQL，仅为特定数据库的用户使用以下角色设置之一：

- 要使用 IAM 数据库身份验证，请为用户分配 `rds_iam` 角色。
- 要使用 Kerberos 数据库身份验证，请为用户分配 `rds_ad` 角色。
- 要使用密码身份验证，请勿为用户分配 `rds_iam` 或 `rds_ad` 角色。

不要通过嵌套的授予访问权限直接或间接地将 `rds_iam` 和 `rds_ad` 角色分配给 PostgreSQL 数据库的用户。如果将 `rds_iam` 角色添加到主用户，IAM 身份验证优先于密码身份验证，因此主用户必须以 IAM 用户身份登录。

> **Important**
>
> 我们强烈建议不要直接在应用程序中使用主用户。请遵守使用数据库用户的最佳实践，按照您的应用程序所需的最少权限创建用户。

主题

- 密码验证
- IAM 数据库身份验证

- Kerberos 身份验证

密码验证

使用密码身份验证，数据库将执行用户账户的所有管理。您可以使用 CREATE USER 等 SQL 语句创建用户，其中包含数据库引擎指定密码所需的适当子句。例如，在 MySQL 中，语句为 CREATE USER ## IDENTIFIED BY ##，而在 PostgreSQL 中，语句为 CREATE USER ## WITH PASSWORD ##。

使用密码身份验证，数据库可控制和验证用户账户。如果数据库引擎具有强大的密码管理功能，那么可以增强安全性。当您拥有较小的用户社区时，使用密码身份验证可能更易于管理数据库身份验证。因为在这种情况下会生成明文密码，因此与 AWS Secrets Manager 集成可以增强安全性。

有关将 Secrets Manager 与 Amazon Aurora 结合使用的信息，请参阅《AWS Secrets Manager User Guide》中的 Creating a basic secret 和 Rotating secrets for supported Amazon RDS databases。有关以编程方式在自定义应用程序中检索密钥的信息，请参阅 AWS Secrets Manager 用户指南中的检索密钥值。

IAM 数据库身份验证

您可以使用 AWS Identity and Access Management（IAM）数据库身份验证对数据库集群进行身份验证。利用此身份验证方法，您在连接到数据库集群时将无需使用密码。而是使用身份验证令牌。

有关 IAM 数据库身份验证的更多信息，包括特定数据库引擎的可用性信息，请参阅 IAM 数据库身份验证。

Kerberos 身份验证

Amazon Aurora 支持使用 Kerberos 和 Microsoft Active Directory 对数据库用户进行外部身份验证。Kerberos 是一种网络身份验证协议，它使用票证和对称密钥加密，而不再需要通过网络传输密码。Kerberos 已内置到 Active Directory 中，用于在网络资源（如数据库）中对用户进行身份验证。

Amazon Aurora 支持 Kerberos 和 Active Directory，从而为数据库用户提供单一登录和集中身份验证的好处。您可以将用户凭证保存在 Active Directory 中。Active Directory 提供了一个集中位置，以存储和管理多个数据库集群的凭证。

要使用来自自行管理的 Active Directory 的凭证，您需要针对数据库集群加入的 Microsoft Active Directory 设置与 AWS Directory Service 的信任关系。

Aurora PostgreSQL 和 Aurora MySQL 通过全林性身份验证或选择性身份验证，支持单向和双向林信任关系。

在某些情况下，可以通过外部信任关系配置 Kerberos 身份验证。这要求自行管理的 Active Directory 具有更多设置。这包括但不限于 Kerberos Forest Search Order。

Aurora 支持对 Aurora MySQL 和 Aurora PostgreSQL 数据库集群进行 Kerberos 身份验证。有关更多信息，请参阅对 Aurora MySQL 使用 Kerberos 身份验证和在 Aurora PostgreSQL 中使用 Kerberos 身份验证：

使用 Amazon Aurora 和 AWS Secrets Manager 管理密码

Amazon Aurora 集成了 Secrets Manager，以管理您的数据库集群的主用户密码。

主题

- 区域和版本可用性
- Secrets Manager 与 Amazon Aurora 集成的限制
- 使用 AWS Secrets Manager 管理主用户密码的概述
- 使用 Secrets Manager 管理主用户密码的优势
- Secrets Manager 集成所需的权限
- 强制 Aurora 在 AWS Secrets Manager 中管理主用户密码
- 使用 Secrets Manager 管理数据库集群的主用户密码
- 轮换数据库集群的主用户密码密钥
- 查看有关数据库集群的密钥的详细信息

区域和版本可用性

功能可用性和支持因每个数据库引擎的特定版本以及 AWS 区域而异。有关 Secrets Manager 与 Amazon Aurora 集成的版本和区域可用性的更多信息，请参阅 支持 Secrets Manager 集成的区域和 Aurora 数据库引擎。

Secrets Manager 与 Amazon Aurora 集成的限制

以下功能不支持使用 Secrets Manager 管理主用户密码：

- Amazon RDS 蓝绿部署
- 属于 Aurora Global Database 的数据库集群
- Aurora Serverless v1 数据库集群
- Aurora 只读副本

使用 AWS Secrets Manager 管理主用户密码的概述

借助 AWS Secrets Manager，您可以将代码中的硬编码凭证（包括数据库密码）替换为对 Secrets Manager 的 API 调用，从而以编程方式检索密钥。有关 Secrets Manager 的更多信息，请参阅《AWS Secrets Manager 用户指南》https://docs.aws.amazon.com/secretsmanager/latest/userguide/。

当您在 Secrets Manager 中存储数据库密钥时，您的 AWS 账户会产生费用。有关定价的信息，请参阅 AWS Secrets Manager 定价。

在执行以下操作之一时，您可以指定 Aurora 在 Secrets Manager 中管理 Amazon Aurora 数据库集群的主用户密码：

- 创建数据库集群
- 修改数据库集群
- 从 Amazon S3 还原数据库集群（仅限 Aurora MySQL）

当您指定 Aurora 在 Secrets Manager 中管理主用户密码时，Aurora 会生成密码并将其存储在 Secrets Manager 中。您可以直接与密钥交互以检索主用户的凭证。您还可以指定客户托管密钥来加密密钥，或者使用 Secrets Manager 提供的 KMS 密钥。

Aurora 管理密钥的设置，默认情况下每七天轮换一次密钥。您可以修改某些设置，例如轮换计划。如果您删除在 Secrets Manager 中管理密钥的数据库集群，则该密钥及其关联的元数据也会被删除。

要使用密钥中的凭证连接到数据库集群，您可以从 Secrets Manager 检索密钥。有关更多信息，请参阅《AWS Secrets Manager 用户指南》中的从 AWS Secrets Manager 中检索密钥和使用 AWS Secrets Manager 密钥中的凭证连接到 SQL 数据库。

使用 Secrets Manager 管理主用户密码的优势

使用 Secrets Manager 管理 Aurora 主用户密码具有以下优势：

- Aurora 会自动生成数据库凭证。
- Aurora 会自动在 AWS Secrets Manager 中存储和管理数据库凭证。
- Aurora 定期轮换数据库凭证，而无需更改应用程序。
- Secrets Manager 保护数据库凭证免受人员访问和纯文本视图的影响。
- Secrets Manager 允许在密钥中检索数据库凭证以进行数据库连接。
- Secrets Manager 允许使用 IAM 细粒度控制对密钥中的数据库凭证的访问权限。
- 您可以选择使用不同的 KMS 密钥将数据库加密与凭证加密分开。
- 您可以省去手动管理和轮换数据库凭证。
- 您可以使用 AWS CloudTrail 和 Amazon CloudWatch 轻松监控数据库凭证。

有关 Secrets Manager 的优势的更多信息，请参阅 AWS Secrets Manager 用户指南。

Secrets Manager 集成所需的权限

用户必须拥有所需的权限才能执行与 Secrets Manager 集成相关的操作。您可以创建 IAM policy，以便授予权限对所需的指定资源执行特定的 API 操作。然后，可以将这些策略附加到需要这些权限的 IAM 权限集或角色。有关更多信息，请参阅 Amazon Aurora 的 Identity and Access Management。

对于创建、修改或还原操作，指定 Aurora 在 Secrets Manager 中管理主用户密码的用户必须具有执行以下操作的权限：

- `kms:DescribeKey`
- `secretsmanager:CreateSecret`
- `secretsmanager:TagResource`

需要 `kms:DescribeKey` 权限才能访问 `MasterUserSecretKmsKeyId` 的客户托管密钥并描述 `aws/secretsmanager`。

对于创建、修改或还原操作，指定用于在 Secrets Manager 中加密密钥的客户托管密钥的用户必须具有执行以下操作的权限：

- `kms:Decrypt`
- `kms:GenerateDataKey`
- `kms:CreateGrant`

对于修改操作，在 Secrets Manager 中轮换主用户密码的用户必须具有执行以下操作的权限：

- `secretsmanager:RotateSecret`

强制 Aurora 在 AWS Secrets Manager 中管理主用户密码

您可以使用 IAM 条件密钥强制 Aurora 在 AWS Secrets Manager 中管理主用户密码。除非主用户密码由 Aurora 在 Secrets Manager 中进行管理，否则以下策略不支持用户创建或还原数据库实例或数据库集群。

JSON

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": ["rds:CreateDBInstance", "rds:CreateDBCluster",
 "rds:RestoreDBInstanceFromS3", "rds:RestoreDBClusterFromS3"],
            "Resource": "*",
            "Condition": {
                "Bool": {
                    "rds:ManageMasterUserPassword": false
                }
            }
        }
    ]
}
```

> **Note**
>
> 此策略在创建时在 AWS Secrets Manager 中强制执行密码管理。但是，您仍然可以禁用 Secrets Manager 集成，并通过修改集群手动设置主密码。
> 为防止这种情况，请在策略的操作块中包括 `rds:ModifyDBInstance`、`rds:ModifyDBCluster`。请注意，这可以防止用户对未启用 Secrets Manager 集成的现有集群进行任何进一步修改。

有关在 IAM 策略中使用条件密钥的更多信息，请参阅 Aurora 的策略条件键 和 示例策略：使用条件键。

使用 Secrets Manager 管理数据库集群的主用户密码

执行以下操作时，可以配置 Aurora 在 Secrets Manager 中管理主用户密码：

- 创建 Amazon Aurora 数据库集群
- 修改 Amazon Aurora 数据库集群
- 将数据从外部 MySQL 数据库迁移到 Amazon Aurora MySQL 数据库集群

您可以使用 RDS 控制台、AWS CLI 或 RDS API 来执行这些操作。

控制台

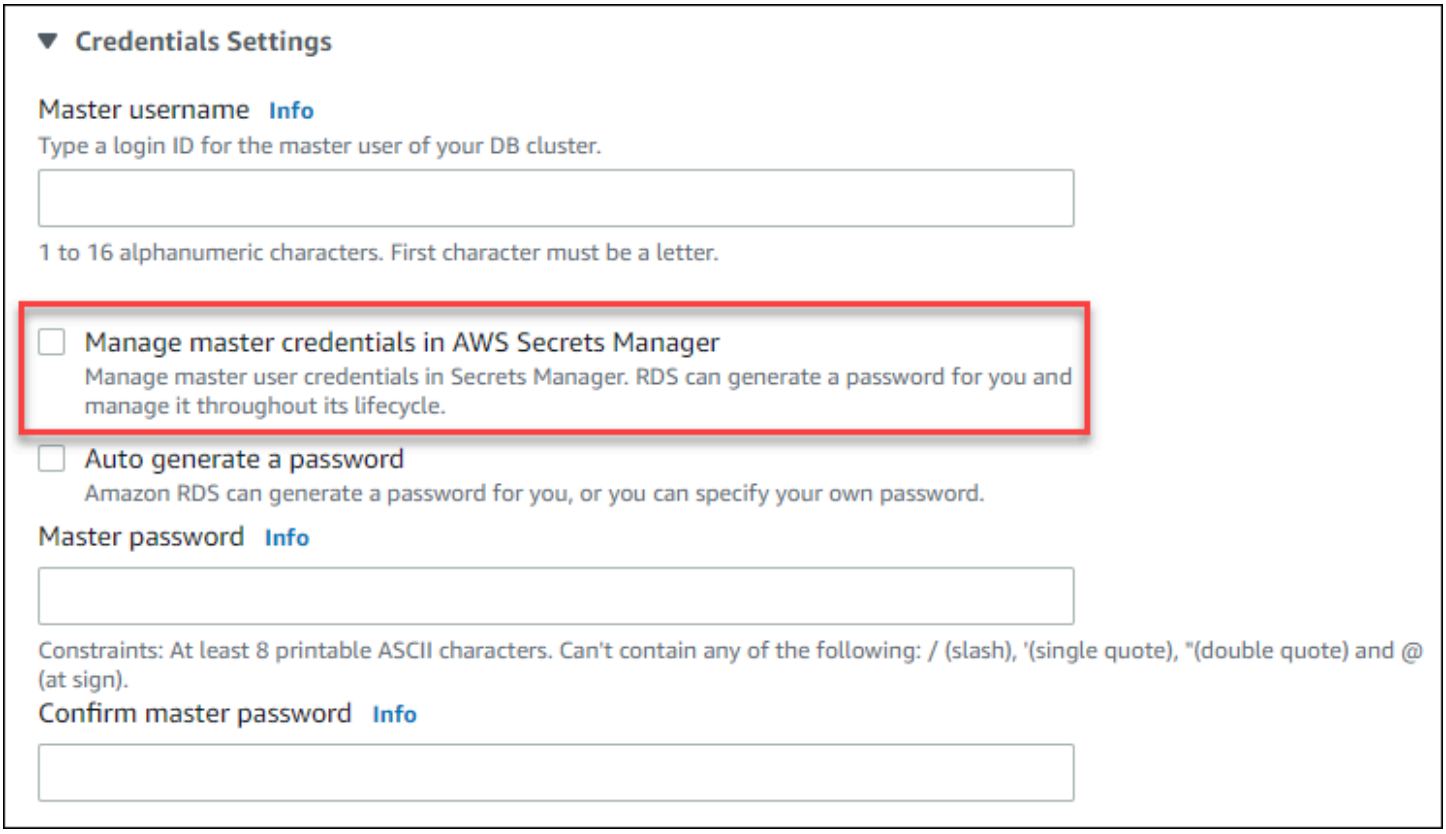
按照说明使用 RDS 控制台创建或修改数据库集群：

- 创建数据库集群
- 修改数据库集群中的数据库实例

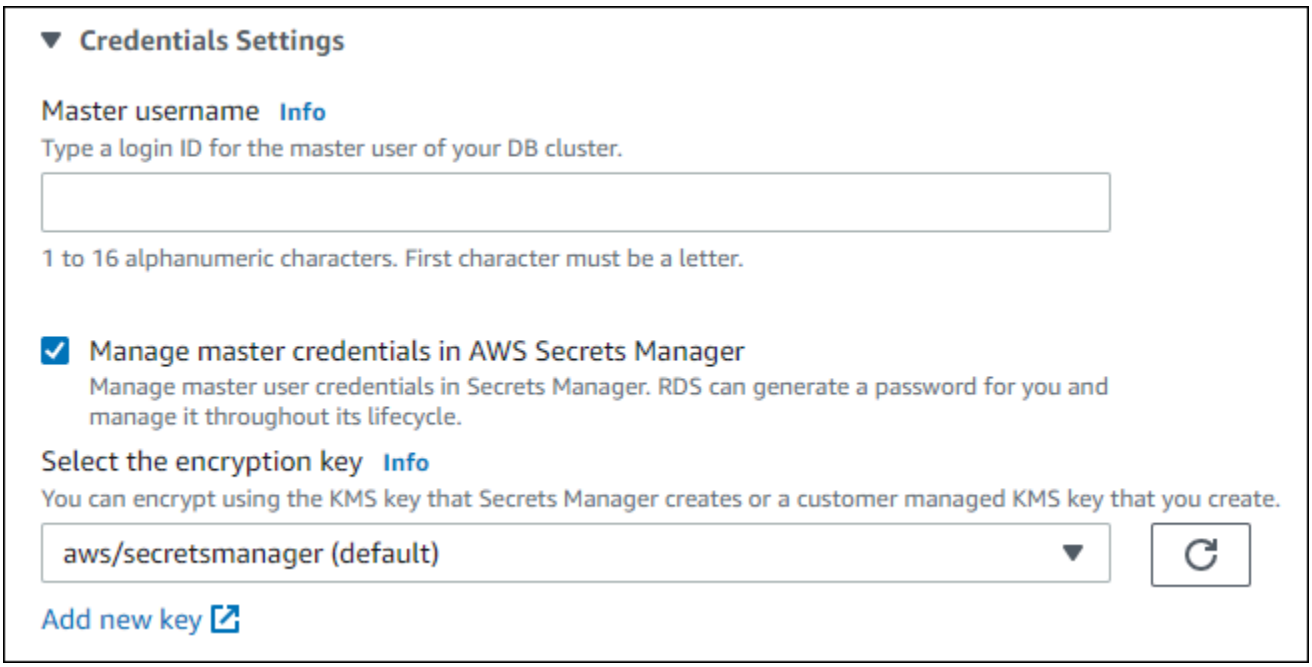
  在 RDS 控制台中，您可以修改任何数据库实例，为整个数据库集群指定主用户密码管理设置。
- 从 Amazon S3 存储桶还原 Amazon Aurora MySQL 数据库集群

当您使用 RDS 控制台执行其中一项操作时，可以在 Secrets Manager 中指定主用户密码由 Aurora 管理。要在创建或还原数据库集群时执行此操作，请在凭证设置中选择在 AWS Secrets Manager 中管理主凭证。修改数据库集群时，请在设置中选择在 AWS Secrets Manager 中管理主凭证。

下图是在创建或还原数据库集群时在 AWS Secrets Manager 中管理主凭证的示例。



当您选择此选项时，Aurora 会生成主用户密码并在其整个生命周期中在 Secrets Manager 中对其进行管理。



您可以选择使用 Secrets Manager 提供的 KMS 密钥或您创建的客户托管密钥对密钥进行加密。在 Aurora 管理数据库集群的数据库凭证后，您无法更改用于加密密钥的 KMS 密钥。

您可以选择其他设置来满足您的要求。

有关创建数据库集群时的可用设置的更多信息，请参阅 Aurora 数据库集群的设置。有关修改数据库集群时的可用设置的更多信息，请参阅 Amazon Aurora 设置。

AWS CLI

要指定 Aurora 在 Secrets Manager 中管理主用户密码，请在以下命令之一中指定 `--manage-master-user-password` 选项：

- create-db-cluster
- modify-db-cluster
- restore-db-cluster-from-s3

当您在这些命令中指定 `--manage-master-user-password` 选项时，Aurora 会生成主用户密码，并在其整个生命周期中在 Secrets Manager 中对其进行管理。

要加密密钥，您可以指定客户托管密钥或使用 Secrets Manager 提供的默认 KMS 密钥。使用 `--master-user-secret-kms-key-id` 选项指定客户托管密钥。AWS KMS 密钥标识符是密钥 ARN、密钥 ID、别名 ARN 或者 KMS 密钥的别名。要使用不同 AWS 账户中的密钥，请指定密钥 ARN 或别名 ARN。在 Aurora 管理数据库集群的数据库凭证后，您无法更改用于加密密钥的 KMS 密钥。

您可以选择其他设置来满足您的要求。

有关创建数据库集群时的可用设置的更多信息，请参阅 Aurora 数据库集群的设置。有关修改数据库集群时的可用设置的更多信息，请参阅 Amazon Aurora 设置。

此示例创建一个数据库集群，并指定 Aurora 在 Secrets Manager 中管理密码。此密钥使用 Secrets Manager 提供的 KMS 密钥进行加密。

Example

对于 Linux、macOS 或 Unix：

```
aws rds create-db-cluster \
     --db-cluster-identifier sample-cluster \
     --engine aurora-mysql \
     --engine-version 8.0 \
     --master-username admin \
     --manage-master-user-password
```

对于 Windows：

```
aws rds create-db-cluster ^
     --db-cluster-identifier sample-cluster ^
     --engine aurora-mysql ^
     --engine-version 8.0 ^
     --master-username admin ^
     --manage-master-user-password
```

RDS API

要指定 Aurora 在 Secrets Manager 中管理主用户密码，请在以下操作之一中将 `ManageMasterUserPassword` 参数设置为 `true`：

- CreateDBCluster
- ModifyDBCluster
- RestoreDBClusterFromS3

当您在其中一个操作中将 `ManageMasterUserPassword` 参数设置为 `true` 时，Aurora 会生成主用户密码并在其整个生命周期中在 Secrets Manager 中对其进行管理。

要加密密钥，您可以指定客户托管式密钥或使用 Secrets Manager 提供的原定设置 KMS 密钥。使用 `MasterUserSecretKmsKeyId` 参数指定客户托管密钥。AWS KMS 密钥标识符是密钥 ARN、密钥 ID、别名 ARN 或者 KMS 密钥的别名。要使用不同 AWS 账户中的密钥，请指定密钥 ARN 或别名 ARN。在 Aurora 管理数据库集群的数据库凭证后，您无法更改用于加密密钥的 KMS 密钥。

轮换数据库集群的主用户密码密钥

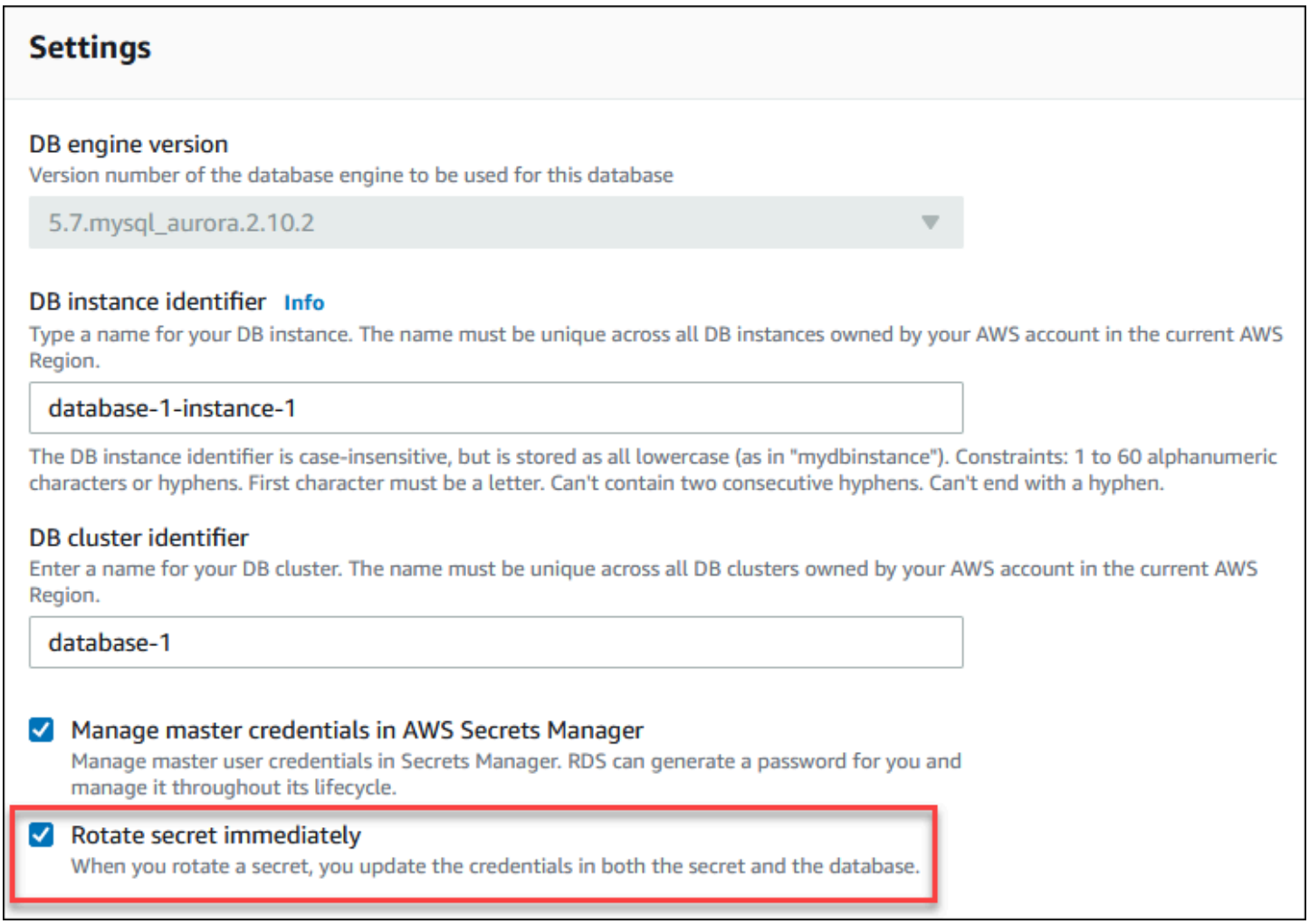
当 Aurora 轮换主用户密码密钥时，Secrets Manager 会为现有密钥生成一个新的密钥版本。密钥的新版本包含新的主用户密码。Aurora 更改数据库集群的主用户密码，以匹配新密钥版本的密码。

您可以立即轮换密钥，而不必等待计划的轮换。要在 Secrets Manager 中轮换主用户密码密钥，请修改数据库集群。有关修改数据库集群的信息，请参阅 修改 Amazon Aurora 数据库集群。

您可以使用 RDS 控制台、AWS CLI 或 RDS API 立即轮换主用户密码密钥。新密码长度始终为 28 个字符，包含至少一个大写和小写字母、一个数字和一个标点符号。

控制台

要使用 RDS 控制台轮换主用户密码密钥，请修改数据库集群并在 Settings（设置）中选择 Rotate secret immediately（立即轮换密钥）。



按照使用控制台、CLI 和 API 修改数据库集群中的说明使用 RDS 控制台修改数据库集群。您必须在确认页面上选择 Apply immediately（立即应用）。

AWS CLI

要使用 AWS CLI 轮换主用户密码密钥，请使用 modify-db-cluster 命令并指定 `--rotate-master-user-password` 选项。轮换主密码时必须指定 `--apply-immediately` 选项。

此示例轮换主用户密码密钥。

Example

对于 Linux、macOS 或 Unix：

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
```

```
    --rotate-master-user-password \
    --apply-immediately
```

对于 Windows：

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --rotate-master-user-password ^
    --apply-immediately
```

RDS API

您可以使用 ModifyDBCluster 操作并将 `RotateMasterUserPassword` 参数设置为 `true` 来轮换主用户密码密钥。轮换主密码时，必须将 `ApplyImmediately` 参数设置为 `true`。

查看有关数据库集群的密钥的详细信息

您可以使用控制台（https://console.aws.amazon.com/secretsmanager/）或 AWS CLI（get-secret-value Secrets Manager 命令）检索您的密钥。

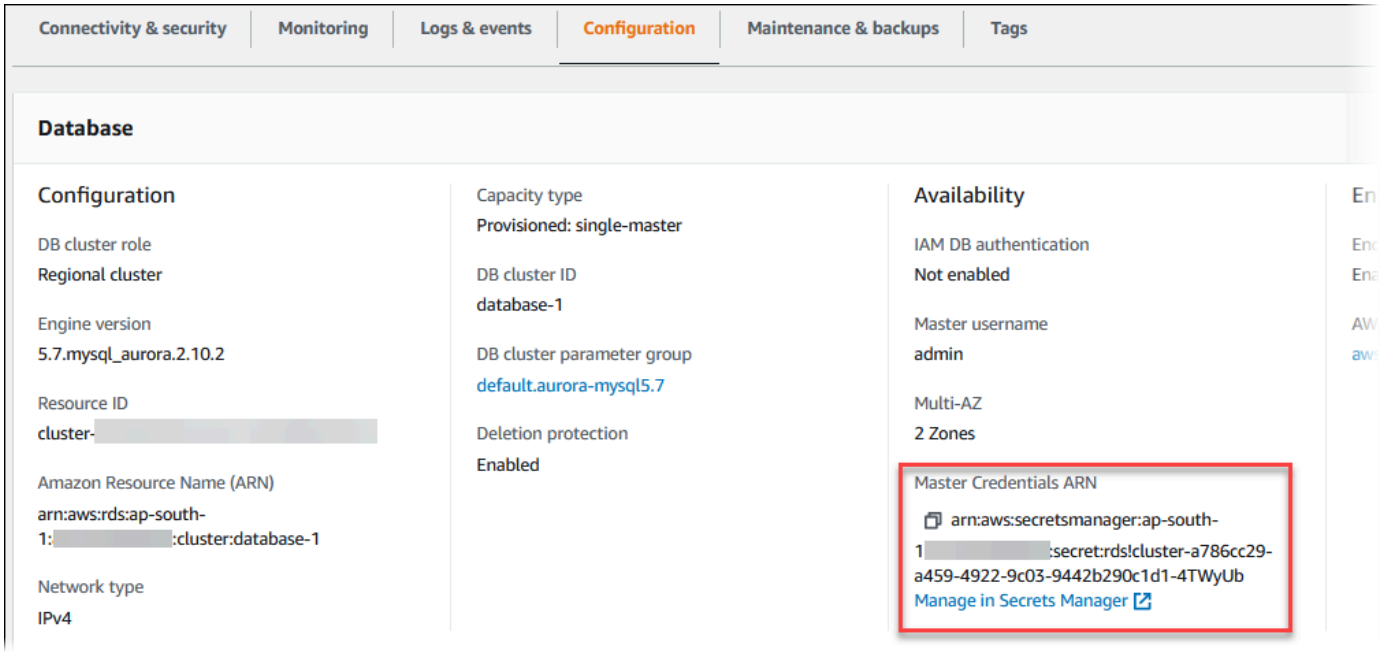
您可以使用 RDS 控制台、AWS CLI 或 RDS API 找到 Aurora 在 Secrets Manager 中管理的密钥的 Amazon 资源名称（ARN）。

控制台

查看有关 Aurora 在 Secrets Manager 中管理的密钥的详细信息

1. 登录AWS Management Console并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Databases (数据库)。
3. 选择数据库集群的名称以显示其详细信息。
4. 选择配置选项卡。

   在 Master Credentials ARN（主凭证 ARN）中，您可以查看密钥 ARN。



您可以单击 Manage in Secrets Manager（在 Secrets Manager 中管理）链接，以在 Secrets Manager 控制台中查看和管理密钥。

AWS CLI

您可以使用 RDS AWS CLI describe-db-clusters 命令查找有关 Aurora 在 Secrets Manager 中管理的密钥的以下信息：

- `SecretArn` – 密钥的 ARN
- `SecretStatus` – 密钥的状态

  可能的状态值包括：

  - `creating` – 密钥正在创建中。
  - `active` – 密钥可用于正常使用和轮换。
  - `rotating` – 密钥正在轮换。
  - `impaired` – 密钥可用于访问数据库凭证，但不能轮换。例如，如果更改权限以使 RDS 无法再访问密钥或密钥的 KMS 密钥，则密钥可能具有此状态。

    当密钥具有此状态时，您可以更正导致该状态的条件。如果您更正导致状态的条件，则状态会一直保持为 `impaired`，直至下一次轮换。或者，您可以修改数据库集群以关闭数据库凭证的自动管理，然后再次修改数据库集群以开启数据库凭证的自动管理。要修改数据库集群，请在 modify-db-cluster 命令中使用 `--manage-master-user-password` 选项。

- `KmsKeyId` – 用于加密密钥的 KMS 密钥的 ARN

指定 `--db-cluster-identifier` 选项可显示特定数据库集群的输出。此示例显示数据库集群使用的密钥的输出。

Example

```
aws rds describe-db-clusters --db-cluster-identifier mydbcluster
```

以下示例显示密钥的输出：

```
"MasterUserSecret": {
                "SecretArn": "arn:aws:secretsmanager:eu-west-1:123456789012:secret:rds!
cluster-033d7456-2c96-450d-9d48-f5de3025e51c-xmJRDx",
                "SecretStatus": "active",
                "KmsKeyId": "arn:aws:kms:eu-
west-1:123456789012:key/0987dcba-09fe-87dc-65ba-ab0987654321"
            }
```

当您拥有密钥 ARN 时，您可以使用 get-secret-value Secrets Manager CLI 命令查看有关该密钥的详细信息。

此示例显示先前示例输出中的密钥的详细信息。

Example

对于 Linux、macOS 或 Unix：

```
aws secretsmanager get-secret-value \
    --secret-id 'arn:aws:secretsmanager:eu-west-1:123456789012:secret:rds!
cluster-033d7456-2c96-450d-9d48-f5de3025e51c-xmJRDx'
```

对于 Windows：

```
aws secretsmanager get-secret-value ^
    --secret-id 'arn:aws:secretsmanager:eu-west-1:123456789012:secret:rds!
cluster-033d7456-2c96-450d-9d48-f5de3025e51c-xmJRDx'
```

RDS API

您可以使用 DescribeDBClusters RDS 操作并将 `DBClusterIdentifier` 参数设置为数据库集群标识符，查看 Aurora 在 Secrets Manager 中管理的密钥的 ARN、状态和 KMS 密钥。输出中包含有关密钥的详细信息。

当您拥有密钥 ARN 时，您可以使用 GetSecretValue Secrets Manager 操作查看有关该密钥的详细信息。

Amazon RDS 中的数据保护

AWS 责任共担模式适用于 Amazon Relational Database Service 中的数据保护。如该模式中所述，AWS 负责保护运行所有 AWS Cloud 的全球基础结构。您负责维护对托管在此基础结构上的内容的控制。您还负责您所使用的 AWS 服务 的安全配置和管理任务。有关数据隐私的更多信息，请参阅数据隐私常见问题。有关欧洲数据保护的信息，请参阅 AWS Security Blog 上的 AWS Shared Responsibility Model and GDPR 博客文章。

出于数据保护目的，建议您保护 AWS 账户 凭证并使用 AWS IAM Identity Center 或 AWS Identity and Access Management（IAM）设置单个用户。这样，每个用户只获得履行其工作职责所需的权限。还建议您通过以下方式保护数据：

- 对每个账户使用多重身份验证（MFA）。
- 使用 SSL/TLS 与 AWS 资源进行通信。我们要求使用 TLS 1.2，建议使用 TLS 1.3。
- 使用 AWS CloudTrail 设置 API 和用户活动日记账记录。有关使用 CloudTrail 跟踪来捕获 AWS 活动的信息，请参阅《AWS CloudTrail 用户指南》中的使用 CloudTrail 跟踪。
- 使用 AWS 加密解决方案以及 AWS 服务中的所有默认安全控制。
- 使用高级托管安全服务（例如 Amazon Macie），它有助于发现和保护存储在 Amazon S3 中的敏感数据。
- 如果在通过命令行界面或 API 访问 AWS 时需要经过 FIPS 140-3 验证的加密模块，请使用 FIPS 端点。有关可用的 FIPS 端点的更多信息，请参阅《美国联邦信息处理标准（FIPS）第 140-3 版》。

强烈建议您切勿将机密信息或敏感信息（如您客户的电子邮件地址）放入标签或自由格式文本字段（如名称字段）。这包括当您通过控制台、API、AWS CLI 或 AWS SDK 使用 Amazon RDS 或其他 AWS 服务 时。在用于名称的标签或自由格式文本字段中输入的任何数据都可能会用于计费或诊断日志。如果您向外部服务器提供网址，强烈建议您不要在网址中包含凭证信息来验证对该服务器的请求。

主题

- 使用加密保护数据
- 互联网络流量隐私

使用加密保护数据

您可以为数据库资源启用加密。您也可以加密与数据库集群的连接。

主题

- 加密 Amazon Aurora 资源
- AWS KMS key 管理
- 使用 SSL/TLS 加密与数据库集群的连接
- 轮换 SSL/TLS 证书

加密 Amazon Aurora 资源

Amazon Aurora 可以加密您的 Amazon Aurora 数据库集群。静态加密的数据包括数据库集群的基础存储、其自动化备份、只读副本和快照。

Amazon Aurora 加密的数据库集群使用行业标准 AES-256 加密算法来对托管 Amazon Aurora 数据库集群的服务器上的数据进行加密。在加密数据后，Amazon Aurora 将以透明方式处理访问的身份验证和数据的解密，并且对性能产生的影响最小。您无需修改数据库客户端应用程序来使用加密。

> **Note**
>
> 对于加密和未加密数据库集群，将对在源与只读副本之间发送的数据进行加密，即使在 AWS 区域之间复制时也是如此。

主题

- Amazon Aurora 资源加密概览
- 加密 Amazon Aurora 数据库集群
- 确定是否为数据库集群开启加密
- Amazon Aurora 加密的可用性
- 传输中加密

- Amazon Aurora 加密的数据库集群的限制

Amazon Aurora 资源加密概览

Amazon Aurora 加密的数据库集群通过保护您的数据免受未经授权的访问来为基础存储提供额外一层数据保护。您可以使用 Amazon Aurora 加密来增强对云中部署的应用程序的数据保护，并满足静态数据加密的合规性要求。对于 Amazon Aurora 加密的数据库集群，所有数据库实例日志、备份和快照均会被加密。有关加密功能的可用性和限制的更多信息，请参阅 Amazon Aurora 加密的可用性和 Amazon Aurora 加密的数据库集群的限制。

Amazon Aurora 使用 AWS Key Management Service 密钥加密这些资源。AWS KMS 将安全、高度可用的硬件和软件结合起来，提供面向云端扩展的密钥管理系统。您可以使用 AWS 托管式密钥，也可以创建客户托管的密钥。

创建加密数据库集群时，您可以为 Amazon Aurora 选择客户托管密钥或 AWS 托管式密钥 来加密数据库集群。如果您没有为客户托管密钥指定密钥标识符，则 Amazon Aurora 会将 AWS 托管式密钥 用于您的新数据库实例。Amazon Aurora 将为您的 AWS 账户创建 Amazon Aurora 的 AWS 托管式密钥。您的 AWS 账户在每个 AWS 区域都有用于 Amazon Aurora 的不同 AWS 托管式密钥。

要管理用于加密和解密 Amazon Aurora 资源的客户自主管理型密钥，您可以使用 AWS Key Management Service（AWS KMS）。

利用 AWS KMS，您可创建客户自主管理型密钥，并定义策略来控制这些客户自主管理型密钥的使用方式。AWS KMS 支持 CloudTrail，因此，您可审核 KMS 密钥使用情况以验证客户自主管理型密钥是否使用得当。可以将客户托管密钥与 Amazon Aurora 和支持的 AWS 服务（例如 Amazon S3、Amazon EBS 和 Amazon Redshift）结合使用。有关与 AWS KMS 集成的服务的列表，请参阅 AWS 服务集成。关于使用 KMS 密钥的一些注意事项：

- 创建加密的数据库实例后，您无法更改该数据库实例使用的 KMS 密钥。因此，请确保先确定您的 KMS 密钥要求，然后再创建加密的数据库实例。

  如果您必须更改数据库集群的加密密钥，请创建集群的手动快照，然后在复制快照时启用加密。有关更多信息，请参阅 re:Post 知识文章。
- 如果复制加密快照，则可以使用不同于用于加密源快照的 KMS 密钥来加密目标快照。
- 如果快照已使用共享该快照的 AWS 托管式密钥 账户的 AWS 进行加密，则您无法共享该快照。
- 数据库集群中的每个数据库实例都使用与数据库集群相同的 KMS 密钥进行加密。
- 您还可以对已加密的 Amazon Aurora 集群的只读副本进行加密。

Important

禁用 KMS 密钥后，Amazon Aurora 可能会失去对数据库集群的 KMS 密钥的访问权限。在这些情况下，加密的数据库集群会进入 `inaccessible-encryption-credentials-recoverable` 状态。数据库集群将保持此状态七天，在此期间实例停止运行。在此期间对该数据库集群进行的 API 调用可能不会成功。要恢复数据库集群，请启用 KMS 密钥并重新启动该数据库集群。从 AWS Management Console、AWS CLI 或 RDS API 启用 KMS 密钥。使用 AWS CLI 命令 start-db-cluster 或 AWS Management Console重新启动数据库集群。`inaccessible-encryption-credentials-recoverable` 状态仅适用于可以停止的数据库集群。这种可恢复状态不适用于无法停止的实例，例如具有跨区域只读副本的集群。有关更多信息，请参阅 the section called “限制”。

如果数据库集群未在七天内恢复，则该集群会进入终端 `inaccessible-encryption-credentials` 状态。在此状态下，数据库集群将无法再使用，您只能从备份还原数据库集群。强烈建议您始终对加密的数据库集群启用备份，以防止数据库中的加密数据丢失。

在数据库集群创建期间，Aurora 会检查发出调用的主体是否有权访问 KMS 密钥，并从 KMS 密钥中生成授权，该授权将在数据库集群的整个生命周期内使用。撤消发出调用的主体对 KMS 密钥的访问权限，不会影响正在运行的数据库。在跨账户场景中使用 KMS 密钥时，例如将快照复制到其他账户，需要与其他账户共享 KMS 密钥。如果您从快照创建数据库集群但未指定不同的 KMS 密钥，则新集群将使用源账户中的 KMS 密钥。创建数据库集群后，撤消密钥访问权限不会影响集群。但是，禁用某个密钥会影响使用该密钥加密的所有数据库集群。为防止出现这种情况，请在快照复制操作期间指定不同的密钥。

有关 KMS 密钥的更多信息，请参阅《AWS Key Management Service 开发人员指南》中的 AWS KMS keys 和AWS KMS key 管理。

加密 Amazon Aurora 数据库集群

要为新数据库集群启用加密，请在控制台上选择 Enable encryption（启用加密）。有关创建数据库集群的信息，请参阅创建 Amazon Aurora 数据库集群。

如果使用 create-db-cluster AWS CLI 命令创建加密的数据库集群，请设置 `--storage-encrypted` 参数。如果使用 CreateDBCluster API 操作，请将 `StorageEncrypted` 参数设置为 true。

创建加密的数据库集群后，您无法更改该数据库集群使用的 KMS 密钥。因此，请确保先确定您的 KMS 密钥要求，然后再创建加密的数据库集群。

如果使用 AWS CLI `create-db-cluster` 命令创建带有客户托管密钥的加密数据库集群，请将 `--kms-key-id` 参数设置为 KMS 密钥的任何密钥标识符。如果您使用 Amazon RDS API

`CreateDBInstance` 操作，请将 `KmsKeyId` 参数设置为 KMS 密钥的任何密钥标识符。要在其他 AWS 账户中使用客户托管密钥，请指定密钥 ARN 或别名 ARN。

确定是否为数据库集群开启加密

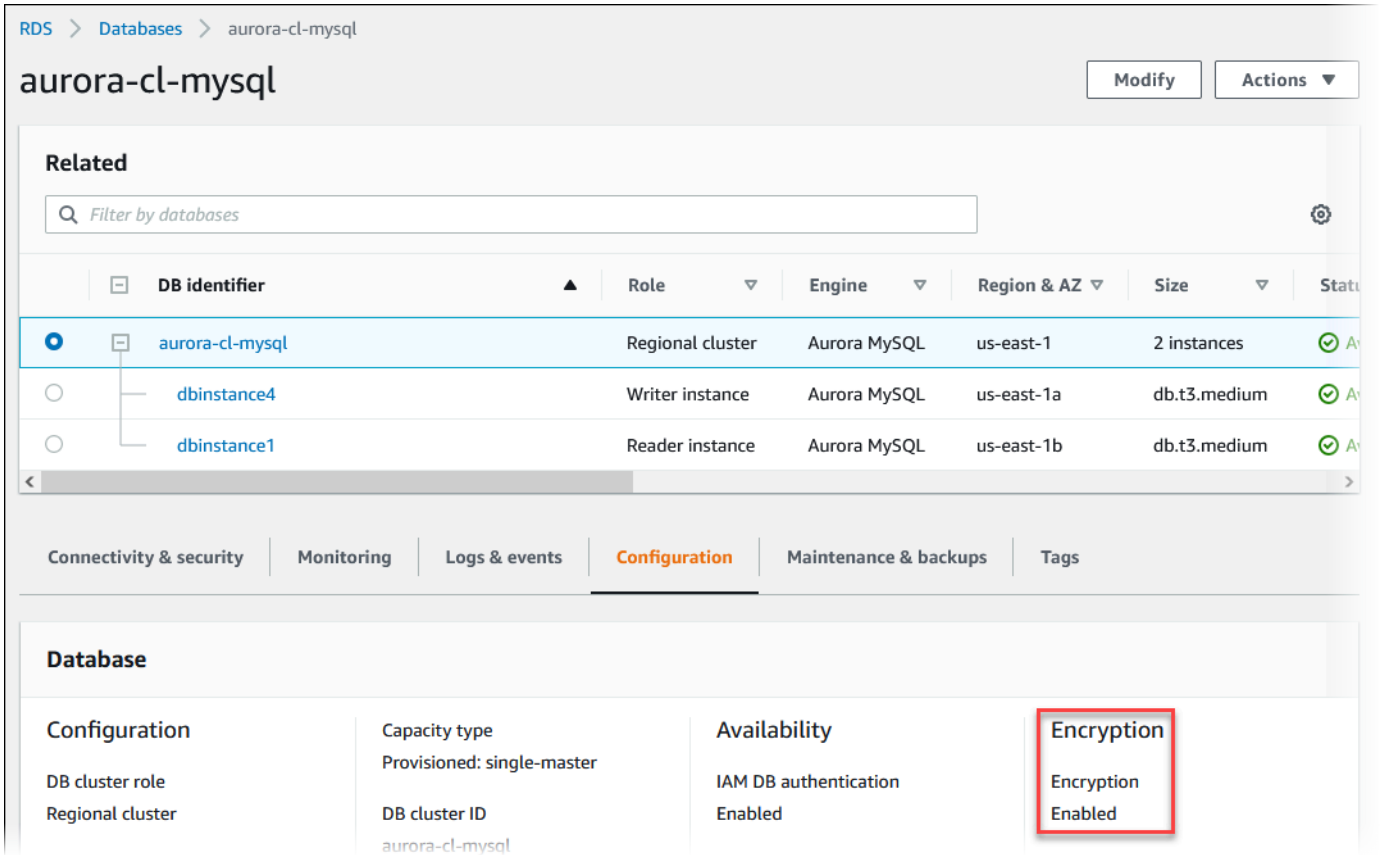
您可以使用 AWS Management Console、AWS CLI 或 RDS API 确定是否为数据库集群开启静态加密。

控制台

要确定是否为数据库集群开启静态加密

1. 登录AWS Management Console并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Databases (数据库)。
3. 选择要检查的数据库集群的名称以显示其详细信息。
4. 选择 Configuration（配置）选项卡，然后检查 Encryption（加密）值。

   它显示 Enabled（已启用）或 Not enabled（未启用）。



AWS CLI

要使用 AWS CLI 确定是否已为数据库集群启用静态加密，请使用以下选项调用 describe-db-clusters 命令：

- `--db-cluster-identifier` – 数据库集群的名称。

下面的示例使用查询就 mydb 数据库集群的静态加密返回 TRUE 或 FALSE。

Example

```
aws rds describe-db-clusters --db-cluster-identifier mydb --query "*[].
{StorageEncrypted:StorageEncrypted}" --output text
```

RDS API

要使用 Amazon RDS API 确定是否已为数据库集群启用静态加密，请使用以下参数调用 DescribeDBClusters 操作：

- `DBClusterIdentifier` – 数据库集群的名称。

Amazon Aurora 加密的可用性

Amazon Aurora 加密当前可用于所有数据库引擎和存储类型。

> **Note**
>
> Amazon Aurora 加密不适用于 db.t2.micro 数据库实例类。

传输中加密

物理层加密

所有在 AWS 全球网络上跨 AWS 区域流动的数据在离开受 AWS 保护的设施之前，都会在物理层自动加密。可用区之间的所有流量均已加密。其它加密层（包括本节中列出的加密层）可能会提供额外保护。

Amazon VPC 对等连接和 Transit Gateway 跨区域对等连接提供的加密

使用 Amazon VPC 和 Transit Gateway 对等连接的所有跨区域流量在退出区域时都会自动批量加密。所有流量在离开受 AWS 保护的设施之前，均会在物理层自动提供额外一层加密。

实例之间的加密

AWS 在所有类型的数据库实例之间提供安全的私有连接。此外，某些实例类型使用底层 Nitro 系统硬件的卸载功能，自动加密实例之间的传输中流量。此加密使用关联数据的身份验证加密（AEAD）算法，采用 256 位加密。这对网络性能没有影响。要在实例之间支持这种额外的传输中流量加密，必须满足以下要求：

- 使用以下实例类型：
  - 通用型：M6i、M6id、M6in、M6idn、M7g
  - 内存优化型：R6i、R6id、R6in、R6idn、R7g、X2idn、X2iedn、X2iezn
- 这些实例位于同一 AWS 区域。
- 这些实例位于相同 VPC 或对等的 VPC 中，并且流量不会通过虚拟网络设备或服务（如负载均衡器或中转网关）传输。

Amazon Aurora 加密的数据库集群的限制

Amazon Aurora 加密的数据库集群存在以下限制：

- 您无法在加密的数据库集群上关闭加密。
- 您无法创建未加密数据库集群的加密快照。
- 加密数据库集群的快照必须使用与数据库集群相同的 KMS 密钥进行加密。
- 您无法将未加密的数据库集群转换为加密的数据库集群。但是，您可以将未加密的快照还原为加密的 Aurora 数据库集群。为此，请在从未加密的快照还原时指定 KMS 密钥。
- 您无法从未加密的 Aurora 数据库集群创建已加密的 Aurora 副本。您无法从加密的 Aurora 数据库集群创建未加密的 Aurora 副本。
- 要将已加密快照从一个 AWS 区域复制到另一个区域，您必须指定目标 AWS 区域的 KMS 密钥。这是因为 KMS 密钥特定于在其中创建它们的 AWS 区域。

  源快照在复制过程中保持加密状态。Amazon Aurora 使用信封加密在复制过程中保护数据。有关信封加密的更多信息，请参阅AWS Key Management Service开发人员指南中的信封加密。
- 您无法对加密数据库集群取消加密。但是，您可以从加密的数据库集群中导出数据，然后将数据导入未加密的数据库集群。

AWS KMS key 管理

Amazon Aurora 自动与 AWS Key Management Service（AWS KMS）集成以进行密钥管理。Amazon Aurora 使用信封加密。有关信封加密的更多信息，请参阅 AWS Key Management Service 开发人员指南中的信封加密。

您可以使用两种类型的 AWS KMS 密钥来加密数据库集群。

- 要完全控制 KMS 密钥，您必须创建客户托管密钥。有关客户托管密钥的更多信息，请参阅 AWS Key Management Service 开发人员指南中的客户托管密钥。
- AWS 托管式密钥 是由与 AWS KMS 集成的 AWS 服务代表您在账户中创建、管理和使用的 KMS 密钥。默认情况下，RDS AWS 托管式密钥（`aws/rds`）用于加密。您无法管理、轮换或删除 RDS AWS 托管式密钥。有关 AWS 托管式密钥 的更多信息，请参阅《AWS Key Management Service 开发人员指南》中的 AWS 托管式密钥。

要管理用于 Amazon Aurora 加密数据库集群的 KMS 密钥，您可以使用 AWS KMS 控制台中的 AWS Key Management Service（AWS KMS）、AWS CLI 或 AWS KMS API。要查看利用 AWS 托管式或客户托管密钥执行的每个操作的审计日志，请使用 AWS CloudTrail。有关密钥轮换的更多信息，请参阅轮换 AWS KMS 密钥。

授权使用客户托管密钥

Aurora 在加密操作中使用客户自主管理型密钥时，其代表创建或更改 Aurora 资源的用户执行操作。

要使用客户自主管理型密钥创建 Aurora 资源，用户必须具有对客户自主管理型密钥调用以下操作的权限：

- `kms:CreateGrant`
- `kms:DescribeKey`

您可以在密钥策略中指定这些所需权限，或者在密钥策略允许的情况下在 IAM 策略中指定这些权限。

> **Important**
>
> 当您在 Amazon RDS 等托管式服务的 AWS KMS 密钥策略中对所有资源（*）使用显式拒绝语句时，您必须指定一个条件来支持拥有资源的账户。如果没有此条件，操作可能会失败，即使拒绝规则中包含针对您的 IAM 用户的例外情况也是如此。

> **Tip**
>
> 为遵循最小特权原则，请不要允许对 `kms:CreateGrant` 拥有完全访问权限。而是仅当 AWS 服务代表用户创建授权时，才使用 kms:ViaService 条件密钥来支持用户对 KMS 密钥创建授权。

您可以通过各种方式使 IAM 策略更严格。例如，如果您要支持将客户自主管理型密钥仅用于源自 Aurora 的请求，请将 kms:ViaService 条件密钥与 `rds.<region>.amazonaws.com` 值结合使用。此外，您可以使用Amazon RDS 加密上下文中的密钥或值作为使用客户托管密钥进行加密的条件。

有关更多信息，请参阅《AWS Key Management Service 开发人员指南》中的允许其他账户中的用户使用 KMS 密钥和 AWS KMS 中的密钥策略。

Amazon RDS 加密上下文

当 Aurora 使用您的 KMS 密钥时，或者当 Amazon EBS 代表 Aurora 使用 KMS 密钥时，服务会指定加密上下文。加密上下文是 AWS KMS 用于确保数据完整性而使用的额外的身份验证数据 (AAD)。在为加密操作指定加密上下文时，该服务必须为解密操作指定同一加密上下文。否则，解密将失败。加密上下文还将写至您的 AWS CloudTrail 日志中，以帮助您了解为什么使用给定的 KMS 密钥。您的 CloudTrail 日志可能包含多个描述 KMS 密钥使用情况的条目，但每个日志条目中的加密上下文可以帮助您确定该特定使用的原因。

至少，Aurora 始终将数据库集群 ID 用于加密上下文，如以下 JSON 格式的示例所示：

```
{ "aws:rds:dbc-id": "cluster-CQYSMDPBRZ7BPMH7Y3RTDG5QY" }
```

此加密上下文可以帮助您确定使用您的 KMS 密钥的数据库集群。

当您的 KMS 密钥用于特定的数据库集群和特定的 Amazon EBS 卷时，数据库集群 ID 和 Amazon EBS 卷 ID 将用于加密上下文，如以下 JSON 格式的示例所示：

```
{
  "aws:rds:dbc-id": "cluster-BRG7VYS3SVIFQW7234EJQOM5RQ",
  "aws:ebs:id": "vol-ad8c6542"
}
```

使用 SSL/TLS 加密与数据库集群的连接

您可以使用应用程序中的安全套接字层 (SSL) 或传输层安全 (TLS) 来加密与运行 Aurora MySQL 或 Aurora PostgreSQL 的数据库集群的连接。

SSL/TLS 连接通过加密可在您的客户端和数据库集群之间移动的数据来提供一层安全性。（可选）您的 SSL/TLS 连接可以通过验证安装在数据库上的服务器证书来执行服务器身份验证。要请求服务器身份验证，请遵循以下一般流程：

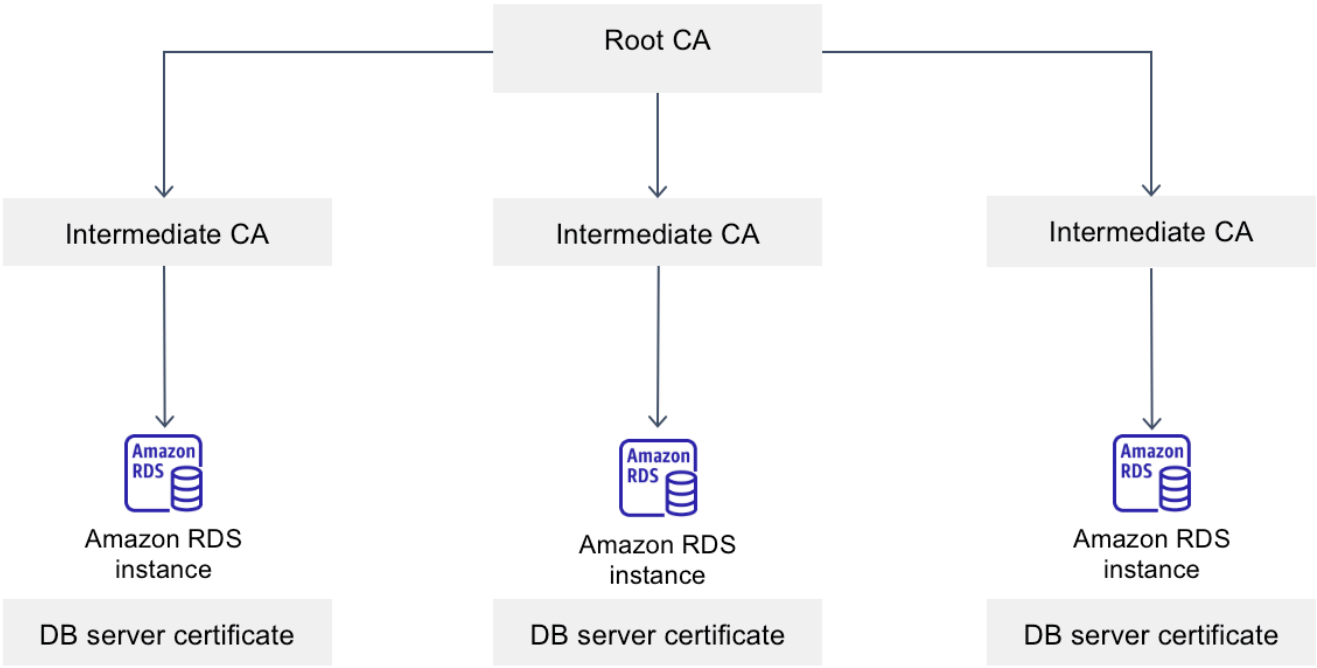
1. 为数据库选择对数据库服务器证书进行签名的证书颁发机构（CA）。有关证书颁发机构的更多信息，请参阅证书颁发机构。
2. 下载证书包，以便在连接到数据库时使用。要下载证书捆绑包，请参阅按 AWS 区域列出的证书捆绑包。

   > **Note**
   >
   > 只能使用 SSL/TLS 连接下载所有证书。

3. 使用数据库引擎用于实现 SSL/TLS 连接的过程连接到数据库。每个数据库引擎都有自己的用于实施 SSL/TLS 的过程。要了解如何为您的数据库实施 SSL/TLS，请使用对应于您的数据库引擎的链接：
   - 使用 Amazon Aurora MySQL 实现高安全性
   - 使用 Amazon Aurora PostgreSQL 实现高安全性

证书颁发机构

证书颁发机构（CA）是用于标识证书链顶部的根 CA 的证书。CA 签署安装在每个数据库实例上的数据库服务器证书。数据库服务器证书将数据库实例标识为可信服务器。



Amazon RDS 提供以下 CA 来签署数据库的数据库服务器证书。

证书颁发机构（CA）	描述	公用名（CN）
rds-ca-2019	使用具有 RSA 2048 私有密钥算法和 SHA256 签名算法的证书颁发机构。此 CA 将于 2024 年到期，不支持服务器证书自动轮换。如果您正在使用此 CA 并希望保持相同的标准，我们建议您切换到 rds-ca-rsa2048-g1 CA。	Amazon RDS *region-id entifier* CA 2019
rds-ca-rsa2048-g1	在大多数 AWS 区域中，使用具有 RSA 2048 私有密钥算法和 SHA256 签名算法的证书颁发机构。 在 AWS GovCloud (US) Regions中，此 CA 使用具有 RSA 2048 私有密钥算法和 SHA384 签名算法的证书颁发机构。	Amazon RDS *region-id entifier* RSA2048 G1

证书颁发机构（CA）	描述	公用名（CN）
	此 CA 的有效期比 rds-ca-2019 CA 长。此 CA 支持服务器证书自动轮换。	
rds-ca-rsa4096-g1	使用具有 RSA 4096 私有密钥算法和 SHA384 签名算法的证书颁发机构。此 CA 支持服务器证书自动轮换。	Amazon RDS *region-id entifier* RSA4096 G1
rds-ca-ecc384-g1	使用具有 ECC 384 私有密钥算法和 SHA384 签名算法的证书颁发机构。此 CA 支持服务器证书自动轮换。	Amazon RDS *region-id entifier* ECC384 G1

> **Note**
>
> 如果您使用的是 AWS CLI，则可以使用 describe-certificates 查看上面列出的证书颁发机构的有效性。

这些 CA 证书包含在区域和全球证书捆绑包中。当您将 rds-ca-rsa2048-g1、rds-ca-rsa4096-g1 或 rds-ca-ecc384-g1 CA 用于数据库时，RDS 会管理数据库上的数据库服务器证书。RDS 会在数据库服务器证书过期之前自动轮换此证书。

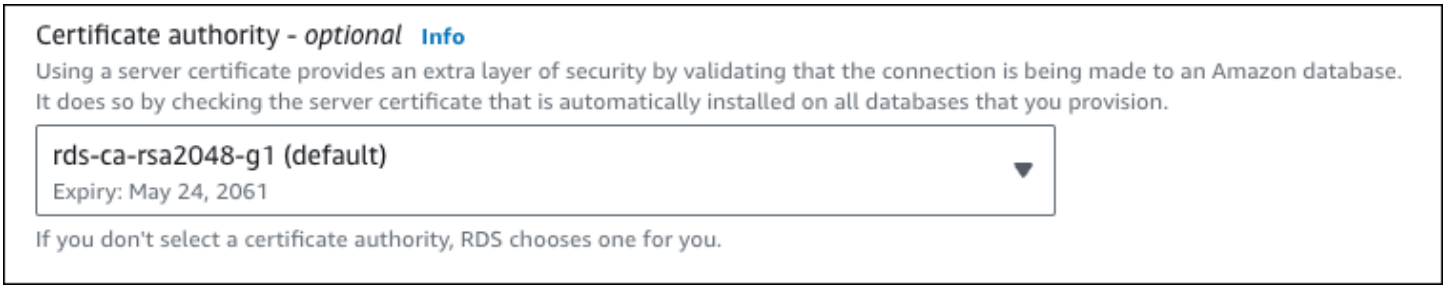
为数据库设置 CA

您可以在执行以下任务时为数据库设置 CA：

- 创建 Aurora 数据库集群 - 当您使用 AWS CLI 或 RDS API 在数据库集群中创建第一个数据库实例时，可以为 Aurora 集群中的数据库实例设置 CA。目前，在使用 RDS 控制台创建数据库集群时，无法为数据库集群中的数据库实例设置 CA。有关说明，请参阅 创建 Amazon Aurora 数据库集群。
- 修改数据库实例 - 您可以通过修改数据库集群中的数据库实例为其设置 CA。有关说明，请参阅 修改数据库集群中的数据库实例。

> **Note**
>
> 默认 CA 设置为 rds-ca-rsa2048-g1。您可以使用 modify-certificates 命令为您的 AWS 账户覆盖默认 CA。

可用的 CA 取决于数据库引擎和数据库引擎版本。使用 AWS Management Console时，您可以使用 Certificate authority（证书颁发机构）设置选择 CA，如下图所示。



控制台仅显示可用于数据库引擎和数据库引擎版本的 CA。如果您使用的是 AWS CLI，则可以使用 create-db-instance 或 modify-db-instance 命令为数据库实例设置 CA。

如果您使用的是 AWS CLI，则可以使用 describe-certificates 命令查看您账户的可用 CA。此命令还在输出的 `ValidTill` 中显示每个 CA 的到期日期。您可以使用 describe-db-engine-versions 命令找到适用于特定数据库引擎和数据库引擎版本的 CA。

以下示例显示了可用于默认 RDS for PostgreSQL 数据库引擎版本的 CA。

```
aws rds describe-db-engine-versions --default-only --engine postgres
```

输出类似于以下内容。`SupportedCACertificateIdentifiers` 中列出了可用的 CA。输出还显示数据库引擎版本是否支持在 `SupportsCertificateRotationWithoutRestart` 中不重新启动的情况下轮换证书。

```
{
    "DBEngineVersions": [
        {
            "Engine": "postgres",
            "MajorEngineVersion": "13",
            "EngineVersion": "13.4",
            "DBParameterGroupFamily": "postgres13",
            "DBEngineDescription": "PostgreSQL",
            "DBEngineVersionDescription": "PostgreSQL 13.4-R1",
```

```
            "ValidUpgradeTarget": [],
            "SupportsLogExportsToCloudwatchLogs": false,
            "SupportsReadReplica": true,
            "SupportedFeatureNames": [
                "Lambda"
            ],
            "Status": "available",
            "SupportsParallelQuery": false,
            "SupportsGlobalDatabases": false,
            "SupportsBabelfish": false,
            "SupportsCertificateRotationWithoutRestart": true,
            "SupportedCACertificateIdentifiers": [
                "rds-ca-2019",
                "rds-ca-rsa2048-g1",
                "rds-ca-ecc384-g1",
                "rds-ca-rsa4096-g1"
            ]
        }
    ]
}
```

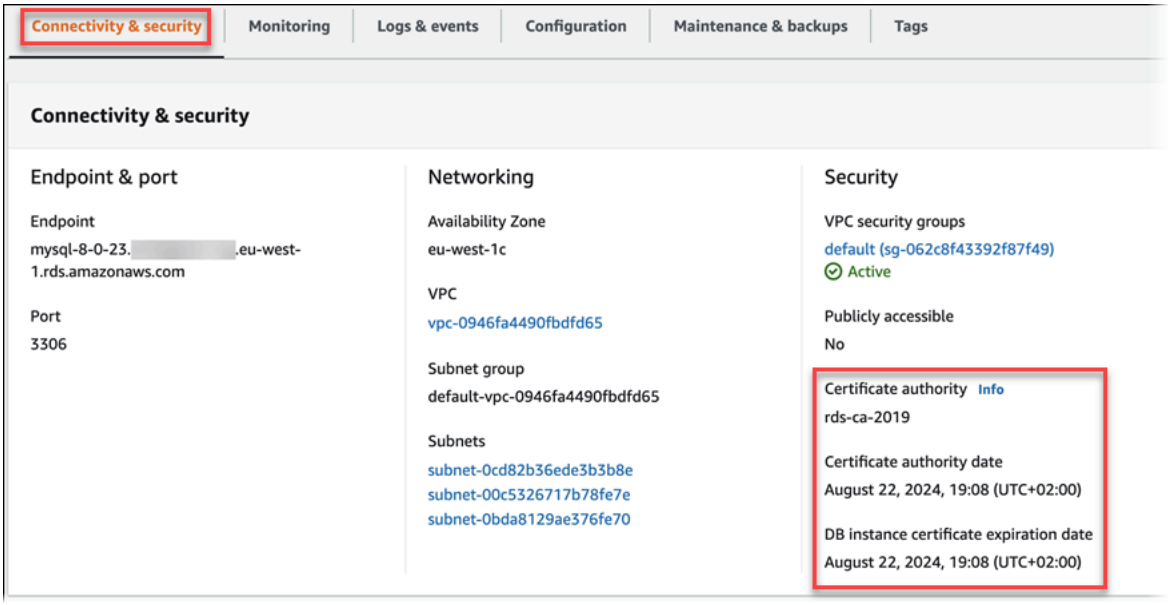
数据库服务器证书有效期

数据库服务器证书的有效期取决于数据库引擎和数据库引擎版本。如果数据库引擎版本支持在不重启的情况下轮换证书，则数据库服务器证书的有效期为 1 年。否则，有效期为 3 年。

有关数据库服务器证书轮换的更多信息，请参阅自动服务器证书轮换。

查看数据库实例的 CA

您可以通过查看控制台中的连接性和安全性选项卡来查看有关数据库的 CA 的详细信息，如下图所示。



如果您使用的是 AWS CLI，您可以使用 describe-db-instances 命令，查看有关数据库实例的 CA 的详细信息。

下载 Aurora 的证书捆绑包

当您使用 SSL 或 TLS 连接到数据库时，数据库实例需要来自 Amazon RDS 的信任证书。在下表中选择相应的链接，来下载与托管数据库的 AWS 区域相对应的捆绑包。

按 AWS 区域列出的证书捆绑包

所有 AWS 区域和 GovCloud（美国）区域的证书捆绑包均包含以下证书：

- `rds-ca-2019` 中间证书和根证书。
- `rds-ca-rsa2048-g1`、`rds-ca-rsa4096-g1` 和 `rds-ca-ecc384-g1` 根 CA 证书。您的应用程序信任存储只需要注册根 CA 证书。

> **Note**
>
> Amazon RDS 代理和 Aurora Serverless v1 使用来自 AWS Certificate Manager (ACM) 的证书。如果您使用的是 RDS 代理，则无需下载 Amazon RDS 证书或更新使用 RDS 代理连接的应用程序。有关更多信息，请参阅 将 TLS/SSL 与 RDS 代理结合使用。
> 如果您使用的是 Aurora Serverless v1，则不需要下载 Amazon RDS 证书。有关更多信息，请参阅 将 TLS/SSL 与 Aurora Serverless v1 结合使用。

要下载 AWS 区域的证书捆绑包，请在下表中选择与托管数据库的 AWS 区域相对应的链接。

AWS 区域	证书捆绑包 (PEM)	证书捆绑包 (PKCS7)
任何商业 AWS 区域	global-bundle.pem	global-bundle.p7b
美国东部（弗吉尼亚州北部）	us-east-1-bundle.pem	us-east-1-bundle.p7b
US East (Ohio)	us-east-2-bundle.pem	us-east-2-bundle.p7b
美国西部（加利福尼亚北部）	us-west-1-bundle.pem	us-west-1-bundle.p7b
美国西部（俄勒冈州）	us-west-2-bundle.pem	us-west-2-bundle.p7b
Africa (Cape Town)	af-south-1-bundle.pem	af-south-1-bundle.p7b
Asia Pacific (Hong Kong)	ap-east-1-bundle.pem	ap-east-1-bundle.p7b
亚太地区（海得拉巴）	ap-south-2-bundle.pem	ap-south-2-bundle.p7b
亚太地区（雅加达）	ap-southeast-3-bundle.pem	ap-southeast-3-bundle.p7b
亚太地区（墨尔本）	ap-southeast-4-bundle.pem	ap-southeast-4-bundle.p7b
亚太地区（孟买）	ap-south-1-bundle.pem	ap-south-1-bundle.p7b
亚太地区（大阪）	ap-northeast-3-bundle.pem	ap-northeast-3-bundle.p7b
亚太区域（东京）	ap-northeast-1-bundle.pem	ap-northeast-1-bundle.p7b
亚太地区（首尔）	ap-northeast-2-bundle.pem	ap-northeast-2-bundle.p7b
亚太地区（新加坡）	ap-southeast-1-bundle.pem	ap-southeast-1-bundle.p7b
亚太地区（悉尼）	ap-southeast-2-bundle.pem	ap-southeast-2-bundle.p7b
加拿大（中部）	ca-central-1-bundle.pem	ca-central-1-bundle.p7b
加拿大西部（卡尔加里）	ca-west-1-bundle.pem	ca-west-1-bundle.p7b
欧洲地区（法兰克福）	eu-central-1-bundle.pem	eu-central-1-bundle.p7b

AWS 区域	证书捆绑包 (PEM)	证书捆绑包 (PKCS7)
欧洲地区（爱尔兰）	eu-west-1-bundle.pem	eu-west-1-bundle.p7b
欧洲地区（伦敦）	eu-west-2-bundle.pem	eu-west-2-bundle.p7b
欧洲地区（米兰）	eu-south-1-bundle.pem	eu-south-1-bundle.p7b
欧洲地区（巴黎）	eu-west-3-bundle.pem	eu-west-3-bundle.p7b
欧洲（西班牙）	eu-south-2-bundle.pem	eu-south-2-bundle.p7b
欧洲地区（斯德哥尔摩）	eu-north-1-bundle.pem	eu-north-1-bundle.p7b
欧洲（苏黎世）	eu-central-2-bundle.pem	eu-central-2-bundle.p7b
以色列（特拉维夫）	il-central-1-bundle.pem	il-central-1-bundle.p7b
中东（巴林）	me-south-1-bundle.pem	me-south-1-bundle.p7b
中东（阿联酋）	me-central-1-bundle.pem	me-central-1-bundle.p7b
南美洲（圣保罗）	sa-east-1-bundle.pem	sa-east-1-bundle.p7b
任何 AWS GovCloud (US) Region	global-bundle.pem	global-bundle.p7b
AWS GovCloud（美国东部）	us-gov-east-1-bundle.pem	us-gov-east-1-bundle.p7b
AWS GovCloud（美国西部）	us-gov-west-1-bundle.pem	us-gov-west-1-bundle.p7b

查看 CA 证书的内容

要检查您的 CA 证书包的内容，请使用以下命令：

```
keytool -printcert -v -file global-bundle.pem
```

轮换 SSL/TLS 证书

Amazon RDS 证书颁发机构证书 rds-ca-2019 于 2024 年 8 月到期。如果您使用或计划使用带有证书验证的安全套接字层（SSL）或传输层安全性协议（TLS）来连接您的 RDS 数据库实例，则考虑使用

新的 CA 证书之一：rds-ca-rsa2048-g1、rds-ca-rsa4096-g1 或 rds-ca-ecc384-g1。如果您当前未将 SSL/TLS 用于证书验证，则可能仍有过期的 CA 证书，如果您计划为证书验证使用 SSL/TLS 连接到 RDS 数据库，则必须将它们更新为新的 CA 证书。

Amazon RDS 提供新 CA 证书作为AWS最佳安全实践。有关新证书和受支持的 AWS 区域的信息，请参阅使用 SSL/TLS 加密与数据库集群的连接。

要更新数据库的 CA 证书，请使用以下方法：

- 通过修改数据库实例来更新 CA 证书
- 通过应用维护来更新 CA 证书

更新数据库实例来使用新的 CA 证书之前，请确保更新连接到 RDS 数据库的客户端或应用程序。

轮换证书的注意事项

在轮换证书之前，请考虑以下情况：

- Amazon RDS 代理和 Aurora Serverless v1 使用来自 AWS Certificate Manager（ACM）的证书。如果您使用的是 RDS 代理，当您轮换 SSL/TLS 证书时，您不需要更新使用 RDS 代理连接的应用程序。有关更多信息，请参阅 将 TLS/SSL 与 RDS 代理结合使用。
- 如果您使用的是 Aurora Serverless v1，则不需要下载 Amazon RDS 证书。有关更多信息，请参阅 将 TLS/SSL 与 Aurora Serverless v1 结合使用。
- 如果您将 Go 版本 1.15 应用程序与在 2020 年 7 月 28 日之前创建或更新到 rds-ca-2019 证书的数据库实例一起使用，则必须再次更新证书。将证书更新为 rds-ca-rsa2048-g1、rds-ca-rsa4096-g1 或 rds-ca-ecc384-g1，具体取决于您的引擎。

  使用新的 CA 证书标识符，。您可以使用 `describe-db-engine-versions` 命令找到适用于特定数据库引擎和数据库引擎版本的 CA。

  如果您在 2020 年 7 月 28 日之后创建了数据库或更新了其证书，则无需执行任何操作。有关更多信息，请参阅 Go GitHub 问题 #39568。

通过修改数据库实例来更新 CA 证书

以下示例将您的 CA 证书从 rds-ca-2019 更新为 rds-ca-rsa2048-g1。您可以选择其他证书。有关更多信息，请参阅证书颁发机构。

更新您的应用程序信任存储，以减少任何与更新 CA 证书相关的停机时间。有关与 CA 证书轮换相关的重启的更多信息，请参阅自动服务器证书轮换。

通过修改数据库实例来更新 CA 证书

1. 下载新的 SSL/TLS 证书，如使用 SSL/TLS 加密与数据库集群的连接中所述。
2. 更新应用程序以使用新的 SSL/TLS 证书。

   更新应用程序以使用新 SSL/TLS 证书的方法取决于特定的应用程序。请与应用程序开发人员一起更新应用程序的 SSL/TLS 证书。

   有关检查 SSL/TLS 连接和更新每个数据库引擎的应用程序的信息，请参阅以下主题：

   - 使用新的 TLS 证书更新应用程序，以连接到 Aurora MySQL 数据库集群
   - 使用新 SSL/TLS 证书更新应用程序，以连接到 Aurora PostgreSQL 数据库集群

   有关更新 Linux 操作系统信任存储的示例脚本，请参阅将证书导入信任存储的示例脚本。

   > **Note**
   >
   > 证书捆绑包包含新旧 CA 证书，因此您可以安全地升级应用程序并在转换期间保持连接。如果您正在使用 AWS Database Migration Service 将数据库迁移到数据库集群，我们建议您使用证书捆绑包来确保迁移期间的连接性。

3. 修改数据库实例，以便将 CA 从 rds-ca-2019 更改为 rds-ca-rsa2048-g1。要检查您的数据库是否需要重启才能更新 CA 证书，请使用 describe-db-engine-versions 命令并检查 `SupportsCertificateRotationWithoutRestart` 标志。

   > **Note**
   >
   > 修改后重启您的 Babelfish 集群以更新 CA 证书。

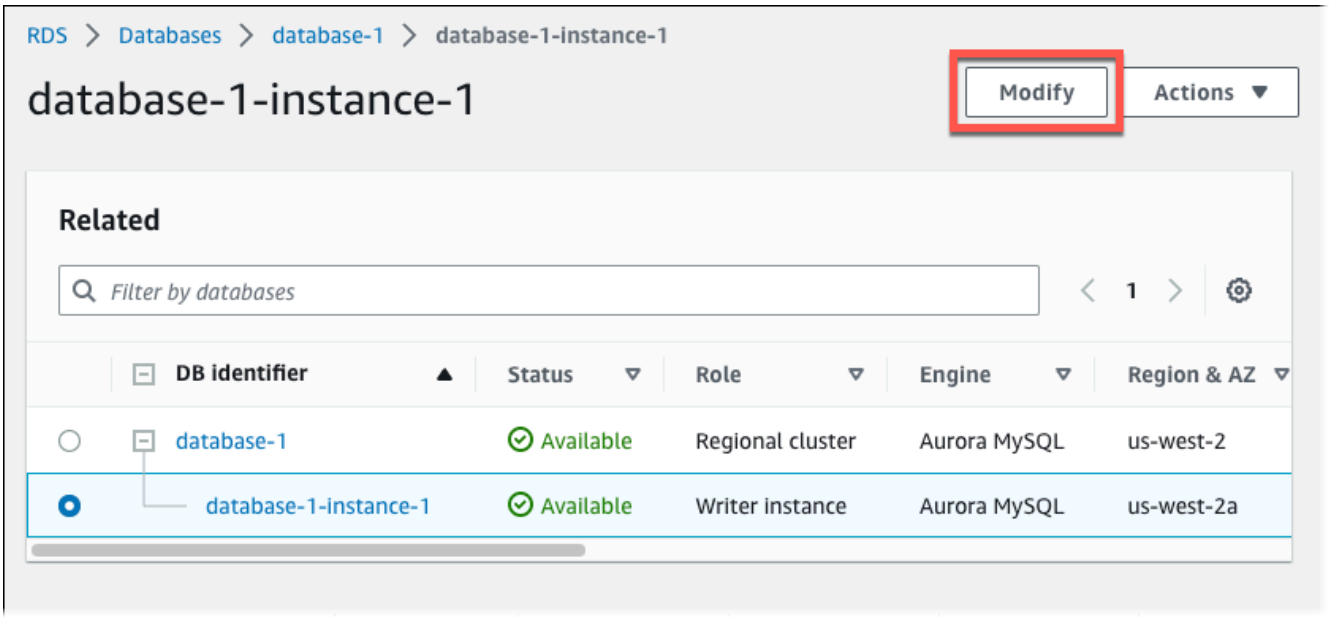
   > **Important**
   >
   > 如果您在证书到期后遇到连接问题，请通过在控制台中指定 Apply immediately (立即应用) 或者 使用 `--apply-immediately` 指定 AWS CLI 选项来使用“立即应用”选项。默认情况下，此操作安排在您的下个维护时段运行。

要为与默认 RDS CA 不同的集群 CA 设置覆盖，请使用 modify-certificates CLI 命令。

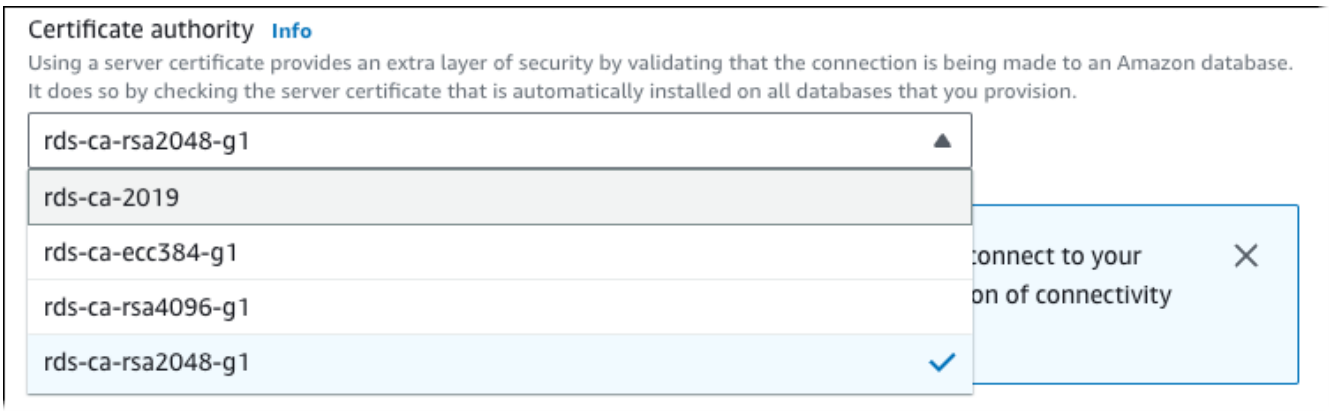
您可以使用 AWS Management Console或 AWS CLI，对于数据库实例，将 CA 证书从 rds-ca-2019 更改为 rds-ca-rsa2048-g1。

Console

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择数据库，然后选择要修改的数据库实例。
3. 选择 Modify(修改)。



4. 在连接部分中，选择 rds-ca-rsa2048-g1。



5. 选择继续，查看修改摘要。

6. 要立即应用更改，请选择立即应用。
7. 在确认页面上，检查您的更改。如果更改正确无误，请选择修改数据库实例以保存更改。

> **Important**
>
> 安排此操作时，请确保已预先更新客户端信任存储。

也可以选择 Back 编辑您的更改，或选择 Cancel 取消更改。

AWS CLI

要使用 AWS CLI 将数据库实例的 CA 从 rds-ca-2019 更改为 rds-ca-rsa2048-g1，请调用 modify-db-instance 或 modify-db-cluster 命令。指定数据库实例标识符和 `--ca-certificate-identifier` 选项。

使用 `--apply-immediately` 参数可立即应用更新。默认情况下，此操作安排在您的下个维护时段运行。

> **Important**
>
> 安排此操作时，请确保已预先更新客户端信任存储。

Example

以下示例通过将 CA 证书设置为 `rds-ca-rsa2048-g1` 来修改 `mydbinstance`。

对于 Linux、macOS 或 Unix：

```
aws rds modify-db-instance \
    --db-instance-identifier mydbinstance \
    --ca-certificate-identifier rds-ca-rsa2048-g1
```

对于 Windows：

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

```
    --ca-certificate-identifier rds-ca-rsa2048-g1
```

> **Note**
>
> 如果您的实例需要重启，可以使用 modify-db-instance CLI 命令并指定 `--no-certificate-rotation-restart` 选项。

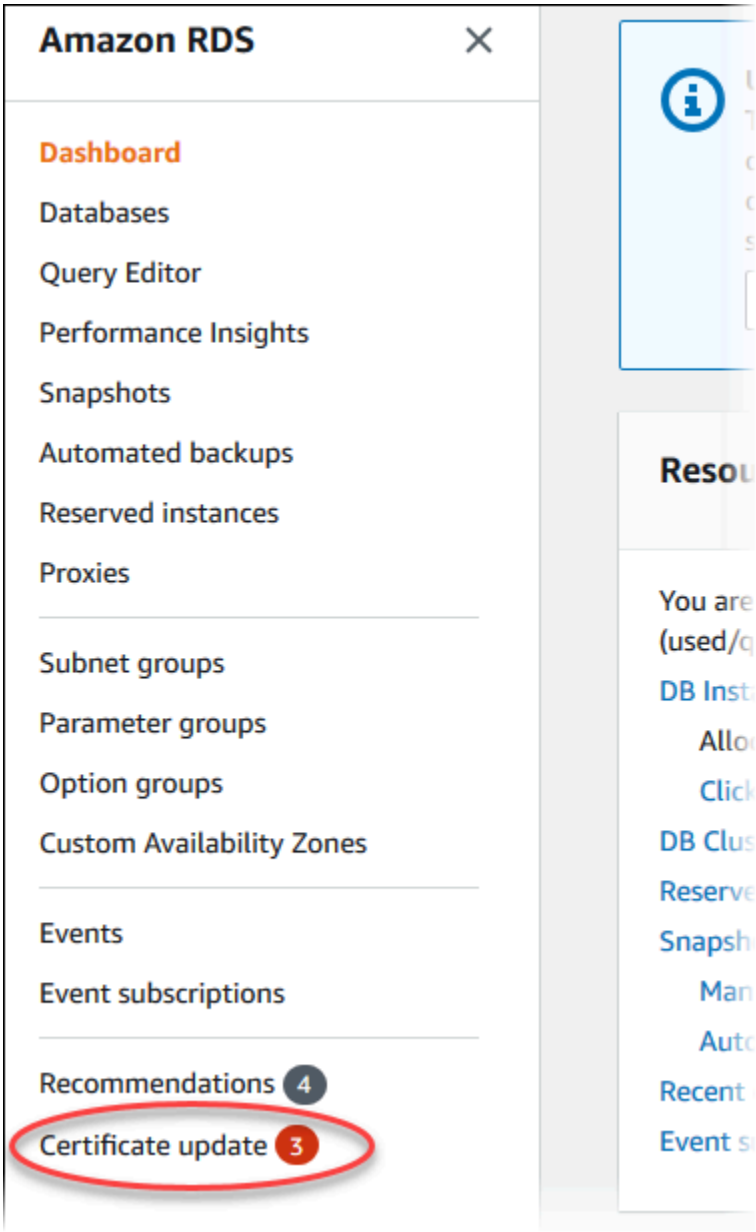
通过应用维护来更新 CA 证书

完成以下步骤，通过应用维护来更新 CA 证书。

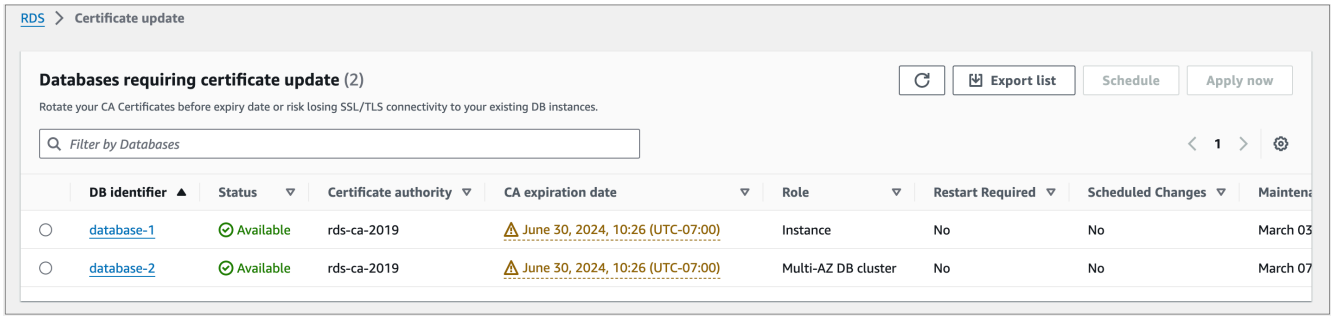
Console

通过应用维护来更新 CA 证书

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择证书更新。



将显示需要更新证书的数据库页面。



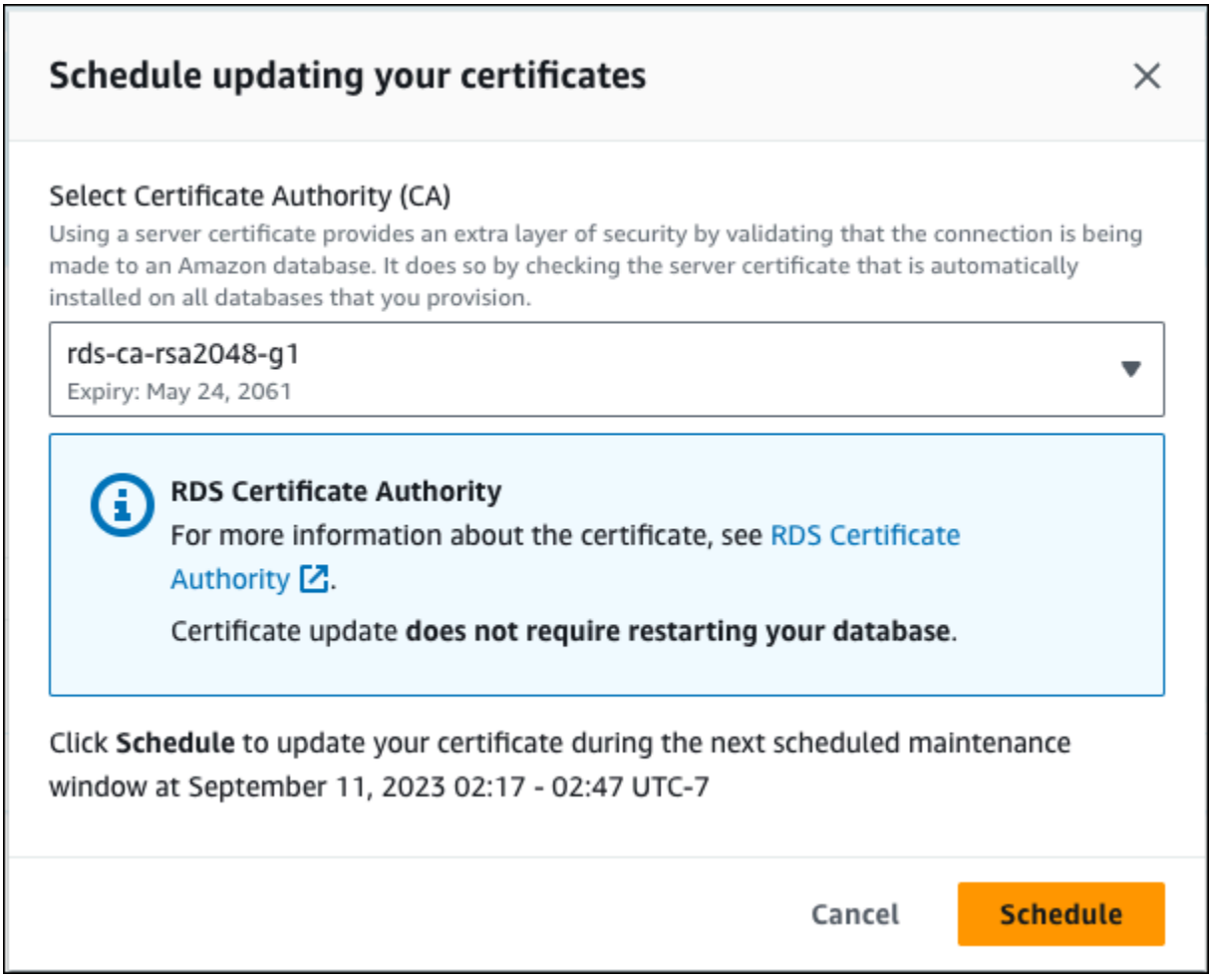
> **Note**
>
> 该页面仅显示当前 AWS 区域的数据库实例。如果您在多个 AWS 区域中有数据库，请检查每个 AWS 区域中的该页面，以查看具有旧 SSL/TLS 证书的所有数据库实例。

3. 选择要更新的数据库实例。

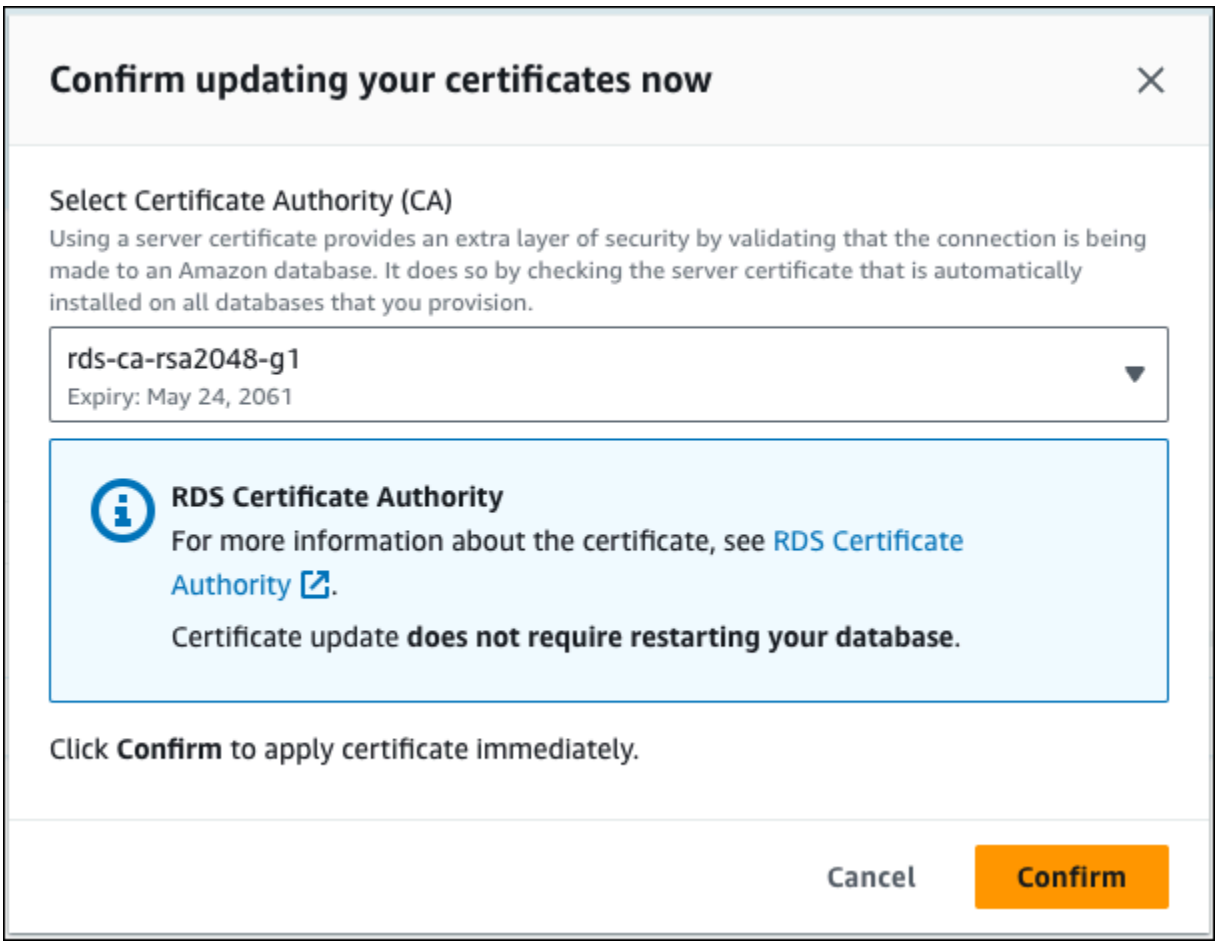
   您可以通过选择计划来计划下一维护时段的证书轮换。通过选择立即应用来立即应用轮换。

> **Important**
>
> 如果您在证书到期后遇到连接问题，请使用立即应用选项。

4. a. 如果您选择计划，系统会提示您确认 CA 证书轮换。此提示还说明更新的计划时段。



   b. 如果您选择立即应用，系统会提示您确认 CA 证书轮换。



> **Important**
>
> 在计划数据库上的 CA 证书轮换之前，请更新使用 SSL/TLS 和服务器证书进行连接的所有客户端应用程序。这些更新特定于您的数据库引擎。更新这些客户端应用程序后，可以确认 CA 证书轮换。

要继续，请选中该复选框，然后选择 Confirm (确认)。

5. 对要更新的每个数据库实例重复步骤 3 和 4。

自动服务器证书轮换

如果根 CA 支持自动服务器证书轮换，RDS 会自动处理数据库服务器证书的轮换。RDS 使用相同的根 CA 进行自动轮换，因此您无需下载新的 CA 服务包。请参阅证书颁发机构。

数据库服务器证书的轮换和有效期取决于您的数据库引擎：

- 如果您的数据库引擎支持无需重启即可轮换，则 RDS 会自动轮换数据库服务器证书，而无需您执行任何操作。RDS 尝试在您首选的维护时段中，在数据库服务器证书的半生命周期轮换您的数据库服务器证书。新的数据库服务器证书的有效期为 12 个月。
- 如果您的数据库引擎不支持无需重启即可轮换，则 RDS 会在数据库服务器证书到期前至少 6 个月通知您有关维护事件的信息。新的数据库服务器证书的有效期为 36 个月。

使用 describe-db-engine-versions 命令并检查 `SupportsCertificateRotationWithoutRestart`标志，以确定数据库引擎版本是否支持无需重启即可轮换证书。有关更多信息，请参阅 为数据库设置 CA。

将证书导入信任存储的示例脚本

以下是将证书捆绑包导入信任存储的示例 shell 脚本。

每个示例 Shell 脚本都使用 keytool，它是 Java 开发工具包 (JDK) 的一部分。有关安装 JDK 的信息，请参阅 JDK 安装指南。

Linux

下面是一个示例 Shell 脚本，它将证书捆绑包导入 Linux 操作系统上的信任存储。

```
mydir=tmp/certs
if [ ! -e "${mydir}" ]
then
mkdir -p "${mydir}"
fi truststore=${mydir}/rds-truststore.jks storepassword=changeit

curl -sS "https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem">
 ${mydir}/global-bundle.pem
awk 'split_after == 1 {n++;split_after=0} /-----END CERTIFICATE-----/
 {split_after=1}{print > "rds-ca-" n+1 ".pem"}' < ${mydir}/global-bundle.pem

for CERT in rds-ca-*; do alias=$(openssl x509 -noout -text -in $CERT | perl -ne
 'next unless /Subject:/; s/.*(CN=|CN = )//; print')
  echo "Importing $alias"
  keytool -import -file ${CERT} -alias "${alias}" -storepass ${storepassword} -
keystore ${truststore} -noprompt
  rm $CERT
done

rm ${mydir}/global-bundle.pem
```

```
echo "Trust store content is: "

keytool -list -v -keystore "$truststore" -storepass ${storepassword} | grep Alias |
 cut -d " " -f3- | while read alias
do expiry=`keytool -list -v -keystore "$truststore" -storepass ${storepassword} -
alias "${alias}" | grep Valid | perl -ne 'if(/until: (.*?)\n/) { print "$1\n"; }'`
   echo " Certificate ${alias} expires in '$expiry'"
done
```

macOS

下面是一个示例 Shell 脚本，它将证书捆绑包导入 macOS 上的信任存储。

```
mydir=tmp/certs
if [ ! -e "${mydir}" ]
then
mkdir -p "${mydir}"
fi truststore=${mydir}/rds-truststore.jks storepassword=changeit

curl -sS "https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem">
 ${mydir}/global-bundle.pem
split -p "-----BEGIN CERTIFICATE-----" ${mydir}/global-bundle.pem rds-ca-

for CERT in rds-ca-*; do alias=$(openssl x509 -noout -text -in $CERT | perl -ne
 'next unless /Subject:/; s/.*(CN=|CN = )//; print')
  echo "Importing $alias"
  keytool -import -file ${CERT} -alias "${alias}" -storepass ${storepassword} -
keystore ${truststore} -noprompt
  rm $CERT
done

rm ${mydir}/global-bundle.pem

echo "Trust store content is: "

keytool -list -v -keystore "$truststore" -storepass ${storepassword} | grep Alias |
 cut -d " " -f3- | while read alias
do expiry=`keytool -list -v -keystore "$truststore" -storepass ${storepassword} -
alias "${alias}" | grep Valid | perl -ne 'if(/until: (.*?)\n/) { print "$1\n"; }'`
   echo " Certificate ${alias} expires in '$expiry'"
done
```

要了解有关将 SSL 与 Amazon RDS 结合使用的更多最佳实践，请参阅 Best practices for successful SSL connections to Amazon RDS for Oracle。

互联网络流量隐私

Amazon Aurora 与本地应用程序之间以及 Amazon Aurora 与同一 AWS 区域内的其他 AWS 资源之间的连接都受到保护。

服务与本地客户端和应用之间的流量

在您的私有网络和 AWS 之间有两个连接选项：

- 一个AWS Site-to-Site VPN 连接。有关更多信息，请参阅什么是 AWS Site-to-Site VPN？
- AWS Direct Connect 连接。有关更多信息，请参阅什么是 AWS Direct Connect？

您可使用 AWS 发布的 API 操作，通过网络获取对 Amazon Aurora 的访问权限。客户端必须支持以下内容：

- 传输层安全性协议（TLS）。我们要求使用 TLS 1.2，建议使用 TLS 1.3。
- 具有完全向前保密（PFS）的密码套件，例如 DHE（临时 Diffie-Hellman）或 ECDHE（临时椭圆曲线 Diffie-Hellman）。大多数现代系统（如 Java 7 及更高版本）都支持这些模式。

此外，必须使用访问密钥 ID 和与 IAM 主体关联的秘密访问密钥来对请求进行签名。或者，您可以使用 AWS Security Token Service（AWS STS）生成临时安全凭证来对请求进行签名。

Amazon Aurora 的 Identity and Access Management

AWS Identity and Access Management（IAM）是一项 AWS 服务，可以帮助管理员安全地控制对 AWS 资源的访问。IAM 管理员控制谁可以通过身份验证（登录）和获得授权（具有权限）以使用 Amazon RDS 资源。IAM 是一项无需额外费用即可使用的 AWS 服务。

主题

- 受众
- 使用身份进行身份验证
- 使用策略管理访问
- Amazon Aurora 如何与 IAM 协同工作
- Amazon Aurora 的基于身份的策略示例
- 适用于 Amazon RDS 的 AWS 托管式策略
- AWS 托管式策略的 Amazon RDS 更新
- 防范跨服务混淆代理问题
- IAM 数据库身份验证
- 对 Amazon Aurora 身份和访问权限问题进行故障排除

受众

如何使用 AWS Identity and Access Management (IAM) 因您在 Amazon Aurora 中执行的操作而异。

服务用户 – 如果您使用 Aurora 服务来完成作业，则您的管理员会为您提供所需的凭证和权限。当您使用更多 Aurora 功能来完成工作时，您可能需要额外权限。了解如何管理访问权限可帮助您向管理员请求适合的权限。如果您无法访问 Aurora 中的一项功能，请参阅对 Amazon Aurora 身份和访问权限问题进行故障排除。

服务管理员 – 如果您在公司负责管理 Aurora 资源，则您可能具有 Aurora 的完全访问权限。您有责任确定您的员工应访问哪些 Aurora 功能和资源。然后，您必须向 管理员提交请求，这样才能更改您的服务用户的权限。请查看该页面上的信息以了解 IAM 的基本概念。要了解有关您的公司如何将 IAM 与 Aurora 搭配使用的更多信息，请参阅Amazon Aurora 如何与 IAM 协同工作。

管理员 – 如果您是管理员，您可能希望了解有关您可以如何编写策略以管理 Aurora 的访问权限的详细信息。要查看您可在 IAM 中使用的基于身份的 Aurora 示例策略，请参阅Amazon Aurora 的基于身份的策略示例。

使用身份进行身份验证

身份验证是您使用身份凭证登录 AWS 的方法。您必须作为 AWS 账户根用户、IAM 用户或通过代入 IAM 角色进行身份验证（登录到 AWS）。

您可以使用通过身份源提供的凭证以联合身份登录到 AWS。AWS IAM Identity Center（IAM Identity Center）用户、您公司的单点登录身份验证以及您的 Google 或 Facebook 凭证都是联合身份的示例。当您以联合身份登录时，您的管理员以前使用 IAM 角色设置了身份联合验证。当您使用联合身份验证访问 AWS 时，您就是在间接代入角色。

根据您的用户类型，您可以登录 AWS Management Console 或 AWS 访问门户。有关登录到 AWS 的更多信息，请参阅《AWS 登录 User Guide》中的 How to sign in to your AWS 账户。

如果您以编程方式访问 AWS，则 AWS 将提供软件开发工具包（SDK）和命令行界面（CLI），以便使用您的凭证以加密方式签署您的请求。如果您不使用 AWS 工具，则必须自行对请求签名。有关使用推荐的方法自行签署请求的更多信息，请参阅《IAM 用户指南》中的用于签署 API 请求的 AWS 签名版本 4。

无论使用何种身份验证方法，您都可能需要提供其他安全信息。例如，AWS 建议您使用多重身份验证（MFA）来提高账户的安全性。要了解更多信息，请参阅《AWS IAM Identity Center 用户指南》中的多重身份验证和《IAM 用户指南》中的 IAM 中的 AWS多重身份验证。

AWS 账户根用户

当您创建 AWS 账户时，最初使用的是一个对账户中所有 AWS 服务和资源拥有完全访问权限的登录身份。此身份称为 AWS 账户根用户，使用您创建账户时所用的电子邮件地址和密码登录，即可获得该身份。强烈建议您不要使用根用户执行日常任务。保护好根用户凭证，并使用这些凭证来执行仅根用户可以执行的任务。有关要求您以根用户身份登录的任务的完整列表，请参阅 IAM 用户指南中的需要根用户凭证的任务。

联合身份

作为最佳实践，要求人类用户（包括需要管理员访问权限的用户）结合使用联合身份验证和身份提供程序，以使用临时凭证来访问 AWS 服务。

联合身份是来自企业用户目录、Web 身份提供程序、AWS Directory Service、Identity Center 目录的用户，或任何使用通过身份源提供的凭证来访问 AWS 服务的用户。当联合身份访问 AWS 账户时，他们代入角色，而角色提供临时凭证。

要集中管理访问权限，建议您使用 AWS IAM Identity Center。您可以在 IAM Identity Center 中创建用户和组，也可以连接并同步到您自己的身份源中的一组用户和组以跨所有 AWS 账户和应用程序使用。

有关 IAM Identity Center 的信息，请参阅 AWS IAM Identity Center 用户指南中的什么是 IAM Identity Center？。

IAM 用户和群组

IAM 用户是 AWS 账户内对某个人员或应用程序具有特定权限的一个身份。在可能的情况下，我们建议使用临时凭证，而不是创建具有长期凭证（如密码和访问密钥）的 IAM 用户。但是，如果您有一些特定的使用场景需要长期凭证以及 IAM 用户，建议您轮换访问密钥。有关更多信息，请参阅《IAM 用户指南》中的对于需要长期凭证的用例，应在需要时更新访问密钥。

IAM 组是一个指定一组 IAM 用户的身份。您不能使用组的身份登录。您可以使用组来一次性为多个用户指定权限。如果有大量用户，使用组可以更轻松地管理用户权限。例如，您可能具有一个名为 IAMAdmins 的组，并为该组授予权限以管理 IAM 资源。

用户与角色不同。用户唯一地与某个人员或应用程序关联，而角色旨在让需要它的任何人代入。用户具有永久的长期凭证，而角色提供临时凭证。要了解更多信息，请参阅《IAM 用户指南》中的 IAM 用户使用案例。

可以使用 IAM 数据库身份验证对数据库集群进行身份验证。

IAM 数据库身份验证使用 Aurora。有关使用 IAM 对数据库集群进行身份验证的更多信息，请参阅 IAM 数据库身份验证。

IAM 角色

IAM 角色是 AWS 账户中具有特定权限的身份。它类似于用户，但未与特定人员关联。您可以通过切换角色，在 AWS Management Console 中暂时代入 IAM 角色。您可以调用 AWS CLI 或 AWS API 操作或使用自定义网址以担任角色。有关使用角色的方法的更多信息，请参阅《IAM 用户指南》中的使用 IAM 角色。

具有临时凭证的 IAM 角色在以下情况下很有用：

- 临时用户权限 – 用户可代入 IAM 角色，暂时获得针对特定任务的不同权限。
- 联合用户访问：要向联合身份分配权限，请创建角色并为角色定义权限。当联合身份进行身份验证时，该身份将与角色相关联并被授予由此角色定义的权限。有关用于联合身份验证的角色的信息，请参阅《IAM 用户指南》中的针对第三方身份提供商创建角色（联合身份验证）。如果您使用 IAM Identity Center，则需要配置权限集。为控制您的身份在进行身份验证后可以访问的内容，IAM Identity Center 将权限集与 IAM 中的角色相关联。有关权限集的信息，请参阅 AWS IAM Identity Center 用户指南中的权限集。

- 跨账户存取：您可以使用 IAM 角色以允许不同账户中的某个人（可信主体）访问您的账户中的资源。角色是授予跨账户访问权限的主要方式。但是，对于某些 AWS 服务，您可以将策略直接附加到资源（而不是使用角色作为代理）。要了解用于跨账户访问的角色和基于资源的策略之间的差别，请参阅《IAM 用户指南》中的 IAM 角色与基于资源的策略有何不同。
- 跨服务访问：某些 AWS 服务使用其它 AWS 服务中的特征。例如，当您在某个服务中进行调用时，该服务通常会在 Amazon EC2 中运行应用程序或在 Simple Storage Service（Amazon S3）中存储对象。服务可能会使用发出调用的主体的权限、使用服务角色或使用服务相关角色来执行此操作。
  - 转发访问会话：当您使用 IAM 用户或角色在 AWS 中执行操作时，您将被视为主体。使用某些服务时，您可能会执行一个操作，然后此操作在其他服务中启动另一个操作。FAS 使用主体调用 AWS 服务的权限，结合请求的 AWS 服务，向下游服务发出请求。只有在服务收到需要与其他 AWS 服务或资源交互才能完成的请求时，才会发出 FAS 请求。在这种情况下，您必须具有执行这两项操作的权限。有关发出 FAS 请求时的策略详情，请参阅转发访问会话。
  - 服务角色 - 服务角色是服务代表您在您的账户中执行操作而分派的 IAM 角色。IAM 管理员可以在 IAM 中创建、修改和删除服务角色。有关更多信息，请参阅《IAM 用户指南》中的创建向 AWS 服务委派权限的角色。
  - 服务相关角色：服务相关角色是与 AWS 服务 关联的一种服务角色。服务可以代入代表您执行操作的角色。服务相关角色显示在您的 AWS 账户 中，并由该服务拥有。IAM 管理员可以查看但不能编辑服务相关角色的权限。
- 在 Amazon EC2 上运行的应用程序：您可以使用 IAM 角色管理在 EC2 实例上运行并发出 AWS CLI 或 AWS API 请求的应用程序的临时凭证。这优先于在 EC2 实例中存储访问密钥。要将 AWS 角色分配给 EC2 实例并使其对该实例的所有应用程序可用，您可以创建一个附加到实例的实例配置文件。实例配置文件包含角色，并使 EC2 实例上运行的程序能够获得临时凭证。有关更多信息，请参阅《IAM 用户指南》中的使用 IAM 角色向在 Amazon EC2 实例上运行的应用程序授予权限。

要了解是否使用 IAM 角色，请参阅 IAM 用户指南 中的何时创建 IAM 角色（而不是用户）。

使用策略管理访问

您将创建策略并将其附加到 IAM 身份或AWS资源，以便控制AWS中的访问。策略是 AWS 中的对象；在与身份或资源相关联时，策略定义其权限。在某个实体（根用户、用户或 IAM 角色）发出请求时，AWS 将评估这些策略。策略中的权限确定是允许还是拒绝请求。大多数策略在 AWS 中存储为 JSON 文档。有关 JSON 策略文档的结构和内容的更多信息，请参阅 IAM 用户指南中的 JSON 策略概览。

管理员可以使用策略来指定哪些用户有权访问 AWS 资源，以及他们可以对这些资源执行哪些操作。每个 IAM 实体（权限集或角色）最初没有任何权限。换言之，预设情况下，用户什么都不能做，甚至不

能更改他们自己的密码。要为用户授予执行某些操作的权限，管理员必须将权限策略附加到用户。或者，管理员可以将用户添加到具有预期权限的组中。当管理员为某个组授予访问权限时，该组内的全部用户都会获得这些访问权限。

IAM 策略定义操作的权限，无关乎您使用哪种方法执行操作。例如，假设您有一个允许 `iam:GetRole` 操作的策略。具有该策略的用户可以从 AWS Management Console、AWS CLI 或 AWS API 获取角色信息。

基于身份的策略

基于身份的策略是可附加到身份（如权限集或角色）的 JSON 权限策略文档。这些策略控制身份可在何种条件下对哪些资源执行哪些操作。要了解如何创建基于身份的策略，请参阅 IAM 用户指南中的创建 IAM policy。

基于身份的策略可以进一步归类为内联策略或托管策略。内联策略直接嵌入到单个权限集或角色中。托管式策略是可以附加到 AWS 账户中的多个权限集和角色的独立策略。托管式策略包括 AWS 托管式策略和客户管理型策略。要了解如何在托管式策略和内联策略之间进行选择，请参阅 IAM 用户指南中的在托管式策略与内联策略之间进行选择。

有关特定于 Amazon Aurora 的 AWS 托管式策略的信息，请参阅适用于 Amazon RDS 的 AWS 托管式策略。

其他策略类型

AWS 支持额外的、不太常用的策略类型。这些策略类型可以设置更常用的策略类型向您授予的最大权限。

- 权限边界 – 权限边界是一项高级功能，借助该功能，您可以设置基于身份的策略可以授予 IAM 实体（权限集或角色）的最大权限。您可为实体设置权限边界。这些结果权限是实体的基于身份的策略及其权限边界的交集。在 `Principal` 字段中指定权限集或角色的基于资源的策略不受权限边界限制。任一项策略中的显式拒绝将覆盖允许。有关权限边界的更多信息，请参阅 IAM 用户指南中的 IAM 实体的权限边界。
- 服务控制策略 (SCP) – SCP 是 JSON 策略，指定了组织或组织单位 (OU) 在 AWS Organizations 中的最大权限。AWS Organizations 是一项服务，用于分组和集中管理您的企业拥有的多个 AWS 账户。如果在组织内启用了所有特征，则可对任意或全部账户应用服务控制策略（SCP）。SCP 限制成员账户中实体（包括每个 AWS 账户根用户）的权限。有关 Organizations 和 SCP 的更多信息，请参阅《AWS Organizations 用户指南》中的 SCP 的工作原理。
- 会话策略 – 会话策略是当您以编程方式为角色或联合用户创建临时会话时作为参数传递的高级策略。结果会话的权限是权限集或角色的基于身份的策略和会话策略的交集。权限也可以来自基于资

源的策略。任一项策略中的显式拒绝将覆盖允许。有关更多信息，请参阅 IAM 用户指南中的会话策略。

多个策略类型

当多个类型的策略应用于一个请求时，生成的权限更加复杂和难以理解。要了解 AWS 如何确定在涉及多种策略类型时是否允许请求，请参阅 IAM 用户指南中的策略评测逻辑。

Amazon Aurora 如何与 IAM 协同工作

在使用 IAM 管理对 Amazon Aurora 的访问之前，您应了解哪些 IAM 功能可与 Aurora 协同工作。

下表列出了可与 Amazon Aurora 结合使用的 IAM 功能：

IAM 特征	Amazon Aurora 支持
基于身份的策略	是
基于资源的策略	否
策略操作	是
策略资源	是
策略条件键（特定于服务）	是
ACL	否
基于属性的访问控制 (ABAC)（策略中的标签）	是
临时凭证	是
转发访问会话	是
服务角色	是
服务相关角色	是

要大致了解 Amazon Aurora 和其他 AWS 服务如何与 IAM 一起使用，请参阅《IAM 用户指南》中的与 IAM 一起使用的 AWS 服务。

主题

- Aurora 基于身份的策略
- Aurora 内基于资源的策略
- Aurora 的策略操作
- Aurora 的策略资源
- Aurora 的策略条件键
- Aurora 中的访问控制列表 (ACL)
- 策略中具有 Aurora 标签的基于属性的访问控制 (ABAC)
- 将临时凭证用于 Aurora
- Aurora 的转发访问会话
- Aurora 的服务角色
- Aurora 的服务相关角色

Aurora 基于身份的策略

支持基于身份的策略：是。

基于身份的策略是可附加到身份（如 IAM 用户、用户组或角色）的 JSON 权限策略文档。这些策略控制用户和角色可在何种条件下对哪些资源执行哪些操作。要了解如何创建基于身份的策略，请参阅《IAM 用户指南》中的使用客户托管策略定义自定义 IAM 权限。

通过使用 IAM 基于身份的策略，您可以指定允许或拒绝的操作和资源以及允许或拒绝操作的条件。您无法在基于身份的策略中指定主体，因为它适用于其附加的用户或角色。要了解可在 JSON 策略中使用的所有元素，请参阅《IAM 用户指南》中的 IAM JSON 策略元素引用。

Aurora 基于身份的策略示例

要查看 Aurora 基于身份的策略的示例，请参阅 Amazon Aurora 的基于身份的策略示例。

Aurora 内基于资源的策略

支持基于资源的策略：否。

基于资源的策略是附加到资源的 JSON 策略文档。基于资源的策略的示例包括 IAM 角色信任策略和 Amazon S3 存储桶策略。在支持基于资源的策略的服务中，服务管理员可以使用它们来控制对特定资源的访问。对于在其中附加策略的资源，策略定义指定主体可以对该资源执行哪些操作以及在什么条件

下执行。您必须在基于资源的策略中指定主体。主体可以包括账户、用户、角色、联合用户或 AWS 服务。

要启用跨账户访问，您可以将整个账户或其他账户中的 IAM 实体指定为基于资源的策略中的主体。将跨账户主体添加到基于资源的策略只是建立信任关系工作的一半而已。当主体和资源处于不同的 AWS 账户 中时，则信任账户中的 IAM 管理员还必须授予主体实体（用户或角色）对资源的访问权限。他们通过将基于身份的策略附加到实体以授予权限。但是，如果基于资源的策略向同一个账户中的主体授予访问权限，则不需要额外的基于身份的策略。有关更多信息，请参阅《IAM 用户指南》中的 IAM 中的跨账户资源访问。

Aurora 的策略操作

支持策略操作：是。

管理员可以使用 AWS JSON 策略来指定谁有权访问什么内容。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

JSON 策略的 `Action` 元素描述可用于在策略中允许或拒绝访问的操作。策略操作通常与关联的 AWS API 操作同名。有一些例外情况，例如没有匹配 API 操作的仅限权限 操作。还有一些操作需要在策略中执行多个操作。这些附加操作称为相关操作。

在策略中包含操作以授予执行关联操作的权限。

Aurora 中的策略操作在操作前使用以下前缀：`rds:`。例如，要授予某人使用 Amazon RDS `DescribeDBInstances` API 操作妙手数据库实例的权限，您应将 `rds:DescribeDBInstances` 操作纳入其策略中。策略语句必须包含 `Action` 或 `NotAction` 元素。Aurora 定义了一组自己的操作，以描述您可以使用该服务执行的任务。

要在单个语句中指定多项 操作，请使用逗号将它们隔开，如下所示。

```
"Action": [
      "rds:action1",
      "rds:action2"
```

您也可以使用通配符 (*) 指定多个操作。例如，要指定以单词 `Describe` 开头的所有操作，请包括以下操作。

```
"Action": "rds:Describe*"
```

有关 Aurora 操作的列表，请参阅服务授权参考中的 Amazon RDS 定义的操作。

Aurora 的策略资源

支持策略资源：是。

管理员可以使用 AWS JSON 策略来指定谁有权访问什么内容。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

Resource JSON 策略元素指定要向其应用操作的一个或多个对象。语句必须包含 Resource 或 NotResource 元素。作为最佳实践，请使用其 Amazon 资源名称（ARN）指定资源。对于支持特定资源类型（称为资源级权限）的操作，您可以执行此操作。

对于不支持资源级权限的操作（如列出操作），请使用通配符（*）指示语句应用于所有资源。

```
"Resource": "*"
```

数据库实例资源具有以下 Amazon 资源名称 (ARN)。

```
arn:${Partition}:rds:${Region}:${Account}:{ResourceType}/${Resource}
```

有关 ARN 格式的更多信息，请参阅 Amazon Resource Name (ARN) 和 AWS 服务命名空间。

例如，要在语句中指定 dbtest 数据库实例，请使用以下 ARN。

```
"Resource": "arn:aws:rds:us-west-2:123456789012:db:dbtest"
```

要指定属于特定账户的所有数据库实例，请使用通配符 (*)。

```
"Resource": "arn:aws:rds:us-east-1:123456789012:db:*"
```

某些 RDS API 操作（例如，用于创建资源的那些操作）无法在特定资源上执行。在这些情况下，请使用通配符 (*)。

```
"Resource": "*"
```

许多 Amazon RDS API 操作涉及多种资源。例如，CreateDBInstance 创建数据库实例。您可以指定，在创建数据库实例时， 用户必须使用特定的安全组和参数组。要在单个语句中指定多个资源，请使用逗号分隔 ARN。

```
"Resource": [
```

```
        "resource1",
        "resource2"
```

有关 Aurora 资源类型及其 ARN 的列表，请参阅服务授权参考中的 Amazon RDS 定义的操作。要了解您可以在哪些操作中指定每个资源的 ARN，请参阅 Amazon RDS 定义的操作。

Aurora 的策略条件键

支持特定于服务的策略条件键：是。

管理员可以使用 AWS JSON 策略来指定谁有权访问什么内容。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

在 Condition 元素（或 Condition 块）中，可以指定语句生效的条件。Condition 元素是可选的。您可以创建使用条件运算符（例如，等于或小于）的条件表达式，以使策略中的条件与请求中的值相匹配。

如果您在一个语句中指定多个 Condition 元素，或在单个 Condition 元素中指定多个键，则 AWS 使用逻辑 AND 运算评估它们。如果您为单个条件键指定多个值，则 AWS 使用逻辑 OR 运算来评估条件。在授予语句的权限之前必须满足所有的条件。

在指定条件时，您也可以使用占位符变量。例如，只有在使用 IAM 用户名标记 IAM 用户时，您才能为其授予访问资源的权限。有关更多信息，请参阅《IAM 用户指南》中的 IAM 策略元素：变量和标签。

AWS 支持全局条件键和特定于服务的条件键。要查看所有 AWS 全局条件键，请参阅《IAM 用户指南》中的 AWS 全局条件上下文键。

Aurora 定义了自己的一组条件键，还支持使用一些全局条件键。要查看所有 AWS 全局条件键，请参阅 IAM 用户指南中的 AWS 全局条件上下文键 。

所有 RDS API 操作都支持 aws:RequestedRegion 条件键。

有关 Aurora 条件键的列表，请参阅服务授权参考中的 Amazon RDS 的条件键。要了解您可以对哪些操作和资源使用条件键，请参阅 Amazon RDS 定义的操作。

Aurora 中的访问控制列表 (ACL)

支持访问控制列表（ACL）：否

访问控制列表（ACL）控制哪些主体（账户成员、用户或角色）有权访问资源。ACL 与基于资源的策略类似，但它们不使用 JSON 策略文档格式。

策略中具有 Aurora 标签的基于属性的访问控制 (ABAC)

在策略中支持基于属性的访问权限控制（ABAC）标签：是

基于属性的访问控制（ABAC）是一种授权策略，该策略基于属性来定义权限。在 AWS 中，这些属性称为标签。您可以将标签附加到 IAM 实体（用户或角色）以及 AWS 资源。标记实体和资源是 ABAC 的第一步。然后设计 ABAC 策略，以在主体的标签与他们尝试访问的资源标签匹配时允许操作。

ABAC 在快速增长的环境中非常有用，并在策略管理变得繁琐的情况下可以提供帮助。

要基于标签控制访问，您需要使用 `aws:ResourceTag/key-name`、`aws:RequestTag/key-name` 或 `aws:TagKeys` 条件键在策略的条件元素中提供标签信息。

如果某个服务对于每种资源类型都支持所有这三个条件键，则对于该服务，该值为是。如果某个服务仅对于部分资源类型支持所有这三个条件键，则该值为部分。

有关 ABAC 的更多信息，请参阅《IAM 用户指南》中的使用 ABAC 授权定义权限。要查看设置 ABAC 步骤的教程，请参阅《IAM 用户指南》中的使用基于属性的访问权限控制（ABAC）。

有关标记 Aurora 资源的更多信息，请参阅指定条件：使用自定义标签。要查看基于身份的策略（用于根据资源上的标签来限制对该资源的访问）的示例，请参阅授予权限以允许对在特定标签中包含两个不同值的资源执行操作。

将临时凭证用于 Aurora

支持临时凭证：是。

某些 AWS 服务 在您使用临时凭证登录时无法正常工作。有关更多信息，包括 AWS 服务 与临时凭证配合使用，请参阅 IAM 用户指南中的使用 IAM 的 AWS 服务。

如果您不使用用户名和密码而用其他方法登录到 AWS Management Console，则使用临时凭证。例如，当您使用贵公司的单点登录（SSO）链接访问 AWS 时，该过程将自动创建临时凭证。当您以用户身份登录控制台，然后切换角色时，您还会自动创建临时凭证。有关切换角色的更多信息，请参阅《IAM 用户指南》中的从用户切换到 IAM 角色（控制台）。

您可以使用 AWS CLI 或者 AWS API 创建临时凭证。之后，您可以使用这些临时凭证访问 AWS。AWS 建议您动态生成临时凭证，而不是使用长期访问密钥。有关更多信息，请参阅 IAM 中的临时安全凭证。

Aurora 的转发访问会话

支持转发访问会话：是。

当您使用 IAM 用户或角色在 AWS 中执行操作时，您将被视为主体。使用某些服务时，您可能会执行一个操作，然后此操作在其他服务中启动另一个操作。FAS 使用主体调用 AWS 服务的权限，结合请求的 AWS 服务，向下游服务发出请求。只有在服务收到需要与其他 AWS 服务或资源交互才能完成的请求时，才会发出 FAS 请求。在这种情况下，您必须具有执行这两项操作的权限。有关发出 FAS 请求时的策略详情，请参阅转发访问会话。

Aurora 的服务角色

支持服务角色：是。

服务角色是由一项服务担任、代表您执行操作的 IAM 角色。IAM 管理员可以在 IAM 中创建、修改和删除服务角色。有关更多信息，请参阅《IAM 用户指南》中的创建向 AWS 服务委派权限的角色。

> **Warning**
>
> 更改服务角色的权限可能会破坏 Aurora 功能。仅当 Aurora 提供相关指导时才编辑服务角色。

Aurora 的服务相关角色

支持服务相关角色：是。

服务相关角色是一种与 AWS 服务 相关的服务角色。服务可以代入代表您执行操作的角色。服务相关角色显示在您的 AWS 账户 中，并由该服务拥有。IAM 管理员可以查看但不能编辑服务相关角色的权限。

有关使用 Aurora 服务相关角色的详细信息，请参阅将服务相关角色用于 Amazon Aurora。

Amazon Aurora 的基于身份的策略示例

原定设置情况下，权限集和角色没有创建或修改 Aurora 资源的权限。它们还无法使用 AWS Management Console、AWS CLI 或 AWS API 执行任务。管理员必须创建 IAM policy，以便为权限集和角色授予权限以对所需的指定资源执行特定的 API 操作。然后，管理员必须将这些策略附加到需要这些权限的权限集或角色。

要了解如何使用这些示例 JSON 策略文档创建 IAM 基于身份的策略，请参阅《IAM 用户指南》中的在 JSON 选项卡上创建策略。

主题

- 策略最佳实践
- 使用 Aurora 控制台
- 使用控制台所需的权限
- 允许用户查看他们自己的权限
- 在 Aurora 中创建、修改和删除资源的权限策略
- 示例策略：使用条件键
- 指定条件：使用自定义标签
- 在创建过程中授予为 Aurora 资源添加标签的权限

策略最佳实践

基于身份的策略确定某个人是否可以创建、访问或删除您账户中的 Amazon RDS 资源。这些操作可能会使 AWS 账户 产生成本。创建或编辑基于身份的策略时，请遵循以下指南和建议：

- AWS 托管式策略及转向最低权限许可入门 – 要开始向用户和工作负载授予权限，请使用 AWS 托管式策略来为许多常见使用场景授予权限。您可以在 AWS 账户 中找到这些策略。我们建议通过定义特定于您的使用场景的 AWS 客户托管式策略来进一步减少权限。有关更多信息，请参阅《IAM 用户指南》中的 AWS 托管式策略或工作职能的 AWS 托管式策略。
- 应用最低权限：在使用 IAM 策略设置权限时，请仅授予执行任务所需的权限。为此，您可以定义在特定条件下可以对特定资源执行的操作，也称为最低权限许可。有关使用 IAM 应用权限的更多信息，请参阅《IAM 用户指南》中的 IAM 中的策略和权限。
- 使用 IAM 策略中的条件进一步限制访问权限：您可以向策略添加条件来限制对操作和资源的访问。例如，您可以编写策略条件来指定必须使用 SSL 发送所有请求。如果通过特定 AWS 服务（例如 AWS CloudFormation）使用服务操作，您还可以使用条件来授予对服务操作的访问权限。有关更多信息，请参阅《IAM 用户指南》中的 IAM JSON 策略元素：条件。
- 使用 IAM Access Analyzer 验证您的 IAM 策略，以确保权限的安全性和功能性 – IAM Access Analyzer 会验证新策略和现有策略，以确保策略符合 IAM 策略语言（JSON）和 IAM 最佳实践。IAM Access Analyzer 提供 100 多项策略检查和可操作的建议，以帮助您制定安全且功能性强的策略。有关更多信息，请参阅《IAM 用户指南》中的使用 IAM Access Analyzer 验证策略。
- 需要多重身份验证（MFA）：如果您所处的场景要求您的 AWS 账户 中有 IAM 用户或根用户，请启用 MFA 来提高安全性。若要在调用 API 操作时需要 MFA，请将 MFA 条件添加到您的策略中。有关更多信息，请参阅《IAM 用户指南》中的使用 MFA 保护 API 访问。

有关 IAM 中的最佳实操的更多信息，请参阅《IAM 用户指南》中的 IAM 中的安全最佳实践。

使用 Aurora 控制台

要访问 Amazon Aurora 控制台，您必须拥有一组最低的权限。这些权限必须允许您列出和查看有关您的 AWS 账户中的 Amazon Aurora 资源的详细信息。如果创建比必需的最低权限更为严格的基于身份的策略，对于附加了该策略的实体（用户或角色），控制台将无法按预期正常运行。

对于只需要调用 AWS CLI 或 AWS API 的用户，无需为其提供最低控制台权限。相反，只允许访问与您尝试执行的 API 操作相匹配的操作。

要确保这些实体仍可使用 Aurora 控制台，也可向实体附加以下AWS托管策略。

```
AmazonRDSReadOnlyAccess
```

有关更多信息，请参阅 IAM 用户指南 中的为用户添加权限。

使用控制台所需的权限

对于要使用控制台的用户，该用户必须拥有一组最小权限。这些权限允许用户描述其AWS账户的 Amazon Aurora 资源并提供其他相关信息（包括 Amazon EC2 安全和网络信息）。

如果创建比必需的最低权限更为严格的 IAM 策略，对于附加了该 IAM 策略的用户，控制台无法按预期正常运行。要确保这些用户仍可使用控制台，也可向用户附加 `AmazonRDSReadOnlyAccess` 托管策略，如使用策略管理访问中所述。

对于只需要调用 AWS CLI 或 Amazon RDS API 的用户，无需为其提供最低限度的控制台权限。

以下策略授予对AWS根账户的所有 Amazon Aurora 资源的完全访问权：

```
AmazonRDSFullAccess
```

允许用户查看他们自己的权限

该示例说明了您如何创建策略，以允许 IAM 用户查看附加到其用户身份的内联和托管式策略。此策略包括在控制台上完成此操作或者以编程方式使用 AWS CLI 或 AWS API 所需的权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
```

```
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

在 Aurora 中创建、修改和删除资源的权限策略

以下各节展示了授予和限制资源访问权限的权限策略示例：

允许用户在 AWS 账户中创建数据库实例

以下是支持 ID 为 123456789012 的账户为您的 AWS 账户创建数据库实例的示例策略。该策略要求新数据库实例的名称以 test 开头。新数据库实例还必须使用 MySQL 数据库引擎和 db.t2.micro 数据库实例类。此外，新数据库实例必须使用以 default 开头的选项组和数据库参数组，并且它必须使用 default 子网组。

JSON

```
{
```

```
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "AllowCreateDBInstanceOnly",
         "Effect": "Allow",
         "Action": [
            "rds:CreateDBInstance"
         ],
         "Resource": [
            "arn:aws:rds:*:123456789012:db:test*",
            "arn:aws:rds:*:123456789012:og:default*",
            "arn:aws:rds:*:123456789012:pg:default*",
            "arn:aws:rds:*:123456789012:subgrp:default"
         ],
         "Condition": {
            "StringEquals": {
               "rds:DatabaseEngine": "mysql",
               "rds:DatabaseClass": "db.t2.micro"
            }
         }
      }
   ]
}
```

此策略包含一个为 用户指定以下权限的语句：

- 该策略可让该账户使用 CreateDBInstance API 操作创建数据库实例（这还适用于 create-db-instance AWS CLI 命令和 AWS Management Console）。
- `Resource` 元素指定用户可以执行操作的资源。使用 Amazon Resource Name (ARN) 指定资源。此 ARN 包括资源所属服务的名称（`rds`）、AWS 区域（在该示例中，* 指示任何区域）、AWS 账号（在该示例中，123456789012 为账号）以及资源的类型。有关创建 ARN 的更多信息，请参阅Amazon RDS 中的 Amazon 资源名称（ARN）。

  该示例中的 `Resource` 元素为用户指定有关资源的以下策略限制：

  - 新数据库实例的数据库实例标识符必须以 `test` 开头 (例如，`testCustomerData1`、`test-region2-data`)。
  - 新数据库实例的选项组必须以 `default` 开头。
  - 新数据库实例的数据库参数组必须以 `default` 开头。
  - 新数据库实例的子网组必须是 `default` 子网组。

- Condition 元素指定数据库引擎必须是 MySQL 并且数据库实例类必须是 db.t2.micro。Condition 元素指定策略生效的条件。您可以通过使用 Condition 元素添加其他权限或限制。有关指定条件的更多信息，请参阅Aurora 的策略条件键。此示例指定 rds:DatabaseEngine 和 rds:DatabaseClass 条件。有关 rds:DatabaseEngine 的有效条件值的信息，请参阅 CreateDBInstance 中的 Engine 参数下的列表。有关 rds:DatabaseClass 的有效条件值的信息，请参阅 数据库实例类支持的数据库引擎。

该策略不指定 Principal 元素，因为在基于身份的策略中，您未指定获取权限的委托人。附加了策略的用户是隐式委托人。向 IAM 角色附加权限策略后，该角色的信任策略中标识的委托人将获取权限。

有关 Aurora 操作的列表，请参阅服务授权参考中的 Amazon RDS 定义的操作。

允许用户对任何 RDS 资源执行任何 Describe 操作

以下权限策略对用户授予权限以运行以 Describe 开头的所有操作。这些操作显示有关 RDS 资源 (如数据库实例) 的信息。Resource 元素中的通配符 (*) 表示可对账户拥有的所有 Amazon Aurora 资源执行操作。

JSON

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "AllowRDSDescribe",
         "Effect": "Allow",
         "Action": "rds:Describe*",
         "Resource": "*"
      }
   ]
}
```

允许用户创建使用指定数据库参数组和子网组的数据库实例

以下权限策略授予权限以允许用户仅创建必须使用 mydbpg 数据库参数组和 mydbsubnetgroup 数据库子网组的数据库实例。

JSON

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "VisualEditor0",
         "Effect": "Allow",
         "Action": "rds:CreateDBInstance",
         "Resource": [
            "arn:aws:rds:*:*:pg:mydbpg",
            "arn:aws:rds:*:*:subgrp:mydbsubnetgroup"
         ]
      }
   ]
}
```

授予权限以允许对在特定标签中包含两个不同值的资源执行操作

您可以在基于身份的策略中使用条件，以便基于标签控制对 Aurora 资源的访问。以下策略所授予的权限允许对 `stage` 标签设置为 `development` 或 `test` 的实例执行 `CreateDBSnapshot` API 操作。

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"AllowAnySnapshotName",
         "Effect":"Allow",
         "Action":[
            "rds:CreateDBSnapshot"
         ],
         "Resource":"arn:aws:rds:*:123456789012:snapshot:*"
      },
      {
         "Sid":"AllowDevTestToCreateSnapshot",
         "Effect":"Allow",
         "Action":[
            "rds:CreateDBSnapshot"
```

```
         ],
         "Resource":"arn:aws:rds:*:123456789012:db:*",
         "Condition":{
            "StringEquals":{
                "rds:db-tag/stage":[
                  "development",
                  "test"
               ]
            }
         }
      }
   ]
}
```

以下策略所授予的权限允许对 stage 标签设置为 development 或 test 的实例执行 ModifyDBInstance API 操作。

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"AllowChangingParameterOptionSecurityGroups",
         "Effect":"Allow",
         "Action":[
            "rds:ModifyDBInstance"
         ],
         "Resource": [
            "arn:aws:rds:*:123456789012:pg:*",
            "arn:aws:rds:*:123456789012:secgrp:*",
            "arn:aws:rds:*:123456789012:og:*"
         ]
      },
      {
         "Sid":"AllowDevTestToModifyInstance",
         "Effect":"Allow",
         "Action":[
            "rds:ModifyDBInstance"
         ],
         "Resource":"arn:aws:rds:*:123456789012:db:*",
```

```
         "Condition":{
            "StringEquals":{
                "rds:db-tag/stage":[
                  "development",
                  "test"
               ]
            }
         }
      }
   ]
}
```

防止用户删除数据库实例

以下权限策略授予权限以防止用户删除特定数据库实例。例如，您可能想禁止任何非管理员用户删除您的生产数据库实例。

JSON

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "DenyDelete1",
         "Effect": "Deny",
         "Action": "rds:DeleteDBInstance",
         "Resource": "arn:aws:rds:us-west-2:123456789012:db:my-mysql-instance"
      }
   ]
}
```

拒绝对资源的所有访问

您可以明确拒绝对资源的访问。拒绝策略优先于允许策略。以下策略明确拒绝用户管理资源的能力：

JSON

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Deny",
         "Action": "rds:*",
         "Resource": "arn:aws:rds:us-east-1:123456789012:db:mydb"
      }
   ]
}
```

示例策略：使用条件键

以下示例说明了如何在 Amazon Aurora IAM 权限策略中使用条件键。

示例 1：授予权限以创建使用特定数据库引擎的非多可用区数据库实例

以下策略使用 RDS 条件键，并仅允许用户创建采用 MySQL 数据库引擎且不使用多可用区的数据库实例。Condition 元素指示数据库引擎须为 MySQL 的要求。

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "AllowMySQLCreate",
         "Effect": "Allow",
         "Action": "rds:CreateDBInstance",
         "Resource": "*",
         "Condition": {
            "StringEquals": {
               "rds:DatabaseEngine": "mysql"
            },
            "Bool": {
               "rds:MultiAz": false
            }
         }
      }
   ]
}
```

示例 2：明确拒绝权限，以禁止创建特定数据库实例类的数据库实例和使用预置 IOPS 的数据库实例

以下策略显式拒绝创建使用数据库实例类 r3.8xlarge 和 m4.10xlarge（最大、最贵的数据库实例类）的数据库实例的权限。此策略还禁止用户创建使用预置的 IOPS (这会带来额外成本) 的数据库实例。

显式拒绝权限会取代授予的任何其他权限。这可确保用户身份不会无意中获得您绝不希望授予的权限。

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "DenyLargeCreate",
         "Effect": "Deny",
         "Action": "rds:CreateDBInstance",
         "Resource": "*",
         "Condition": {
            "StringEquals": {
               "rds:DatabaseClass": [
                  "db.r3.8xlarge",
                  "db.m4.10xlarge"
               ]
            }
         }
      },
      {
         "Sid": "DenyPIOPSCreate",
         "Effect": "Deny",
         "Action": "rds:CreateDBInstance",
         "Resource": "*",
         "Condition": {
            "NumericNotEquals": {
               "rds:Piops": "0"
            }
         }
      }
   ]
}
```

示例 3：限制可用于对资源进行标记的一组标签键和值的值

下面的策略使用 RDS 条件键，并允许将键为 stage 的标签添加到值为 test、qa 和 production 的资源。

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "rds:AddTagsToResource",
            "rds:RemoveTagsFromResource"
         ],
         "Resource": "*",
         "Condition": {
            "streq": {
               "rds:req-tag/stage": [
                  "test",
                  "qa",
                  "production"
               ]
            }
         }
      }
   ]
}
```

指定条件：使用自定义标签

Amazon Aurora 支持在 IAM 策略中使用自定义标签指定条件。

例如，假定您将一个名为 `environment` 的标签添加到具有 `beta`、`staging`、`production` 等值的数据库实例。如果您这样做，则可创建一个策略来根据 `environment` 标签值以仅允许某些用户使用数据库实例。

> **Note**
>
> 自定义标签标识符区分大小写。

下表列出了可以在 `Condition` 元素中使用的 RDS 标签标识符。

RDS 标签标识符	适用于
`db-tag`	数据库实例，包括只读副本

RDS 标签标识符	适用于
snapshot-tag	数据库快照
ri-tag	预留数据库实例
og-tag	数据库选项组
pg-tag	数据库参数组
subgrp-tag	数据库子网组
es-tag	事件订阅
cluster-tag	数据库集群
cluster-pg-tag	数据库集群参数组
cluster-snapshot-tag	数据库集群快照

自定义标签条件的语法如下：

"Condition":{"StringEquals":{"rds:*rds-tag-identifier*/*tag-name*": ["*value*"]} }

例如，以下 Condition 元素适用于具有名为 environment 的标签且标签值为 production 的数据库实例。

"Condition":{"StringEquals":{"rds:db-tag/**environment**": ["**production**"]} }

有关创建标签的信息，请参阅为 Amazon Aurora 和 Amazon RDS 资源添加标签。

Important

如果您使用标签管理对 RDS 资源的访问，建议您保护对 RDS 资源的标签的访问。您可通过为 AddTagsToResource 和 RemoveTagsFromResource 操作创建策略来管理对标签的访问。例如，以下策略不允许用户为所有资源添加或删除标签。之后，您可创建策略来允许特定用户添加或删除标签。

```
{
   "Version":"2012-10-17",
```

```
   "Statement":[
      {
         "Sid":"DenyTagUpdates",
         "Effect":"Deny",
         "Action":[
            "rds:AddTagsToResource",
            "rds:RemoveTagsFromResource"
         ],
         "Resource":"*"
      }
   ]
}
```

有关 Aurora 操作的列表，请参阅服务授权参考中的 Amazon RDS 定义的操作。

示例策略：使用自定义标签

以下示例说明了如何在 Amazon Aurora IAM 权限策略中使用自定义标签。有关向 Amazon Aurora 资源添加标签的更多信息，请参阅Amazon RDS 中的 Amazon 资源名称（ARN）。

> **Note**
>
> 所有示例都使用 us-west-2 区域和虚构的账户 ID。

示例 1：授予权限以允许对在特定标签中包含两个不同值的资源执行操作

以下策略所授予的权限允许对 `stage` 标签设置为 `development` 或 `test` 的实例执行 `CreateDBSnapshot` API 操作。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"AllowAnySnapshotName",
         "Effect":"Allow",
         "Action":[
            "rds:CreateDBSnapshot"
         ],
         "Resource":"arn:aws:rds:*:123456789012:snapshot:*"
      },
```

```
      {
         "Sid":"AllowDevTestToCreateSnapshot",
         "Effect":"Allow",
         "Action":[
            "rds:CreateDBSnapshot"
         ],
         "Resource":"arn:aws:rds:*:123456789012:db:*",
         "Condition":{
            "StringEquals":{
                "rds:db-tag/stage":[
                  "development",
                  "test"
               ]
            }
         }
      }
   ]
}
```

以下策略所授予的权限允许对 stage 标签设置为 development 或 test 的实例执行 ModifyDBInstance API 操作。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"AllowChangingParameterOptionSecurityGroups",
         "Effect":"Allow",
         "Action":[
            "rds:ModifyDBInstance"
         ],
         "Resource":" [
            "arn:aws:rds:*:123456789012:pg:*",
            "arn:aws:rds:*:123456789012:secgrp:*",
            "arn:aws:rds:*:123456789012:og:*"
         ]
      },
      {
         "Sid":"AllowDevTestToModifyInstance",
         "Effect":"Allow",
         "Action":[
            "rds:ModifyDBInstance"
         ],
```

```
         "Resource":"arn:aws:rds:*:123456789012:db:*",
         "Condition":{
            "StringEquals":{
                "rds:db-tag/stage":[
                  "development",
                  "test"
               ]
            }
         }
      }
   ]
}
```

示例 2：明确拒绝权限，以禁止创建使用指定数据库参数组的数据库实例

以下策略通过显式拒绝权限，禁止创建在数据库参数组中包含特定标签值的数据库实例。如果您需要在创建数据库实例时始终使用特定客户创建的数据库参数组，则可以应用此策略。使用 Deny 的策略最常用于限制由更宽泛的策略所授予的访问权限。

显式拒绝权限会取代授予的任何其他权限。这可确保用户身份不会无意中获得您绝不希望授予的权限。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"DenyProductionCreate",
         "Effect":"Deny",
         "Action":"rds:CreateDBInstance",
         "Resource":"arn:aws:rds:*:123456789012:pg:*",
         "Condition":{
            "StringEquals":{
               "rds:pg-tag/usage":"prod"
            }
         }
      }
   ]
}
```

示例 3：授予权限以允许对实例名称以用户名为前缀的数据库实例执行操作

以下策略授予的权限允许对具有如下性质的数据库实例调用除 AddTagsToResource 和 RemoveTagsFromResource 以外的任何 API：该数据库实例的实例名称以用户名称作为前缀，并且具有值为 stage 的 devo 标签或没有名为 stage 的标签。

策略中的 Resource 行通过 Amazon Resource Name (ARN) 标识资源。有关对 Amazon Aurora 资源使用 ARN 的更多信息，请参阅Amazon RDS 中的 Amazon 资源名称（ARN）。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"AllowFullDevAccessNoTags",
         "Effect":"Allow",
         "NotAction":[
            "rds:AddTagsToResource",
            "rds:RemoveTagsFromResource"
         ],
         "Resource":"arn:aws:rds:*:123456789012:db:${aws:username}*",
         "Condition":{
            "StringEqualsIfExists":{
               "rds:db-tag/stage":"devo"
            }
         }
      }
   ]
}
```

在创建过程中授予为 Aurora 资源添加标签的权限

某些 RDS API 操作支持在创建资源时指定标签。您可以使用资源标签来实现基于属性的控制 (ABAC)。有关更多信息，请参阅什么是适用于 AWS 的 ABAC？和使用标签控制对 AWS 资源的访问。

为使用户能够在创建时为资源添加标签，他们必须有权使用创建该资源的操作（如 rds:CreateDBCluster）。如果在创建操作中指定了标签，则 RDS 会对 rds:AddTagsToResource 操作执行额外的授权，以验证用户是否具备创建标签的权限。因此，用户还必须具有使用 rds:AddTagsToResource 操作的显式权限。

在 rds:AddTagsToResource 操作的 IAM 策略定义中，可以使用 aws:RequestTag 条件键要求在请求中使用标签来为资源加标签。

例如，以下策略支持用户在创建数据库实例期间创建数据库实例和应用标签，但只能使用特定的标签键（environment 或 project）：

```
{
   "Version": "2012-10-17",
   "Statement": [
       {
           "Effect": "Allow",
           "Action": [
               "rds:CreateDBInstance"
           ],
           "Resource": "*"
       },
       {
           "Effect": "Allow",
           "Action": [
               "rds:AddTagsToResource"
           ],
           "Resource": "*",
           "Condition": {
               "StringEquals": {
                   "aws:RequestTag/environment": ["production", "development"],
                   "aws:RequestTag/project": ["dataanalytics", "webapp"]
               },
               "ForAllValues:StringEquals": {
                   "aws:TagKeys": ["environment", "project"]
               }
           }
       }
   ]
}
```

此策略拒绝以下任何创建数据库实例请求：包含除 environment 或 project 标签之外的其它标签，或者未指定这两个标签中的任何一个。此外，用户必须为标签指定与策略中支持的值相匹配的值。

以下策略支持用户创建数据库集群和在创建过程中应用除 environment=prod 标签以外的任何标签：

```
{
   "Version": "2012-10-17",
   "Statement": [
       {
           "Effect": "Allow",
```

```
            "Action": [
                "rds:CreateDBCluster"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "rds:AddTagsToResource"
            ],
            "Resource": "*",
            "Condition": {
                "StringNotEquals": {
                    "aws:RequestTag/environment": "prod"
                }
            }
        }
    ]
}
```

支持在创建时添加标签的 RDS API 操作

以下 RDS API 操作支持在创建资源时添加标签。对于这些操作，您可以在创建资源时指定标签：

- CreateBlueGreenDeployment
- CreateCustomDBEngineVersion
- CreateDBCluster
- CreateDBClusterEndpoint
- CreateDBClusterParameterGroup
- CreateDBClusterSnapshot
- CreateDBInstance
- CreateDBInstanceReadReplica
- CreateDBParameterGroup
- CreateDBProxy
- CreateDBProxyEndpoint
- CreateDBSecurityGroup
- CreateDBShardGroup
- CreateDBSnapshot

- CreateDBSubnetGroup
- CreateEventSubscription
- CreateGlobalCluster
- CreateIntegration
- CreateOptionGroup
- CreateTenantDatabase
- CopyDBClusterParameterGroup
- CopyDBClusterSnapshot
- CopyDBParameterGroup
- CopyDBSnapshot
- CopyOptionGroup
- RestoreDBClusterFromS3
- RestoreDBClusterFromSnapshot
- RestoreDBClusterToPointInTime
- RestoreDBInstanceFromDBSnapshot
- RestoreDBInstanceFromS3
- RestoreDBInstanceToPointInTime
- PurchaseReservedDBInstancesOffering

如果您使用 AWS CLI 或 API 创建带有标签的资源，则 Tags 参数用于在创建期间对资源应用标签。

对于这些 API 操作，如果添加标签失败，则不会创建资源，并且请求失败且显示错误。这可确保要么创建带有标签的资源，要么根本不创建资源，从而防止创建不带所需标签的资源。

适用于 Amazon RDS 的 AWS 托管式策略

要向权限集和角色添加权限，与自己编写策略相比，使用 AWS 托管式策略更简单。创建仅为团队提供所需权限的 IAM 客户管理型策略需要时间和专业知识。要快速入门，您可以使用我们的 AWS 托管策略。这些策略涵盖常见使用案例，可在您的 AWS 账户 中使用。有关 AWS 托管策略的更多信息，请参阅《IAM 用户指南》中的AWS 托管策略。

AWS 服务 负责维护和更新 AWS 托管策略。您无法更改 AWS 托管策略中的权限。服务偶尔会向 AWS 管理型策略添加额外权限以支持新特征。此类型的更新会影响附加了策略的所有身份（权限集和角色）。当启动新特征或新操作可用时，服务最有可能会更新 AWS 管理型策略。服务不会从 AWS 托管式策略中删除权限，因此策略更新不会破坏您的现有权限。

此外，AWS还支持跨多种服务的工作职能的托管式策略。例如，`ReadOnlyAccess` AWS 托管式策略提供对许多 AWS 服务和资源的只读访问权限。当服务启动新特征时，AWS 会为新操作和资源添加只读权限。有关工作职能策略的列表和说明，请参阅《IAM 用户指南》中的适用于工作职能的 AWS 托管式策略。

主题

- AWS 托管式策略：AmazonRDSReadOnlyAccess
- AWS 托管式策略：AmazonRDSFullAccess
- AWS 托管式策略：AmazonRDSDataFullAccess
- AWS 托管式策略：AmazonRDSEnhancedMonitoringRole
- AWS 托管式策略：AmazonRDSPerformanceInsightsReadOnly
- AWS 托管式策略：AmazonRDSPerformanceInsightsFullAccess
- AWS 托管式策略：AmazonRDSDirectoryServiceAccess
- AWS 托管式策略：AmazonRDSServiceRolePolicy
- AWS 托管式策略：AmazonRDSPreviewServiceRolePolicy
- AWS 托管式策略：AmazonRDSBetaServiceRolePolicy

AWS 托管式策略：AmazonRDSReadOnlyAccess

此策略允许通过 AWS Management Console 对 Amazon RDS 进行只读访问。

权限详细信息

该策略包含以下权限：

- `rds` – 允许主体描述 Amazon RDS 资源并列出 Amazon RDS 资源的标签。
- `cloudwatch` – 允许主体获取 Amazon CloudWatch 指标统计数据。
- `ec2` – 可让主体描述可用区和网络资源。
- `logs` – 允许主体描述日志组的 CloudWatch Logs 日志流，并获取 CloudWatch Logs 日志事件。
- `devops-guru` – 允许主体描述具有 Amazon DevOps Guru 覆盖范围的资源，该覆盖范围由 CloudFormation 堆栈名称或资源标签指定。

有关此策略的更多信息，包括 JSON 策略文档，请参阅《AWS 托管式策略参考指南》中的 AmazonRDSReadOnlyAccess。

AWS 托管式策略：AmazonRDSFullAccess

此策略通过 AWS Management Console 提供了对 Amazon RDS 的完全访问权限。

权限详细信息

该策略包含以下权限：

- `rds` – 允许主体完全访问 Amazon RDS。
- `application-autoscaling` – 允许主体描述和管理 Application Auto Scaling 扩展目标和策略。
- `cloudwatch` – 允许主体获取 CloudWatch 指标统计数据并管理 CloudWatch 警报。
- `ec2` – 可让主体描述可用区和网络资源。
- `logs` – 允许主体描述日志组的 CloudWatch Logs 日志流，并获取 CloudWatch Logs 日志事件。
- `outposts` – 允许主体获取 AWS Outposts 实例类型。
- `pi` – 允许主体获取 Performance Insights 指标。
- `sns` – 允许主体访问 Amazon Simple Notification Service (Amazon SNS) 订阅和主题，并发布 Amazon SNS 消息。
- `devops-guru` – 允许主体描述具有 Amazon DevOps Guru 覆盖范围的资源，该覆盖范围由 CloudFormation 堆栈名称或资源标签指定。

有关此策略的更多信息，包括 JSON 策略文档，请参阅《AWS 托管式策略参考指南》中的 AmazonRDSFullAccess。

AWS 托管式策略：AmazonRDSDataFullAccess

此策略提供完全访问权限，允许在特定 AWS 账户中的 Aurora Serverless 集群上使用 Data API 和查询编辑器。此策略允许 AWS 账户从 AWS Secrets Manager 获取密钥的值。

您可以将 `AmazonRDSDataFullAccess` 策略附加到 IAM 身份。

权限详细信息

该策略包含以下权限：

- `dbqms` – 允许主体访问、创建、删除、描述和更新查询。Database Query Metadata Service (`dbqms`) 是一项仅限内部使用的服务。它为 AWS Management Console 上多项 AWS 服务（包括 Amazon RDS）提供查询编辑器最近的和保存的查询。
- `rds-data` – 允许主体在 Aurora Serverless 数据库上运行 SQL 语句。
- `secretsmanager` – 允许主体从 AWS Secrets Manager 获取密钥的值。

有关此策略的更多信息，包括 JSON 策略文档，请参阅《AWS 托管式策略参考指南》中的 AmazonRDSDataFullAccess。

AWS 托管式策略：AmazonRDSEnhancedMonitoringRole

此策略提供了对 Amazon CloudWatch Logs 的访问权限，支持 Amazon RDS 增强监控。

权限详细信息

该策略包含以下权限：

- `logs` – 允许主体创建 CloudWatch Logs 日志组和保留策略，并创建和描述日志组的 CloudWatch Logs 日志流。它还允许主体放置和获取 CloudWatch Logs 日志事件。

有关此策略的更多信息，包括 JSON 策略文档，请参阅《AWS 托管式策略参考指南》中的 AmazonRDSEnhancedMonitoringRole。

AWS 托管式策略：AmazonRDSPerformanceInsightsReadOnly

此策略提供了对 Amazon RDS Performance Insights 的只读访问权限，用于处理 Amazon RDS 数据库实例和 Amazon Aurora 数据库集群。

此策略现在包含 `Sid`（语句 ID）作为策略语句的标识符。

权限详细信息

该策略包含以下权限：

- `rds` – 允许主体描述 Amazon RDS 数据库实例和 Amazon Aurora 数据库集群。
- `pi` – 允许主体调用 Amazon RDS Performance Insights API 并访问 Performance Insights 指标。

有关此策略的更多信息，包括 JSON 策略文档，请参阅《AWS 托管式策略参考指南》中的 AmazonRDSPerformanceInsightsReadOnly。

AWS 托管式策略：AmazonRDSPerformanceInsightsFullAccess

此策略提供了对 Amazon RDS 性能详情的完全访问权限，用于处理 Amazon RDS 数据库实例和 Amazon Aurora 数据库集群。

此策略现在包含 `Sid`（语句 ID）作为策略语句的标识符。

权限详细信息

该策略包含以下权限：

- `rds` – 允许主体描述 Amazon RDS 数据库实例和 Amazon Aurora 数据库集群。
- `pi` – 允许主体调用 Amazon RDS 性能详情 API，以及创建、查看和删除性能分析报告。
- `cloudwatch` – 允许主体列出所有 Amazon CloudWatch 指标，并获取指标数据和统计数据。

有关此策略的更多信息，包括 JSON 策略文档，请参阅《AWS 托管式策略参考指南》中的 AmazonRDSPerformanceInsightsFullAccess。

AWS 托管式策略：AmazonRDSDirectoryServiceAccess

此策略允许 Amazon RDS 调用 AWS Directory Service。

权限详细信息

此策略包含以下权限：

- `ds` – 允许主体描述 AWS Directory Service 目录并控制对 AWS Directory Service 目录的授权。

有关此策略的更多信息，包括 JSON 策略文档，请参阅《AWS 托管式策略参考指南》中的 AmazonRDSDirectoryServiceAccess。

AWS 托管式策略：AmazonRDSServiceRolePolicy

您不能将 AmazonRDSServiceRolePolicy 策略附加到您的 IAM 实体。此附加到服务相关角色的策略允许 Amazon RDS 代表您执行操作。有关更多信息，请参阅 Amazon Aurora 的服务相关角色权限。

AWS 托管式策略：AmazonRDSPreviewServiceRolePolicy

您不应将 AmazonRDSPreviewServiceRolePolicy 附加到自己的 IAM 实体。此策略附加到一个服务相关角色，该角色支持 Amazon RDS 代表 RDS 数据库资源调用 AWS 服务。有关更多信息，请参阅 Amazon RDS 预览版的服务相关角色。

权限详细信息

该策略包含以下权限：

- ec2 – 可让主体描述可用区和网络资源。
- secretsmanager – 允许主体从 AWS Secrets Manager 获取密钥的值。
- cloudwatch、logs - 可让 Amazon RDS 通过 CloudWatch 代理将数据库实例指标和日志上传到 CloudWatch。

有关此策略的更多信息，包括 JSON 策略文档，请参阅《AWS 托管式策略参考指南》中的 AmazonRDSDataFullAccess。

AWS 托管式策略：AmazonRDSBetaServiceRolePolicy

您不应将 AmazonRDSBetaServiceRolePolicy 附加到自己的 IAM 实体。此策略附加到一个服务相关角色，该角色支持 Amazon RDS 代表 RDS 数据库资源调用 AWS 服务。有关更多信息，请参阅 Amazon RDS 测试版的服务相关角色权限。

权限详细信息

该策略包含以下权限：

- ec2 - 可让 Amazon RDS 在提供时间点还原功能的数据库实例上执行备份操作。
- secretsmanager - 可让 Amazon RDS 管理由 Amazon RDS 创建的数据库实例特定密钥。
- cloudwatch、logs - 可让 Amazon RDS 通过 CloudWatch 代理将数据库实例指标和日志上传到 CloudWatch。

有关此策略的更多信息，包括 JSON 策略文档，请参阅《AWS 托管式策略参考指南》中的 AmazonRDSBetaServiceRolePolicy。

AWS 托管式策略的 Amazon RDS 更新

查看有关 Amazon RDS 的 AWS 托管式策略的更新的详细信息（自从此服务开始跟踪这些更改）。有关此页面更改的自动提示，请订阅 Amazon RDS 文档历史记录页面上的 RSS 源。

更改	描述	日期
AWS 托管式策略：AmazonRDSPreviewServiceRolePolicy – 对现有策略的更新	Amazon RDS 从 `AWSServiceRoleForRDSPreview` 服务相关角色的 `AmazonRDSPreviewServiceRolePolicy` 中删除了 `sns:Publish` 权限。有关更多信息，请参阅 AWS 托管式策略：AmazonRDSPreviewServiceRolePolicy。	2024 年 8 月 7 日
AWS 托管式策略：AmazonRDSBetaServiceRolePolicy - 对现有策略的更新	Amazon RDS 从 `AWSServiceRoleForRDSBeta` 服务相关角色的 `AmazonRDSBetaServiceRolePolicy` 中删除了 `sns:Publish` 权限。有关更多信息，请参阅 AWS 托管式策略：AmazonRDSBetaServiceRolePolicy。	2024 年 8 月 7 日
AWS 托管式策略：AmazonRDSServiceRolePolicy - 对现有策略的更新	Amazon RDS 从 `AWSServiceRoleForRDS` 服务相关角色的 `AmazonRDSServiceRolePolicy` 中删除了 `sns:Publish` 权限。有关更多信息，请参阅 AWS 托管式策略：AmazonRDSServiceRolePolicy。	2024 年 7 月 2 日

更改	描述	日期
适用于 Amazon RDS 的 AWS 托管式策略 - 对现有策略的更新	Amazon RDS 向 `AWSServiceRoleForRDSCustom` 服务相关角色的 `AmazonRDSCustomServiceRolePolicy` 添加了一项新权限，允许 RDS Custom for SQL Server 修改底层数据库主机实例类型。RDS 还增加了用于获取数据库主机的实例类型信息的 `ec2:DescribeInstanceTypes` 权限。有关更多信息，请参阅 适用于 Amazon RDS 的 AWS 托管式策略。	2024 年 4 月 8 日
适用于 Amazon RDS 的 AWS 托管式策略：新策略	Amazon RDS 添加了一个名为 `AmazonRDSCustomInstanceProfileRolePolicy` 的新托管式策略，允许 RDS Custom 通过 EC2 实例配置文件执行自动化操作和数据库管理任务。有关更多信息，请参阅 适用于 Amazon RDS 的 AWS 托管式策略。	2024 年 2 月 27 日
Amazon Aurora 的服务相关角色权限 – 更新到现有策略	Amazon RDS 为 `AWSServiceRoleForRDS` 服务相关角色的 `AmazonRDSServiceRolePolicy` 添加了新的语句 ID。 有关更多信息，请参阅 Amazon Aurora 的服务相关角色权限。	2024 年 1 月 19 日

更改	描述	日期
适用于 Amazon RDS 的 AWS 托管式策略 – 对现有策略的更新	AmazonRDSPerformanceInsightsReadOnly 和 AmazonRDSPerformanceInsightsFullAccess 托管式策略现在将 Sid（语句 ID）作为标识符包括在策略语句中。 有关更多信息，请参阅AWS 托管式策略：AmazonRDSPerformanceInsightsReadOnly和AWS 托管式策略：AmazonRDSPerformanceInsightsFullAccess	2023 年 10 月 23 日
适用于 Amazon RDS 的 AWS 托管式策略 – 对现有策略的更新	Amazon RDS 将新权限添加到 AmazonRDSFullAccess 托管式策略。这些权限允许您生成、查看和删除一段时间内的性能分析报告。 有关为性能详情配置访问策略的更多信息，请参阅为 Performance Insights 配置访问策略。	2023 年 8 月 17 日

更改	描述	日期
适用于 Amazon RDS 的 AWS 托管式策略 – 新策略和对现有策略的更新	Amazon RDS 向 `AmazonRDSPerformanceInsightsReadOnly` 托管式策略和名为 `AmazonRDSPerformanceInsightsFullAccess` 的新托管式策略添加了新权限。这些权限允许您分析一段时间内的性能详情、查看分析结果和建议以及删除报告。 有关为性能详情配置访问策略的更多信息，请参阅为 Performance Insights 配置访问策略。	2023 年 8 月 16 日
适用于 Amazon RDS 的 AWS 托管式策略 – 更新到现有策略	Amazon RDS 向 `AmazonRDSFullAccess` 和 `AmazonRDSReadOnlyAccess` 添加了新的 Amazon CloudWatch 命名空间 `ListMetrics` 。 Amazon RDS 需要此命名空间来列出特定的资源使用情况指标。 有关更多信息，请参阅《Amazon CloudWatch 用户指南》中的管理对您的 CloudWatch 资源的访问权限的概述。	2023 年 4 月 4 日

更改	描述	日期
Amazon Aurora 的服务相关角色权限 – 更新到现有策略	Amazon RDS 为 AWSServiceRoleForRDS 服务相关角色的 AmazonRDSServiceRolePolicy 添加了新权限，以便与 AWS Secrets Manager 集成。RDS 需要与 Secrets Manager 集成才能在 Secrets Manager 中管理主用户密码。该密钥使用保留的命名惯例并限制客户更新。 有关更多信息，请参阅 使用 Amazon Aurora 和 AWS Secrets Manager 管理密码。	2022 年 12 月 22 日
适用于 Amazon RDS 的 AWS 托管式策略 – 对现有策略的更新	Amazon RDS 向 AmazonRDSFullAccess 和 AmazonRDSReadOnlyAccess 托管式策略添加了新权限，以允许您在 RDS 控制台中开启 Amazon DevOps Guru。需要此权限才能检查 DevOps Guru 是否已开启。 有关更多信息，请参阅 配置适用于 RDS 的 DevOps Guru 的 IAM 访问策略。	2022 年 12 月 19 日

更改	描述	日期
Amazon Aurora 的服务相关角色权限 – 更新到现有策略	Amazon RDS 向 `PutMetricData` 的 `AmazonRDSPreviewServiceRolePolicy` 添加了新的 Amazon CloudWatch 命名空间。 Amazon RDS 需要此命名空间来发布资源用量指标。 有关更多信息，请参阅《Amazon CloudWatch 用户指南》中的使用条件键限制对 CloudWatch 命名空间的访问。	2022 年 6 月 7 日
Amazon Aurora 的服务相关角色权限 – 更新到现有策略	Amazon RDS 向 `PutMetricData` 的 `AmazonRDSBetaServiceRolePolicy` 添加了新的 Amazon CloudWatch 命名空间。 Amazon RDS 需要此命名空间来发布资源用量指标。 有关更多信息，请参阅《Amazon CloudWatch 用户指南》中的使用条件键限制对 CloudWatch 命名空间的访问。	2022 年 6 月 7 日

更改	描述	日期
Amazon Aurora 的服务相关角色权限 – 更新到现有策略	Amazon RDS 向 PutMetricData 的 AWSServiceRoleForRDS 添加了新的 Amazon CloudWatch 命名空间。 Amazon RDS 需要此命名空间来发布资源用量指标。 有关更多信息，请参阅《Amazon CloudWatch 用户指南》中的使用条件键限制对 CloudWatch 命名空间的访问。	2022 年 4 月 22 日
适用于 Amazon RDS 的 AWS 托管式策略：新策略	Amazon RDS 添加了名为 AmazonRDSPerformanceInsightsReadOnly 的新托管式策略，以允许 Amazon RDS 代表数据库实例调用 AWS 服务。 有关为性能详情配置访问策略的更多信息，请参阅为 Performance Insights 配置访问策略。	2022 年 3 月 10 日

更改	描述	日期
Amazon Aurora 的服务相关角色权限 – 更新到现有策略	Amazon RDS 向 `PutMetricData` 的 `AWSServiceRoleForRDS` 添加了新的 Amazon CloudWatch 命名空间。 要发布 CloudWatch 指标，Amazon DocumentDB（MongoDB 兼容）和 Amazon Neptune 需要这些命名空间。 有关更多信息，请参阅《Amazon CloudWatch 用户指南》中的使用条件键限制对 CloudWatch 命名空间的访问。	2022 年 3 月 4 日
Amazon RDS 开始跟踪更改	Amazon RDS 为其 AWS 托管式策略开启了跟踪更改。	2021 年 10 月 26 日

防范跨服务混淆代理问题

混淆代理问题是一个安全问题，即没有执行操作权限的实体可能会迫使更具权限的实体执行该操作。在 AWS 中，跨服务模拟可能会导致混淆代理问题。

一个服务（呼叫服务）调用另一项服务（所谓的服务）时，可能会发生跨服务模拟。可以操纵调用服务来使用其权限，以不应该具有的访问权限对另一个客户的资源进行操作。为防止这种情况，AWS 提供可帮助您保护所有服务数据的工具，而这些服务中的服务主体有权限访问账户中的资源。有关更多信息，请参阅《IAM 用户指南》中的混淆代理人问题。

要限制 Amazon RDS 为其他服务提供对资源的权限，我们建议在资源策略中使用 `aws:SourceArn` 和 `aws:SourceAccount` 全局条件上下文键。

在某些情况下，`aws:SourceArn` 值不包含账户 ID，例如将 Amazon Resource Name (ARN) 用于 Amazon S3 存储桶时。在这些情况下，请确保使用两个全局条件上下文键来限制权限。在某些情况下，您可以使用两个全局条件上下文键并让 `aws:SourceArn` 值包含账户 ID。在这些情况下，当 `aws:SourceAccount` 值和 `aws:SourceArn` 值中的账户使用相同策略语句时，确保二者使用相同的账户 ID。如果您只希望将一个资源与跨服务访问相关联，请使用 `aws:SourceArn`。如果您想允许指定 AWS 账户中的任何资源与跨服务使用相关联，请使用 `aws:SourceAccount`。

确保 `aws:SourceArn` 的值是 Amazon RDS 资源类型的 ARN。有关更多信息，请参阅 Amazon RDS 中的 Amazon 资源名称（ARN）。

防范混淆代理问题最有效的方法是使用 `aws:SourceArn` 全局条件上下文键和资源的完整 ARN。在某些情况下，您可能不知道资源的完整 ARN，或者您可能正在指定多个资源。在这些情况下，请将带通配符 (*) 的 `aws:SourceArn` 全局上下文条件键用于 ARN 的未知部分。例如，`arn:aws:rds:*:123456789012:*`。

以下示例演示如何使用 Amazon RDS 中的 `aws:SourceArn` 和 `aws:SourceAccount` 全局条件上下文键来防范混淆代理问题。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Sid": "ConfusedDeputyPreventionExamplePolicy",
    "Effect": "Allow",
    "Principal": {
      "Service": "rds.amazonaws.com"
```

```
    },
    "Action": "sts:AssumeRole",
    "Condition": {
      "ArnLike": {
        "aws:SourceArn": "arn:aws:rds:us-east-1:123456789012:db:mydbinstance"
      },
      "StringEquals": {
        "aws:SourceAccount": "123456789012"
      }
    }
  }
}
```

有关使用 aws:SourceArn 和 aws:SourceAccount 全局条件上下文键的更多策略示例，请参阅以下部分：

- 授予向 Amazon SNS 主题发布通知的权限。
- 设置 Amazon S3 存储桶的访问权限（PostgreSQL 导入）
- 设置 Amazon S3 存储桶的访问权限（PostgreSQL 导出）

IAM 数据库身份验证

您可以使用 AWS Identity and Access Management（IAM）数据库身份验证对数据库集群进行身份验证。IAM 数据库身份验证适用于 Aurora MySQL 和 Aurora PostgreSQL。利用此身份验证方法，您在连接到数据库集群时将无需使用密码。而是使用身份验证令牌。

身份验证令牌 是 Amazon Aurora 根据请求生成的唯一字符串。身份验证令牌是使用 AWS 签名版本 4 生成的。每个令牌的使用期限为 15 分钟。您无需将用户凭证存储在数据库中，因为身份验证是使用 IAM 进行外部管理的。此外，您仍可使用标准数据库身份验证。令牌仅用于身份验证，建立后不会影响会话。

IAM 数据库身份验证具有以下优势：

- 数据库的出站和进站网络流量是使用安全套接字层 (SSL) 或传输层安全性 (TLS) 加密的。有关将 SSL/TLS 与 Amazon Aurora 一起使用的更多信息，请参阅使用 SSL/TLS 加密与数据库集群的连接。
- 您可以使用 IAM 集中管理对数据库资源的访问，而不是单独管理对每个数据库集群的访问。
- 对于在 Amazon EC2 上运行的应用程序，您可以使用 EC2 实例特定的配置文件凭证访问数据库以提高安全性，而不是使用密码。

一般来说，如果应用程序每秒创建的连接少于 200 个，而且您不想直接在应用程序代码中管理用户名和密码，请考虑使用 IAM 数据库身份验证。

Amazon Web Services（AWS）JDBC 驱动程序支持 IAM 数据库身份验证。有关更多信息，请参阅 Amazon Web Services（AWS）JDBC 驱动程序 GitHub 存储库中的 AWS IAM 身份验证插件。

Amazon Web Services（AWS）Python 驱动程序支持 IAM 数据库身份验证。有关更多信息，请参阅 Amazon Web Services（AWS）Python 驱动程序 GitHub 存储库中的 AWS IAM 身份验证插件。

浏览以下主题，学习设置 IAM 用于数据库身份验证的过程：

- 启用和禁用 IAM 数据库身份验证
- 创建和使用适用于 IAM 数据库访问的 IAM 策略
- 使用 IAM 身份验证创建数据库账户
- 使用 IAM 身份验证连接到数据库集群

区域和版本可用性

功能可用性和支持因每个 Aurora 数据库引擎的特定版本以及 AWS 区域 而异。有关适用于 Aurora 和 IAM 数据库身份验证的版本和区域可用性的更多信息，请参阅 支持 IAM 数据库身份验证的区域和 Aurora 数据库引擎。

对于 Aurora MySQL，除 db.t2.small 和 db.t3.small 外，所有受支持的数据库实例类都支持 IAM 数据库身份验证。有关受支持的数据库实例类的信息，请参阅 数据库实例类支持的数据库引擎。

CLI 和开发工具包支持

IAM 数据库身份验证可用于 AWS CLI 以及以下特定于语言的 AWS 软件开发工具包：

- 适用于 .NET 的 AWS SDK
- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java 的 AWS SDK
- 适用于 JavaScript 的 AWS SDK
- 适用于 PHP 的 AWS SDK
- 适用于 Python (Boto3) 的 AWS SDK
- 适用于 Ruby 的 AWS SDK

IAM 数据库身份验证的限制

使用 IAM 数据库身份验证时，以下限制适用：

- IAM 数据库身份验证会限制连接数，即每秒 200 个连接。

  使用同一身份验证令牌的连接不受限制。建议您尽可能重复使用身份验证令牌。
- 目前，IAM 数据库身份验证并不支持所有的全局条件上下文键。

  有关全局条件上下文键的更多信息，请参阅《IAM 用户指南》中的 AWS 全局条件上下文键。
- 对于 PostgreSQL，如果将 IAM 角色（`rds_iam`）添加到某个用户（包括 RDS 主用户），IAM 身份验证将优先于密码身份验证，因此该用户必须以 IAM 用户身份登录。
- 对于 Aurora PostgreSQL，您不能使用 IAM 身份验证来建立复制连接。
- 不能使用自定义 Route 53 DNS 记录代替数据库集群端点来生成身份验证令牌。

- CloudWatch 和 CloudTrail 不记录 IAM 身份验证。这些服务不会跟踪授权 IAM 角色启用数据库连接的 `generate-db-auth-token` API 调用。
- IAM 数据库身份验证需要数据库集群上的计算资源。您的数据库上必须有 300 到 1000 MiB 的额外内存，才能实现可靠的连接。要查看工作负载所需的内存，请在启用 IAM 数据库身份验证之前和之后，比较增强型监控进程列表中 RDS 进程的 RES 列。请参阅在 RDS 控制台中查看操作系统指标。

  如果您使用的是可突增类实例，请通过将其它参数使用的内存（如缓冲区和缓存）减少相同的量，从而避免内存耗尽。
- 对于 Aurora MySQL，您不能对使用 IAM 身份验证配置的数据库用户使用基于密码的身份验证。
- 对于任何引擎，RDS on Outposts 不支持 IAM 数据库身份验证。

IAM 数据库身份验证建议

在使用 IAM 数据库身份验证时，建议使用以下方法：

- 当您的应用程序每秒需要少于 200 个新的 IAM 数据库身份验证连接时，请使用 IAM 数据库身份验证。

  使用 Amazon Aurora 的数据库引擎不会对每秒的身份验证尝试次数施加任何限制。不过，在使用 IAM 数据库身份验证时，您的应用程序必须生成身份验证令牌。之后，您的应用程序将使用该令牌连接到数据库集群。如果超出每秒的最大新连接数限制，则 IAM 数据库身份验证的额外开销可能会导致连接受到限制。

  考虑在应用程序中使用连接池来减少持续的连接创建。这可以减少 IAM 数据库身份验证的开销，并允许应用程序重用现有连接。或者，考虑对这些使用案例使用 RDS 代理。RDS 代理有额外费用。请参阅 RDS 代理定价。
- IAM 数据库身份验证令牌的大小取决于许多因素，包括 IAM 标签的数量、IAM 服务策略、ARN 长度以及其他 IAM 和数据库属性。此令牌的最小大小通常约为 1KB，但可以更大。由于使用 IAM 身份验证将此令牌用作数据库的连接字符串中的密码，因此，您应确保您的数据库驱动程序（例如 ODBC）和/或任何工具不会因其大小而限制或以其他方式截断该令牌。截断的令牌将导致数据库和 IAM 执行的身份验证失败。
- 如果您在创建 IAM 数据库身份验证令牌时使用临时凭证，则在使用 IAM 数据库身份验证令牌发出连接请求时，临时凭证必须仍然有效。

不支持的 AWS 全局条件上下文键

IAM 数据库身份验证不支持 AWS 全局条件上下文键的以下子集。

- aws:Referer
- aws:SourceIp
- aws:SourceVpc
- aws:SourceVpce
- aws:UserAgent
- aws:VpcSourceIp

有关更多信息，请参阅《IAM 用户指南》中的 AWS 全局条件上下文键。

启用和禁用 IAM 数据库身份验证

默认情况下，已对数据库集群禁用 IAM 数据库身份验证。您可以使用 AWS Management Console、AWS CLI 或 API 启用或禁用 IAM 数据库身份验证。

您可以在执行以下操作之一时启用 IAM 数据库身份验证：

- 要创建启用 IAM 数据库身份验证的新数据库集群，请参阅 创建 Amazon Aurora 数据库集群。
- 要修改数据库集群以启用 IAM 数据库身份验证，请参阅 修改 Amazon Aurora 数据库集群。
- 要从启用了 IAM 数据库身份验证的快照还原数据库集群，请参阅从数据库集群快照还原。
- 要将数据库集群还原到已启用 IAM 数据库身份验证的某个时间点，请参阅 将数据库集群还原到指定时间。

控制台

每个创建或修改工作流程都有一个数据库身份验证部分，您可以在其中启用或禁用 IAM 数据库身份验证。在该部分中，选择密码和 IAM 数据库身份验证以启用 IAM 数据库身份验证。

为现有数据库集群启用或禁用 IAM 数据库身份验证

1. 通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择 Databases (数据库)。
3. 选择要修改的数据库集群。

> **Note**
>
> 只有当数据库集群中的所有数据库实例都与 IAM 兼容时，才能启用 IAM 身份验证。在区域和版本可用性中检查兼容性要求。

4. 选择修改。
5. 在 Database authentication (数据库身份验证) 部分中，选择 Password and IAM database authentication (密码和 IAM 数据库身份验证)，启用 IAM 数据库身份验证。选择密码身份验证或密码和 Kerberos 身份验证以禁用 IAM 身份验证。
6. 也可以选择启用向 CloudWatch Logs 发布 IAM 数据库身份验证日志。在日志导出下，选择 iam-db-auth-error log 选项。将您的日志发布到 CloudWatch Logs 会消耗存储空间，并且您需要为该存储空间支付费用。请务必删除所有不再需要的 CloudWatch Logs。
7. 选择 Continue (继续)。
8. 要立即应用更改，请在修改计划部分中选择立即。
9. 选择 修改集群。

AWS CLI

要使用 AWS CLI 创建采用 IAM 身份验证的新数据库集群，请使用 `create-db-cluster` 命令。指定 `--enable-iam-database-authentication` 选项。

要更新现有的数据库集群以采用或不采用 IAM 身份验证，请使用 AWS CLI 命令 `modify-db-cluster`。根据需要指定 `--enable-iam-database-authentication` 或 `--no-enable-iam-database-authentication` 选项。

> **Note**
>
> 只有当数据库集群中的所有数据库实例都与 IAM 兼容时，才能启用 IAM 身份验证。在区域和版本可用性中检查兼容性要求。

默认情况下，Aurora 在下一个维护时段执行修改。如果您要覆盖该选项并尽快启用 IAM 数据库身份验证，请使用 `--apply-immediately` 参数。

如果您将还原数据库集群，请使用下列 AWS CLI 命令之一：

- `restore-db-cluster-to-point-in-time`

- restore-db-cluster-from-db-snapshot

IAM 数据库身份验证设置默认为源快照的设置。要更改此设置，请根据需要设置 --enable-iam-database-authentication 或 --no-enable-iam-database-authentication 选项。

RDS API

要使用 API 创建采用 IAM 身份验证的新数据库实例，请使用 API 操作 CreateDBCluster。将 EnableIAMDatabaseAuthentication 参数设置为 true。

要更新现有的数据库集群以采用 IAM 身份验证，请使用 API 操作 ModifyDBCluster。将 EnableIAMDatabaseAuthentication 参数设置为 true 可启用 IAM 身份验证，将其设置为 false 可禁用 IAM 身份验证。

> **Note**
>
> 只有当数据库集群中的所有数据库实例都与 IAM 兼容时，才能启用 IAM 身份验证。在区域和版本可用性中检查兼容性要求。

如果您将还原数据库集群，请使用以下 API 操作之一：

- RestoreDBClusterFromSnapshot
- RestoreDBClusterToPointInTime

IAM 数据库身份验证设置默认为源快照的设置。要更改此设置，请将 EnableIAMDatabaseAuthentication 参数设置为 true 以启用 IAM 身份验证，或将其设置为 false 以禁用 IAM 身份验证。

创建和使用适用于 IAM 数据库访问的 IAM 策略

要允许用户或角色连接到数据库集群，您必须创建 IAM policy。之后，将策略附加到权限集或角色。

> **Note**
>
> 要了解有关 IAM 策略的更多信息，请参阅 Amazon Aurora 的 Identity and Access Management。

以下示例策略允许用户使用 IAM 数据库身份验证连接到数据库集群。

JSON

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
             "rds-db:connect"
         ],
         "Resource": [
             "arn:aws:rds-db:us-east-2:1234567890:dbuser:db-ABCDEFGHIJKL01234/
db_user"
         ]
      }
   ]
}
```

JSON

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
             "rds-db:connect"
         ],
         "Resource": [
             "arn:aws:rds-db:us-east-2:1234567890:dbuser:cluster-
ABCDEFGHIJKL01234/db_user"
         ]
      }
   ]
}
```

> **Important**
>
> 具有管理员权限的用户即使在 IAM policy 中没有显式权限，也可以访问数据库集群。如果您希望限制管理员访问数据库集群，您可以创建具有合适的较低权限的 IAM 角色，并将其分配给管理员。

> **Note**
>
> 请不要将 `rds-db:` 前缀与以 `rds:` 开头的其他 RDS API 操作前缀混淆。仅在 IAM 数据库身份验证使用 `rds-db:` 前缀和 `rds-db:connect` 操作。它们在任何其他上下文中无效。

示例策略包含带以下元素的单个语句：

- `Effect` – 指定 `Allow` 以授予数据库集群的访问权限。如果您没有显式允许访问，则默认情况下将拒绝访问。
- `Action` – 指定 `rds-db:connect` 以允许连接到数据库集群。
- `Resource` – 指定 Amazon Resource Name (ARN) 以描述一个数据库集群中的一个数据库账户。ARN 格式如下所示。

```
arn:aws:rds-db:region:account-id:dbuser:DbClusterResourceId/db-user-name
```

在此格式中，替换以下内容：

- *`region`* 是数据库集群所在的 AWS 区域。在示例策略中，AWS 区域为 `us-east-2`。
- *`account-id`* 是数据库集群的 AWS 账号。在示例策略中，账号为 `1234567890`。该用户必须位于与数据库集群的账户相同的账户中。

  要执行跨账户存取，请在数据库集群的账户中使用上面显示的策略创建 IAM 角色，并允许其他账户代入该角色。
- *`DbClusterResourceId`* 是数据库集群的标识符。此标识符对 AWS 区域是唯一的，并且绝不会更改。在示例策略中，标识符为 `cluster-ABCDEFGHIJKL01234`。

要在适用于 Amazon Aurora 的 AWS Management Console 中查找数据库集群资源 ID，请选择数据库集群来查看其详细信息。然后，选择配置选项卡。Resource ID (资源 ID) 将显示在 Configuration (配置) 部分中。

或者，您可以使用 AWS CLI 命令列出当前 AWS 区域中所有数据库集群的标识符和资源 ID，如下所示。

```
aws rds describe-db-clusters --query "DBClusters[*].
[DBClusterIdentifier,DbClusterResourceId]"
```

> **Note**
>
> 如果要通过 RDS 代理连接到数据库，请指定代理资源 ID，如 `prx-ABCDEFGHIJKL01234`。有关将 IAM 数据库身份验证与 RDS 代理结合使用的信息，请参阅 使用 IAM 身份验证连接到代理。

- *`db-user-name`* 是与 IAM 身份验证关联的数据库账户的名称。在示例策略中，数据库账户为 `db_user`。

您可以构造其他 ARN 以支持多种访问模式。以下策略允许访问一个数据库集群中的两个不同的数据库账户。

JSON

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
             "rds-db:connect"
         ],
         "Resource": [
             "arn:aws:rds-db:us-east-2:123456789012:dbuser:db-ABCDEFGHIJKL01234/
jane_doe",
```

```
            "arn:aws:rds-db:us-east-2:123456789012:dbuser:db-ABCDEFGHIJKL01234/
mary_roe"
         ]
      }
   ]
}
```

JSON

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
             "rds-db:connect"
         ],
         "Resource": [
             "arn:aws:rds-db:us-east-2:123456789012:dbuser:cluster-
ABCDEFGHIJKL01234/jane_doe",
             "arn:aws:rds-db:us-east-2:123456789012:dbuser:cluster-
ABCDEFGHIJKL01234/mary_roe"
         ]
      }
   ]
}
```

以下策略使用“*”字符匹配特定 AWS 账户和 AWS 区域的所有数据库集群和数据库账户。

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
```

```
                "rds-db:connect"
            ],
            "Resource": [
                "arn:aws:rds-db:us-east-2:1234567890:dbuser:*/*"
            ]
        }
    ]
}
```

以下策略匹配特定 AWS 账户和 AWS 区域的所有数据库集群。不过，该策略仅允许访问具有 jane_doe 数据库账户的数据库集群。

JSON

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
             "rds-db:connect"
         ],
         "Resource": [
             "arn:aws:rds-db:us-east-2:123456789012:dbuser:*/jane_doe"
         ]
      }
   ]
}
```

用户或角色只能访问数据库用户有权访问的那些数据库。例如，假设数据库集群具有一个名为 dev 的数据库，以及另一个名为 test 的数据库。如果数据库用户 jane_doe 只能访问 dev，则与 jane_doe 用户一起访问数据库集群的任何用户或角色也只能访问 dev。此访问限制还适用于其他数据库对象 (如表、视图等)。

管理员必须创建 IAM policy，以便为实体授予权限，从而对其所需的指定资源执行特定的 API 操作。然后，管理员必须将这些策略附加到需要这些权限的权限集或角色。有关策略示例，请参阅“Amazon Aurora 的基于身份的策略示例”。

将 IAM policy 附加到权限集或角色

在创建允许数据库身份验证的 IAM policy 之后，您需要将该策略附加到权限集或角色。有关此主题的教程，请参阅 IAM 用户指南 中的创建和附加您的第一个客户托管策略。

在演练此教程时，您可以使用此部分中显示的策略示例之一作为起点并根据您的需求进行定制。在教程结束时，您将有一个权限集，该权限集附加了一个可利用 `rds-db:connect` 操作的策略。

> **Note**
>
> 您可以将多个权限集或角色映射到同一数据库用户账户。例如，假设您的 IAM 策略指定了以下资源 ARN。
>
> ```
> arn:aws:rds-db:us-east-2:123456789012:dbuser:cluster-12ABC34DEFG5HIJ6KLMNOP78QR/jane_doe
> ```
>
> 如果将该策略附加到 Jane、Bob 和 Diego，则其中的每个用户都可以使用 `jane_doe` 数据库账户连接到指定的数据库集群。

使用 IAM 身份验证创建数据库账户

在使用 IAM 数据库身份验证时，您不需要为创建的用户账户分配数据库密码。如果删除映射到某个数据库账户的用户，则还应使用 `DROP USER` 语句删除该数据库账户。

> **Note**
>
> 用于 IAM 身份验证的用户名必须与数据库中用户名的大小写相匹配。

主题

- 对 Aurora MySQL 使用 IAM 身份验证
- 对 Aurora PostgreSQL 使用 IAM 身份验证

对 Aurora MySQL 使用 IAM 身份验证

对于 Aurora MySQL，身份验证由 AWSAuthenticationPlugin 处理，这是 AWS 提供的一个插件，可以与 IAM 无缝协作以验证您的用户的身份。以主用户或其他可以创建用户和授予权限的用户身份连接到数据库集群。连接后，发出 CREATE USER 语句，如以下示例中所示。

```
CREATE USER 'jane_doe' IDENTIFIED WITH AWSAuthenticationPlugin AS 'RDS';
```

IDENTIFIED WITH 子句允许 Aurora MySQL 使用 AWSAuthenticationPlugin 对数据库账户 (jane_doe) 进行身份验证。AS 'RDS' 子句是指身份验证方法。确保指定的数据库用户名与 IAM 数据库访问策略中的资源相同。有关更多信息，请参阅“创建和使用适用于 IAM 数据库访问的 IAM 策略”。

> **Note**
>
> 如果看到以下消息，则意味着 AWS 提供的插件对当前的数据库集群不可用。
> `ERROR 1524 (HY000): Plugin 'AWSAuthenticationPlugin' is not loaded`
> 要纠正该错误，请确认您使用支持的配置，并且在数据库集群上启用了 IAM 数据库身份验证。有关更多信息，请参阅“区域和版本可用性”和“启用和禁用 IAM 数据库身份验证”。

在使用 AWSAuthenticationPlugin 创建一个账户后，可以像管理其他数据库账户一样管理此账户。例如，您可以使用 GRANT 语句和 REVOKE 语句修改账户权限，或使用 ALTER USER 语句修改各种账户属性。

使用 IAM 时，数据库网络流量使用 SSL/TLS 进行加密。要允许 SSL 连接，请使用以下命令修改用户账户。

```
ALTER USER 'jane_doe'@'%' REQUIRE SSL;
```

对 Aurora PostgreSQL 使用 IAM 身份验证

要在 Aurora PostgreSQL 中使用 IAM 身份验证，请以主用户或其他可以创建用户和授予权限的用户身份连接到数据库集群。连接后，创建数据库用户，然后向其授予 rds_iam 角色，如以下示例中所示。

```
CREATE USER db_userx;
GRANT rds_iam TO db_userx;
```

确保指定的数据库用户名与 IAM 数据库访问策略中的资源相同。有关更多信息，请参阅 创建和使用适用于 IAM 数据库访问的 IAM 策略。您必须授予 `rds_iam` 角色才能使用 IAM 身份验证。也可以使用该角色的嵌套成员资格或间接授权。

请注意，PostgreSQL 数据库用户可以使用 IAM 或 Kerberos 身份验证（但不能同时使用），因此，此用户也不能具有 `rds_ad` 角色。这也适用于嵌套成员资格。有关更多信息，请参阅 步骤 7：为您的 Kerberos 主体创建 PostgreSQL 用户 。

使用 IAM 身份验证连接到数据库集群

对于 IAM 数据库身份验证，您可以在连接到数据库集群时使用身份验证令牌。身份验证令牌 是您使用的一个字符串而不是密码。在生成身份验证令牌后，将在 15 分钟后失效。如果您尝试使用过期的令牌进行连接，则连接请求将被拒绝。

必须使用 AWS 签名版本 4 为每个身份验证令牌附带一个有效签名。（有关更多信息，请参阅《AWS 一般参考》中的签名版本 4 签名流程。）AWS CLI 和 AWS 软件开发工具包（例如 适用于 Java 的 AWS SDK 或者 适用于 Python (Boto3) 的 AWS SDK）可以自动对您创建的每个令牌进行签名。

从其他AWS服务（如 AWS Lambda）连接到 Amazon Aurora 时，您可以使用身份验证令牌。通过使用令牌，可以避免在代码中放置密码。或者，您可以使用 AWS 开发工具包以编程方式创建身份验证令牌并为其签名。

在获得签名的 IAM 身份验证令牌后，您可以连接到 Aurora 数据库集群。在下文中，您可以了解如何使用命令行工具或 AWS 开发工具包（例如 适用于 Java 的 AWS SDK 或 适用于 Python (Boto3) 的 AWS SDK）执行该操作。

有关更多信息，请参阅以下博客文章：

- 使用 IAM 身份验证通过 SQL Workbench/J 连接到 Aurora MySQL 或 Amazon RDS for MySQL
- 使用 IAM 身份验证通过 pgAdmin Amazon Aurora PostgreSQL 或 Amazon RDS for PostgreSQL 进行连接

先决条件

以下是使用 IAM 身份验证连接到数据库集群的先决条件：

- 启用和禁用 IAM 数据库身份验证
- 创建和使用适用于 IAM 数据库访问的 IAM 策略
- 使用 IAM 身份验证创建数据库账户

主题

- 使用 IAM 身份验证和 AWS 驱动程序连接到数据库集群
- 通过命令行 AWS CLI 和 mysql 客户端，使用 IAM 身份验证连接到数据库集群
- 通过命令行使用 IAM 身份验证连接到数据库集群：AWS CLI 和 psql 客户端
- 使用 IAM 身份验证和 适用于 .NET 的 AWS SDK 连接到数据库集群
- 使用 IAM 身份验证和 适用于 Go 的 AWS SDK 连接到数据库集群
- 使用 IAM 身份验证和 适用于 Java 的 AWS SDK 连接到数据库集群
- 使用 IAM 身份验证和 适用于 Python (Boto3) 的 AWS SDK 连接到数据库集群

使用 IAM 身份验证和 AWS 驱动程序连接到数据库集群

借助 AWS 驱动程序套件，可显著缩短切换和故障转移时间，并支持使用 AWS Secrets Manager、AWS Identity and Access Management（IAM）和联合身份进行身份验证。AWS 驱动程序依靠监控数据库集群状态以及了解集群拓扑，来确定新的写入器。这种方法将切换和故障转移时间缩短到几秒钟，而开源驱动程序的切换和故障转移时间则为几十秒。

有关 AWS 驱动程序的更多信息，请参阅 Aurora MySQL 或 Aurora PostgreSQL 数据库集群的相应语言驱动程序。

通过命令行 AWS CLI 和 mysql 客户端，使用 IAM 身份验证连接到数据库集群

可以使用 AWS CLI 和 `mysql` 命令行工具从命令行连接到 Aurora 数据库集群，如下所述。

先决条件

以下是使用 IAM 身份验证连接到数据库集群的先决条件：

- 启用和禁用 IAM 数据库身份验证
- 创建和使用适用于 IAM 数据库访问的 IAM 策略
- 使用 IAM 身份验证创建数据库账户

> **Note**
>
> 有关使用具有 IAM 身份验证的 SQL WorkBench/J 连接到数据库的信息，请参阅博客文章使用 IAM 身份验证通过 SQL WorkBench/J 连接到 Aurora MySQL 或 Amazon RDS for MySQL。

主题

- 生成 IAM 身份验证令牌
- 连接到数据库集群

生成 IAM 身份验证令牌

以下示例说明了如何使用 AWS CLI 获取签名的身份验证令牌。

```
aws rds generate-db-auth-token \
   --hostname rdsmysql.123456789012.us-west-2.rds.amazonaws.com \
   --port 3306 \
   --region us-west-2 \
   --username jane_doe
```

在该示例中，参数如下所示：

- `--hostname` – 要访问的数据库集群的主机名
- `--port` – 用于连接到数据库集群的端口号
- `--region` – 在其中运行数据库集群的 AWS 区域
- `--username` – 要访问的数据库账户

令牌的前几个字符与以下内容类似。

```
rdsmysql.123456789012.us-west-2.rds.amazonaws.com:3306/?
Action=connect&DBUser=jane_doe&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Expires=900...
```

> **Note**
>
> 您不能使用自定义 Route 53 DNS 记录或 Aurora 自定义端点代替数据库集群端点来生成身份验证令牌。

连接到数据库集群

通用连接格式如下所示。

```
mysql --host=hostName --port=portNumber --ssl-ca=full_path_to_ssl_certificate --enable-
cleartext-plugin --user=userName --password=authToken
```

参数如下所示：

- `--host` – 要访问的数据库集群的主机名
- `--port` – 用于连接到数据库集群的端口号
- `--ssl-ca` – 包含公有密钥的 SSL 证书文件的完整路径

  有关更多信息，请参阅 与 Aurora MySQL 数据库集群的 TLS 连接。

  要下载 SSL 证书，请参阅 使用 SSL/TLS 加密与数据库集群的连接。
- `--enable-cleartext-plugin` – 一个指定 `AWSAuthenticationPlugin` 必须用于此连接的值

  如果您使用 MariaDB 客户端，则无需 `--enable-cleartext-plugin` 选项。
- `--user` – 要访问的数据库账户
- `--password` – 已签名的 IAM 身份验证令牌

身份验证令牌包含几百个字符。很难使用命令行对其进行处理。该问题的解决方式是，将令牌保存到一个环境变量中，然后在连接时使用此变量。以下示例说明了一种执行此解决方法的方式。在该示例中，*/sample_dir/* 是包含公有密钥的 SSL 证书文件的完整路径。

```
RDSHOST="mysqlcluster.cluster-123456789012.us-east-1.rds.amazonaws.com"
TOKEN="$(aws rds generate-db-auth-token --hostname $RDSHOST --port 3306 --region us-
west-2 --username jane_doe )"

mysql --host=$RDSHOST --port=3306 --ssl-ca=/sample_dir/global-bundle.pem --enable-
cleartext-plugin --user=jane_doe --password=$TOKEN
```

在使用 `AWSAuthenticationPlugin` 进行连接时，将使用 SSL 保护连接。要进行验证，请在 `mysql>` 命令提示符处键入以下内容。

```
show status like 'Ssl%';
```

输出中的以下行显示了更多详细信息。

```
+---------------+-------------+
| Variable_name | Value
                                                                    |
+---------------+-------------+
```

```
| ...           | ...
| Ssl_cipher    | AES256-SHA

                                                                    |
| ...           | ...
| Ssl_version   | TLSv1.1

                                                                    |
| ...           | ...
+-----------------------------+
```

如果您想通过代理连接到数据库集群，请参阅使用 IAM 身份验证连接到代理。

通过命令行使用 IAM 身份验证连接到数据库集群：AWS CLI 和 psql 客户端

可以使用 AWS CLI 和 psql 命令行工具从命令行连接到 Aurora PostgreSQL 数据库集群，如下所述。

先决条件

以下是使用 IAM 身份验证连接到数据库集群的先决条件：

- 启用和禁用 IAM 数据库身份验证
- 创建和使用适用于 IAM 数据库访问的 IAM 策略
- 使用 IAM 身份验证创建数据库账户

> **Note**
>
> 有关使用具有 IAM 身份验证的 pgAdmin 连接到数据库的信息，请参阅博客文章使用 IAM 身份验证通过 pgAdmin Amazon Aurora PostgreSQL 或 Amazon RDS for PostgreSQL 进行连接。

主题

- 生成 IAM 身份验证令牌
- 连接到 Aurora PostgreSQL 集群

生成 IAM 身份验证令牌

身份验证令牌包含几百个字符，因此，很难使用命令行对其进行处理。该问题的解决方式是，将令牌保存到一个环境变量中，然后在连接时使用此变量。以下示例说明了如何使用 AWS CLI 通过

generate-db-auth-token 命令获取签名的身份验证令牌，并将其存储在 PGPASSWORD 环境变量中。

```
export RDSHOST="mypostgres-cluster.cluster-123456789012.us-west-2.rds.amazonaws.com"
export PGPASSWORD="$(aws rds generate-db-auth-token --hostname $RDSHOST --port 5432 --
region us-west-2 --username jane_doe )"
```

在该示例中，generate-db-auth-token 命令的参数如下所示：

- --hostname – 要访问的数据库集群（集群终端节点）的主机名。
- --port – 用于连接到数据库集群的端口号
- --region – 在其中运行数据库集群的 AWS 区域
- --username – 要访问的数据库账户

生成的令牌的前几个字符与以下内容类似。

```
mypostgres-cluster.cluster-123456789012.us-west-2.rds.amazonaws.com:5432/?
Action=connect&DBUser=jane_doe&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Expires=900...
```

> **Note**
>
> 您不能使用自定义 Route 53 DNS 记录或 Aurora 自定义端点代替数据库集群端点来生成身份验证令牌。

连接到 Aurora PostgreSQL 集群

使用 psql 进行连接的一般格式如下所示。

```
psql "host=hostName port=portNumber sslmode=verify-full
 sslrootcert=full_path_to_ssl_certificate dbname=DBName user=userName
 password=authToken"
```

参数如下所示：

- host – 要访问的数据库集群（集群终端节点）的主机名。

- port – 用于连接到数据库集群的端口号
- sslmode – 要使用的 SSL 模式

  当您使用 sslmode=verify-full 时，SSL 连接将针对 SSL 证书中的终端节点验证数据库集群终端节点。
- sslrootcert – 包含公有密钥的 SSL 证书文件的完整路径

  有关更多信息，请参阅 利用 SSL/TLS 保护 Aurora PostgreSQL 数据。

  要下载 SSL 证书，请参阅 使用 SSL/TLS 加密与数据库集群的连接。
- dbname – 要访问的数据库
- user – 要访问的数据库账户
- password – 已签名的 IAM 身份验证令牌

> **Note**
>
> 您不能使用自定义 Route 53 DNS 记录或 Aurora 自定义端点代替数据库集群端点来生成身份验证令牌。

以下示例演示了使用 psql 进行连接。在该示例中，psql 将环境变量 RDSHOST 用于主机，将环境变量 PGPASSWORD 用于生产的令牌。*/sample_dir/* 则是包含公有密钥的 SSL 证书文件的完整路径。

```
export RDSHOST="mypostgres-cluster.cluster-123456789012.us-west-2.rds.amazonaws.com"
export PGPASSWORD="$(aws rds generate-db-auth-token --hostname $RDSHOST --port 5432 --
region us-west-2 --username jane_doe )"

psql "host=$RDSHOST port=5432 sslmode=verify-full sslrootcert=/sample_dir/global-
bundle.pem dbname=DBName user=jane_doe password=$PGPASSWORD"
```

如果您想通过代理连接到数据库集群，请参阅使用 IAM 身份验证连接到代理。

使用 IAM 身份验证和 适用于 .NET 的 AWS SDK 连接到数据库集群

您可以使用 适用于 .NET 的 AWS SDK 连接到 Aurora MySQL 或 Aurora PostgreSQL 数据库集群，如下所述。

先决条件

以下是使用 IAM 身份验证连接到数据库集群的先决条件：

- 启用和禁用 IAM 数据库身份验证
- 创建和使用适用于 IAM 数据库访问的 IAM 策略
- 使用 IAM 身份验证创建数据库账户

示例

以下代码示例演示如何生成身份验证令牌，然后使用该令牌连接到数据库集群。

要运行该代码示例，您需要使用 适用于 .NET 的 AWS SDK 网站上提供的 AWS。AWSSDK.CORE 和 AWSSDK.RDS 程序包是必需的。要连接到数据库集群，请使用用于数据库引擎的 .NET 数据库连接器，例如 MySqlConnector for MariaDB 或 MySQL，或 Npgsql for PostgreSQL。

此代码连接到 Aurora MySQL 数据库集群。根据需要修改以下变量的值：

- `server` – 要访问的数据库集群的终端节点。
- `user` – 要访问的数据库账户
- `database` – 要访问的数据库
- `port` – 用于连接到数据库集群的端口号
- `SslMode` – 要使用的 SSL 模式

  当您使用 `SslMode=Required` 时，SSL 连接将针对 SSL 证书中的终端节点验证数据库集群终端节点。
- `SslCa` – Amazon Aurora 的 SSL 证书的完整路径

  要下载证书，请参阅 使用 SSL/TLS 加密与数据库集群的连接。

> **Note**
>
> 您不能使用自定义 Route 53 DNS 记录或 Aurora 自定义端点代替数据库集群端点来生成身份验证令牌。

```
using System;
using System.Data;
using MySql.Data;
```

```
using MySql.Data.MySqlClient;
using Amazon;

namespace ubuntu
{
  class Program
  {
    static void Main(string[] args)
    {
      var pwd =
 Amazon.RDS.Util.RDSAuthTokenGenerator.GenerateAuthToken(RegionEndpoint.USEast1,
 "mysqlcluster.cluster-123456789012.us-east-1.rds.amazonaws.com", 3306, "jane_doe");
      // for debug only Console.Write("{0}\n", pwd);  //this verifies the token is
 generated

      MySqlConnection conn = new
 MySqlConnection($"server=mysqlcluster.cluster-123456789012.us-
east-1.rds.amazonaws.com;user=jane_doe;database=mydB;port=3306;password={pwd};SslMode=Required;
      conn.Open();

      // Define a query
      MySqlCommand sampleCommand = new MySqlCommand("SHOW DATABASES;", conn);

      // Execute a query
      MySqlDataReader mysqlDataRdr = sampleCommand.ExecuteReader();

      // Read all rows and output the first column in each row
      while (mysqlDataRdr.Read())
        Console.WriteLine(mysqlDataRdr[0]);

      mysqlDataRdr.Close();
      // Close connection
      conn.Close();
    }
  }
}
```

此代码连接到 Aurora PostgreSQL 数据库集群。

根据需要修改以下变量的值：

- `Server` – 要访问的数据库集群的终端节点。
- `User ID` – 要访问的数据库账户

- `Database` – 要访问的数据库
- `Port` – 用于连接到数据库集群的端口号
- `SSL Mode` – 要使用的 SSL 模式

  当您使用 `SSL Mode=Required` 时，SSL 连接将针对 SSL 证书中的终端节点验证数据库集群终端节点。
- `Root Certificate` – Amazon Aurora 的 SSL 证书的完整路径

  要下载证书，请参阅 使用 SSL/TLS 加密与数据库集群的连接。

> **Note**
>
> 您不能使用自定义 Route 53 DNS 记录或 Aurora 自定义端点代替数据库集群端点来生成身份验证令牌。

```
using System;
using Npgsql;
using Amazon.RDS.Util;

namespace ConsoleApp1
{
    class Program
    {
        static void Main(string[] args)
        {
            var pwd =
 RDSAuthTokenGenerator.GenerateAuthToken("postgresmycluster.cluster-123456789012.us-
east-1.rds.amazonaws.com", 5432, "jane_doe");
// for debug only Console.Write("{0}\n", pwd);  //this verifies the token is generated

            NpgsqlConnection conn = new
 NpgsqlConnection($"Server=postgresmycluster.cluster-123456789012.us-
east-1.rds.amazonaws.com;User Id=jane_doe;Password={pwd};Database=mydb;SSL
 Mode=Require;Root Certificate=full_path_to_ssl_certificate");
            conn.Open();

            // Define a query
                   NpgsqlCommand cmd = new NpgsqlCommand("select count(*) FROM
 pg_user", conn);
```

```
            // Execute a query
            NpgsqlDataReader dr = cmd.ExecuteReader();

            // Read all rows and output the first column in each row
            while (dr.Read())
                Console.Write("{0}\n", dr[0]);

            // Close connection
            conn.Close();
        }
    }
}
```

如果您想通过代理连接到数据库集群，请参阅使用 IAM 身份验证连接到代理。

使用 IAM 身份验证和 适用于 Go 的 AWS SDK 连接到数据库集群

您可以使用 适用于 Go 的 AWS SDK 连接到 Aurora MySQL 或 Aurora PostgreSQL 数据库集群，如下所述。

先决条件

以下是使用 IAM 身份验证连接到数据库集群的先决条件：

- 启用和禁用 IAM 数据库身份验证
- 创建和使用适用于 IAM 数据库访问的 IAM 策略
- 使用 IAM 身份验证创建数据库账户

示例

要运行上述代码示例，您需要使用 适用于 Go 的 AWS SDK 网站上提供的 AWS。

根据需要修改以下变量的值：

- `dbName` – 要访问的数据库
- `dbUser` – 要访问的数据库账户
- `dbHost` – 要访问的数据库集群的终端节点。

> **Note**
>
> 您不能使用自定义 Route 53 DNS 记录或 Aurora 自定义端点代替数据库集群端点来生成身份验证令牌。

- `dbPort` – 用于连接到数据库集群的端口号
- `region` – 在其中运行数据库集群的 AWS 区域

此外，请确保示例代码中的导入库存在于您的系统中。

> **Important**
>
> 本节中的示例使用以下代码提供从本地环境访问数据库的凭证：
> `creds := credentials.NewEnvCredentials()`
> 如果要从AWS服务（如 Amazon EC2 或 Amazon ECS）访问数据库，则可以用以下代码替换：
> `sess := session.Must(session.NewSession())`
> `creds := sess.Config.Credentials`
> 如果您进行此更改，请确保添加以下导入：
> `"github.com/aws/aws-sdk-go/aws/session"`

主题

- 使用 IAM 身份验证和 适用于 Go 的 AWS SDK V2 进行连接
- 使用 IAM 身份验证和 适用于 Go 的 AWS SDK V1 进行连接

使用 IAM 身份验证和 适用于 Go 的 AWS SDK V2 进行连接

您可以使用 IAM 身份验证和 适用于 Go 的 AWS SDK V2 连接到数据库集群

以下代码示例演示如何生成身份验证令牌，然后使用该令牌连接到数据库集群。

此代码连接到 Aurora MySQL 数据库集群。

```
package main

import (
```

```
     "context"
     "database/sql"
     "fmt"

     "github.com/aws/aws-sdk-go-v2/config"
     "github.com/aws/aws-sdk-go-v2/feature/rds/auth"
     _ "github.com/go-sql-driver/mysql"
)

func main() {

     var dbName string = "DatabaseName"
     var dbUser string = "DatabaseUser"
     var dbHost string = "mysqlcluster.cluster-123456789012.us-
east-1.rds.amazonaws.com"
     var dbPort int = 3306
     var dbEndpoint string = fmt.Sprintf("%s:%d", dbHost, dbPort)
     var region string = "us-east-1"

    cfg, err := config.LoadDefaultConfig(context.TODO())
    if err != nil {
     panic("configuration error: " + err.Error())
    }

    authenticationToken, err := auth.BuildAuthToken(
     context.TODO(), dbEndpoint, region, dbUser, cfg.Credentials)
    if err != nil {
     panic("failed to create authentication token: " + err.Error())
    }

    dsn := fmt.Sprintf("%s:%s@tcp(%s)/%s?tls=true&allowCleartextPasswords=true",
        dbUser, authenticationToken, dbEndpoint, dbName,
    )

    db, err := sql.Open("mysql", dsn)
    if err != nil {
        panic(err)
    }

    err = db.Ping()
    if err != nil {
        panic(err)
    }
```

```
}
```

此代码连接到 Aurora PostgreSQL 数据库集群。

```
package main

import (
     "context"
     "database/sql"
     "fmt"

     "github.com/aws/aws-sdk-go-v2/config"
     "github.com/aws/aws-sdk-go-v2/feature/rds/auth"
     _ "github.com/lib/pq"
)

func main() {

     var dbName string = "DatabaseName"
     var dbUser string = "DatabaseUser"
     var dbHost string = "postgresmycluster.cluster-123456789012.us-
east-1.rds.amazonaws.com"
     var dbPort int = 5432
     var dbEndpoint string = fmt.Sprintf("%s:%d", dbHost, dbPort)
     var region string = "us-east-1"

    cfg, err := config.LoadDefaultConfig(context.TODO())
    if err != nil {
     panic("configuration error: " + err.Error())
    }

    authenticationToken, err := auth.BuildAuthToken(
     context.TODO(), dbEndpoint, region, dbUser, cfg.Credentials)
    if err != nil {
     panic("failed to create authentication token: " + err.Error())
    }

    dsn := fmt.Sprintf("host=%s port=%d user=%s password=%s dbname=%s",
        dbHost, dbPort, dbUser, authenticationToken, dbName,
    )

    db, err := sql.Open("postgres", dsn)
    if err != nil {
```

```
        panic(err)
    }

    err = db.Ping()
    if err != nil {
        panic(err)
    }
}
```

如果您想通过代理连接到数据库集群，请参阅使用 IAM 身份验证连接到代理。

使用 IAM 身份验证和 适用于 Go 的 AWS SDK V1 进行连接

使用 IAM 身份验证和 适用于 Go 的 AWS SDK V1 连接到数据库集群

以下代码示例演示如何生成身份验证令牌，然后使用该令牌连接到数据库集群。

此代码连接到 Aurora MySQL 数据库集群。

```
package main

import (
    "database/sql"
    "fmt"
    "log"

    "github.com/aws/aws-sdk-go/aws/credentials"
    "github.com/aws/aws-sdk-go/service/rds/rdsutils"
    _ "github.com/go-sql-driver/mysql"
)

func main() {
    dbName := "app"
    dbUser := "jane_doe"
    dbHost := "mysqlcluster.cluster-123456789012.us-east-1.rds.amazonaws.com"
    dbPort := 3306
    dbEndpoint := fmt.Sprintf("%s:%d", dbHost, dbPort)
    region := "us-east-1"

    creds := credentials.NewEnvCredentials()
    authToken, err := rdsutils.BuildAuthToken(dbEndpoint, region, dbUser, creds)
    if err != nil {
        panic(err)
    }
```

```
    dsn := fmt.Sprintf("%s:%s@tcp(%s)/%s?tls=true&allowCleartextPasswords=true",
        dbUser, authToken, dbEndpoint, dbName,
    )

    db, err := sql.Open("mysql", dsn)
    if err != nil {
        panic(err)
    }

    err = db.Ping()
    if err != nil {
        panic(err)
    }
}
```

此代码连接到 Aurora PostgreSQL 数据库集群。

```
package main

import (
 "database/sql"
 "fmt"

 "github.com/aws/aws-sdk-go/aws/credentials"
 "github.com/aws/aws-sdk-go/service/rds/rdsutils"
 _ "github.com/lib/pq"
)

func main() {
    dbName := "app"
    dbUser := "jane_doe"
    dbHost := "postgresmycluster.cluster-123456789012.us-east-1.rds.amazonaws.com"
    dbPort := 5432
    dbEndpoint := fmt.Sprintf("%s:%d", dbHost, dbPort)
    region := "us-east-1"

    creds := credentials.NewEnvCredentials()
    authToken, err := rdsutils.BuildAuthToken(dbEndpoint, region, dbUser, creds)
    if err != nil {
        panic(err)
    }
```

```
    dsn := fmt.Sprintf("host=%s port=%d user=%s password=%s dbname=%s",
        dbHost, dbPort, dbUser, authToken, dbName,
    )

    db, err := sql.Open("postgres", dsn)
    if err != nil {
        panic(err)
    }

    err = db.Ping()
    if err != nil {
        panic(err)
    }
}
```

如果您想通过代理连接到数据库集群，请参阅使用 IAM 身份验证连接到代理。

使用 IAM 身份验证和 适用于 Java 的 AWS SDK 连接到数据库集群

您可以使用 适用于 Java 的 AWS SDK 连接到 Aurora MySQL 或 Aurora PostgreSQL 数据库集群，如下所述。

先决条件

以下是使用 IAM 身份验证连接到数据库集群的先决条件：

- 启用和禁用 IAM 数据库身份验证
- 创建和使用适用于 IAM 数据库访问的 IAM 策略
- 使用 IAM 身份验证创建数据库账户
- 设置适用于 Java 的AWS开发工具包

有关如何使用 SDK for Java 2.x 的示例，请参阅 Amazon RDS examples using SDK for Java 2.x。也可以使用 AWS 高级 JDBC 包装器，请参阅 AWS Advanced JDBC Wrapper documentation。

主题

- 生成 IAM 身份验证令牌
- 手动构造 IAM 身份验证令牌
- 连接到数据库集群

生成 IAM 身份验证令牌

如果您使用适用于 Java 的 AWS SDK编写程序，则可使用 RdsIamAuthTokenGenerator 类获取已签名的身份验证令牌。使用此类需要提供 AWS 凭证。为此，创建 DefaultAWSCredentialsProviderChain 类的实例。DefaultAWSCredentialsProviderChain 使用它在默认凭证提供程序链中找到的第一个 AWS 访问密钥和私有密钥。有关 AWS 访问密钥的更多信息，请参阅管理用户的访问密钥。

> **Note**
>
> 您不能使用自定义 Route 53 DNS 记录或 Aurora 自定义端点代替数据库集群端点来生成身份验证令牌。

在创建一个 RdsIamAuthTokenGenerator 实例后，您可调用 getAuthToken 方法以获取已签名的令牌。提供 AWS 区域、主机名、端口号和用户名。以下代码示例说明了如何执行该操作。

```
package com.amazonaws.codesamples;

import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
import com.amazonaws.services.rds.auth.GetIamAuthTokenRequest;
import com.amazonaws.services.rds.auth.RdsIamAuthTokenGenerator;

public class GenerateRDSAuthToken {

    public static void main(String[] args) {

     String region = "us-west-2";
     String hostname = "rdsmysql.123456789012.us-west-2.rds.amazonaws.com";
     String port = "3306";
     String username = "jane_doe";

     System.out.println(generateAuthToken(region, hostname, port, username));
    }

    static String generateAuthToken(String region, String hostName, String port, String
 username) {

     RdsIamAuthTokenGenerator generator = RdsIamAuthTokenGenerator.builder()
      .credentials(new DefaultAWSCredentialsProviderChain())
      .region(region)
      .build();
```

```
     String authToken = generator.getAuthToken(
      GetIamAuthTokenRequest.builder()
      .hostname(hostName)
      .port(Integer.parseInt(port))
      .userName(username)
      .build());

     return authToken;
    }

}
```

手动构造 IAM 身份验证令牌

在 Java 中，生成身份验证令牌的最简单方式是使用 RdsIamAuthTokenGenerator。此类将为您创建一个身份验证令牌，然后使用 AWS 签名版本 4 为该令牌签名。有关更多信息，请参阅《AWS 一般参考》中的签名版本 4 签名流程。

不过，您也可以手动构造身份验证令牌并为之签名，如以下代码示例中所示。

```
package com.amazonaws.codesamples;

import com.amazonaws.SdkClientException;
import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
import com.amazonaws.auth.SigningAlgorithm;
import com.amazonaws.util.BinaryUtils;
import org.apache.commons.lang3.StringUtils;

import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.nio.charset.Charset;
import java.security.MessageDigest;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.SortedMap;
import java.util.TreeMap;

import static com.amazonaws.auth.internal.SignerConstants.AWS4_TERMINATOR;
import static com.amazonaws.util.StringUtils.UTF8;

public class CreateRDSAuthTokenManually {
    public static String httpMethod = "GET";
```

```
    public static String action = "connect";
    public static String canonicalURIParameter = "/";
    public static SortedMap<String, String> canonicalQueryParameters = new TreeMap();
    public static String payload = StringUtils.EMPTY;
    public static String signedHeader = "host";
    public static String algorithm = "AWS4-HMAC-SHA256";
    public static String serviceName = "rds-db";
    public static String requestWithoutSignature;

    public static void main(String[] args) throws Exception {

        String region = "us-west-2";
        String instanceName = "rdsmysql.123456789012.us-west-2.rds.amazonaws.com";
        String port = "3306";
        String username = "jane_doe";

        Date now = new Date();
        String date = new SimpleDateFormat("yyyyMMdd").format(now);
        String dateTimeStamp = new
SimpleDateFormat("yyyyMMdd'T'HHmmss'Z'").format(now);
        DefaultAWSCredentialsProviderChain creds = new
DefaultAWSCredentialsProviderChain();
    String awsAccessKey = creds.getCredentials().getAWSAccessKeyId();
    String awsSecretKey = creds.getCredentials().getAWSSecretKey();
        String expiryMinutes = "900";

        System.out.println("Step 1:  Create a canonical request:");
        String canonicalString = createCanonicalString(username, awsAccessKey, date,
dateTimeStamp, region, expiryMinutes, instanceName, port);
        System.out.println(canonicalString);
        System.out.println();

        System.out.println("Step 2:  Create a string to sign:");
        String stringToSign = createStringToSign(dateTimeStamp, canonicalString,
awsAccessKey, date, region);
        System.out.println(stringToSign);
        System.out.println();

        System.out.println("Step 3:  Calculate the signature:");
        String signature = BinaryUtils.toHex(calculateSignature(stringToSign,
newSigningKey(awsSecretKey, date, region, serviceName)));
        System.out.println(signature);
        System.out.println();
```

```
        System.out.println("Step 4:  Add the signing info to the request");

        System.out.println(appendSignature(signature));
        System.out.println();

    }

    //Step 1: Create a canonical request date should be in format YYYYMMDD and dateTime
should be in format YYYYMMDDTHHMMSSZ
    public static String createCanonicalString(String user, String accessKey, String
date, String dateTime, String region, String expiryPeriod, String hostName, String
port) throws Exception {
        canonicalQueryParameters.put("Action", action);
        canonicalQueryParameters.put("DBUser", user);
        canonicalQueryParameters.put("X-Amz-Algorithm", "AWS4-HMAC-SHA256");
        canonicalQueryParameters.put("X-Amz-Credential", accessKey + "%2F" + date +
"%2F" + region + "%2F" + serviceName + "%2Faws4_request");
        canonicalQueryParameters.put("X-Amz-Date", dateTime);
        canonicalQueryParameters.put("X-Amz-Expires", expiryPeriod);
        canonicalQueryParameters.put("X-Amz-SignedHeaders", signedHeader);
        String canonicalQueryString = "";
        while(!canonicalQueryParameters.isEmpty()) {
            String currentQueryParameter = canonicalQueryParameters.firstKey();
            String currentQueryParameterValue =
canonicalQueryParameters.remove(currentQueryParameter);
            canonicalQueryString = canonicalQueryString + currentQueryParameter + "=" +
currentQueryParameterValue;
            if (!currentQueryParameter.equals("X-Amz-SignedHeaders")) {
                canonicalQueryString += "&";
            }
        }
        String canonicalHeaders = "host:" + hostName + ":" + port + '\n';
        requestWithoutSignature = hostName + ":" + port + "/?" + canonicalQueryString;

        String hashedPayload = BinaryUtils.toHex(hash(payload));
        return httpMethod + '\n' + canonicalURIParameter + '\n' + canonicalQueryString
+ '\n' + canonicalHeaders + '\n' + signedHeader + '\n' + hashedPayload;

    }

    //Step 2: Create a string to sign using sig v4
    public static String createStringToSign(String dateTime, String canonicalRequest,
String accessKey, String date, String region) throws Exception {
```

```
        String credentialScope = date + "/" + region + "/" + serviceName + "/
aws4_request";
        return algorithm + '\n' + dateTime + '\n' + credentialScope + '\n' +
 BinaryUtils.toHex(hash(canonicalRequest));

    }

    //Step 3: Calculate signature
    /**
     * Step 3 of the &AWS; Signature version 4 calculation. It involves deriving
     * the signing key and computing the signature. Refer to
     * http://docs.aws.amazon
     * .com/general/latest/gr/sigv4-calculate-signature.html
     */
    public static byte[] calculateSignature(String stringToSign,
                                            byte[] signingKey) {
        return sign(stringToSign.getBytes(Charset.forName("UTF-8")), signingKey,
                SigningAlgorithm.HmacSHA256);
    }

    public static byte[] sign(byte[] data, byte[] key,
                          SigningAlgorithm algorithm) throws SdkClientException {
        try {
            Mac mac = algorithm.getMac();
            mac.init(new SecretKeySpec(key, algorithm.toString()));
            return mac.doFinal(data);
        } catch (Exception e) {
            throw new SdkClientException(
                    "Unable to calculate a request signature: "
                            + e.getMessage(), e);
        }
    }

    public static byte[] newSigningKey(String secretKey,
                                   String dateStamp, String regionName, String
 serviceName) {
        byte[] kSecret = ("AWS4" + secretKey).getBytes(Charset.forName("UTF-8"));
        byte[] kDate = sign(dateStamp, kSecret, SigningAlgorithm.HmacSHA256);
        byte[] kRegion = sign(regionName, kDate, SigningAlgorithm.HmacSHA256);
        byte[] kService = sign(serviceName, kRegion,
                SigningAlgorithm.HmacSHA256);
        return sign(AWS4_TERMINATOR, kService, SigningAlgorithm.HmacSHA256);
    }
```

```
    public static byte[] sign(String stringData, byte[] key,
                      SigningAlgorithm algorithm) throws SdkClientException {
        try {
            byte[] data = stringData.getBytes(UTF8);
            return sign(data, key, algorithm);
        } catch (Exception e) {
            throw new SdkClientException(
                    "Unable to calculate a request signature: "
                            + e.getMessage(), e);
        }
    }

    //Step 4: append the signature
    public static String appendSignature(String signature) {
        return requestWithoutSignature + "&X-Amz-Signature=" + signature;
    }

    public static byte[] hash(String s) throws Exception {
        try {
            MessageDigest md = MessageDigest.getInstance("SHA-256");
            md.update(s.getBytes(UTF8));
            return md.digest();
        } catch (Exception e) {
            throw new SdkClientException(
                    "Unable to compute hash while signing request: "
                            + e.getMessage(), e);
        }
    }
}
```

连接到数据库集群

以下代码示例演示如何生成一个身份验证令牌，然后使用该令牌连接到运行 Aurora MySQL 的集群。

要运行该代码示例，您需要使用 适用于 Java 的 AWS SDK 网站上提供的 AWS。此外，您需要：

- MySQL Connector/J。此代码示例已使用 `mysql-connector-java-5.1.33-bin.jar` 进行测试。
- 特定于AWS区域的 Amazon Aurora 的中间证书。（有关更多信息，请参阅 使用 SSL/TLS 加密与数据库集群的连接。）在运行时，类加载程序会在与此 Java 代码示例相同的目录中查找证书以便可以找到它。
- 根据需要修改以下变量的值：

- RDS_INSTANCE_HOSTNAME – 要访问的数据库集群的主机名。
- RDS_INSTANCE_PORT – 用于连接到 PostgreSQL 数据库集群的端口号。
- REGION_NAME – 在其中运行数据库集群的AWS区域。
- DB_USER – 要访问的数据库账户
- SSL_CERTIFICATE – 特定于AWS区域的 Amazon Aurora 的中间证书。

  要为AWS区域下载证书，请参阅使用 SSL/TLS 加密与数据库集群的连接。将 SSL 证书放置在此 Java 程序文件所在的同一目录中，以便类加载程序可在运行时找到此证书。

此代码示例将从默认凭证提供程序链获取 AWS 凭证。

> **Note**
>
> 作为安全最佳实践，请为 DEFAULT_KEY_STORE_PASSWORD 指定除此处所示提示以外的密码。

```
package com.amazonaws.samples;

import com.amazonaws.services.rds.auth.RdsIamAuthTokenGenerator;
import com.amazonaws.services.rds.auth.GetIamAuthTokenRequest;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import java.io.File;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.security.KeyStore;
import java.security.cert.CertificateFactory;
import java.security.cert.X509Certificate;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.Properties;

import java.net.URL;
```

```
public class IAMDatabaseAuthenticationTester {
    //&AWS; Credentials of the IAM user with policy enabling IAM Database Authenticated
 access to the db by the db user.
    private static final DefaultAWSCredentialsProviderChain creds = new
 DefaultAWSCredentialsProviderChain();
    private static final String AWS_ACCESS_KEY =
 creds.getCredentials().getAWSAccessKeyId();
    private static final String AWS_SECRET_KEY =
 creds.getCredentials().getAWSSecretKey();

    //Configuration parameters for the generation of the IAM Database Authentication
 token
    private static final String RDS_INSTANCE_HOSTNAME = "rdsmysql.123456789012.us-
west-2.rds.amazonaws.com";
    private static final int RDS_INSTANCE_PORT = 3306;
    private static final String REGION_NAME = "us-west-2";
    private static final String DB_USER = "jane_doe";
    private static final String JDBC_URL = "jdbc:mysql://" + RDS_INSTANCE_HOSTNAME +
 ":" + RDS_INSTANCE_PORT;

    private static final String SSL_CERTIFICATE = "rds-ca-2019-us-west-2.pem";

    private static final String KEY_STORE_TYPE = "JKS";
    private static final String KEY_STORE_PROVIDER = "SUN";
    private static final String KEY_STORE_FILE_PREFIX = "sys-connect-via-ssl-test-
cacerts";
    private static final String KEY_STORE_FILE_SUFFIX = ".jks";
    private static final String DEFAULT_KEY_STORE_PASSWORD = "changeit";

    public static void main(String[] args) throws Exception {
        //get the connection
        Connection connection = getDBConnectionUsingIam();

        //verify the connection is successful
        Statement stmt= connection.createStatement();
        ResultSet rs=stmt.executeQuery("SELECT 'Success!' FROM DUAL;");
        while (rs.next()) {
             String id = rs.getString(1);
            System.out.println(id); //Should print "Success!"
        }

        //close the connection
        stmt.close();
```

```
        connection.close();

        clearSslProperties();

    }

    /**
     * This method returns a connection to the db instance authenticated using IAM
 Database Authentication
     * @return
     * @throws Exception
     */
    private static Connection getDBConnectionUsingIam() throws Exception {
        setSslProperties();
        return DriverManager.getConnection(JDBC_URL, setMySqlConnectionProperties());
    }

    /**
     * This method sets the mysql connection properties which includes the IAM Database
 Authentication token
     * as the password. It also specifies that SSL verification is required.
     * @return
     */
    private static Properties setMySqlConnectionProperties() {
        Properties mysqlConnectionProperties = new Properties();
        mysqlConnectionProperties.setProperty("verifyServerCertificate","true");
        mysqlConnectionProperties.setProperty("useSSL", "true");
        mysqlConnectionProperties.setProperty("user",DB_USER);
        mysqlConnectionProperties.setProperty("password",generateAuthToken());
        return mysqlConnectionProperties;
    }

    /**
     * This method generates the IAM Auth Token.
     * An example IAM Auth Token would look like follows:
     * btusi123.cmz7kenwo2ye.rds.cn-north-1.amazonaws.com.cn:3306/?
Action=connect&DBUser=iamtestuser&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-
Date=20171003T010726Z&X-Amz-SignedHeaders=host&X-Amz-Expires=899&X-Amz-
Credential=AKIAPFXHGVDI5RNFO4AQ%2F20171003%2Fcn-north-1%2Frds-db%2Faws4_request&X-Amz-
Signature=f9f45ef96c1f770cdad11a53e33ffa4c3730bc03fdee820cfdf1322eed15483b
     * @return
     */
    private static String generateAuthToken() {
```

```
        BasicAWSCredentials awsCredentials = new BasicAWSCredentials(AWS_ACCESS_KEY,
AWS_SECRET_KEY);

        RdsIamAuthTokenGenerator generator = RdsIamAuthTokenGenerator.builder()
                .credentials(new
AWSStaticCredentialsProvider(awsCredentials)).region(REGION_NAME).build();
        return generator.getAuthToken(GetIamAuthTokenRequest.builder()

.hostname(RDS_INSTANCE_HOSTNAME).port(RDS_INSTANCE_PORT).userName(DB_USER).build());
    }

    /**
     * This method sets the SSL properties which specify the key store file, its type
and password:
     * @throws Exception
     */
    private static void setSslProperties() throws Exception {
        System.setProperty("javax.net.ssl.trustStore", createKeyStoreFile());
        System.setProperty("javax.net.ssl.trustStoreType", KEY_STORE_TYPE);
        System.setProperty("javax.net.ssl.trustStorePassword",
DEFAULT_KEY_STORE_PASSWORD);
    }

    /**
     * This method returns the path of the Key Store File needed for the SSL
verification during the IAM Database Authentication to
     * the db instance.
     * @return
     * @throws Exception
     */
    private static String createKeyStoreFile() throws Exception {
        return createKeyStoreFile(createCertificate()).getPath();
    }

    /**
     *  This method generates the SSL certificate
     * @return
     * @throws Exception
     */
    private static X509Certificate createCertificate() throws Exception {
        CertificateFactory certFactory = CertificateFactory.getInstance("X.509");
        URL url = new File(SSL_CERTIFICATE).toURI().toURL();
        if (url == null) {
            throw new Exception();
```

```
        }
        try (InputStream certInputStream = url.openStream()) {
            return (X509Certificate) certFactory.generateCertificate(certInputStream);
        }
    }

    /**
     * This method creates the Key Store File
     * @param rootX509Certificate - the SSL certificate to be stored in the KeyStore
     * @return
     * @throws Exception
     */
    private static File createKeyStoreFile(X509Certificate rootX509Certificate) throws
 Exception {
        File keyStoreFile = File.createTempFile(KEY_STORE_FILE_PREFIX,
 KEY_STORE_FILE_SUFFIX);
        try (FileOutputStream fos = new FileOutputStream(keyStoreFile.getPath())) {
            KeyStore ks = KeyStore.getInstance(KEY_STORE_TYPE, KEY_STORE_PROVIDER);
            ks.load(null);
            ks.setCertificateEntry("rootCaCertificate", rootX509Certificate);
            ks.store(fos, DEFAULT_KEY_STORE_PASSWORD.toCharArray());
        }
        return keyStoreFile;
    }

    /**
     * This method clears the SSL properties.
     * @throws Exception
     */
    private static void clearSslProperties() throws Exception {
           System.clearProperty("javax.net.ssl.trustStore");
           System.clearProperty("javax.net.ssl.trustStoreType");
           System.clearProperty("javax.net.ssl.trustStorePassword");
    }

}
```

如果您想通过代理连接到数据库集群，请参阅使用 IAM 身份验证连接到代理。

使用 IAM 身份验证和 适用于 Python (Boto3) 的 AWS SDK 连接到数据库集群

您可以使用 适用于 Python (Boto3) 的 AWS SDK 连接到 Aurora MySQL 或 Aurora PostgreSQL 数据库集群，如下所述。

先决条件

以下是使用 IAM 身份验证连接到数据库集群的先决条件：

- 启用和禁用 IAM 数据库身份验证
- 创建和使用适用于 IAM 数据库访问的 IAM 策略
- 使用 IAM 身份验证创建数据库账户

此外，请确保示例代码中的导入库存在于您的系统中。

示例

该代码示例将配置文件用于共享凭证。有关指定凭证的信息，请参阅 适用于 Python (Boto3) 的 AWS SDK 文档中的凭证。

以下代码示例演示如何生成身份验证令牌，然后使用该令牌连接到数据库集群。

要运行该代码示例，您需要使用 适用于 Python (Boto3) 的 AWS SDK 网站上提供的 AWS。

根据需要修改以下变量的值：

- `ENDPOINT` – 要访问的数据库集群的终端节点。
- `PORT` – 用于连接到数据库集群的端口号
- `USER` – 要访问的数据库账户
- `REGION` – 在其中运行数据库集群的 AWS 区域
- `DBNAME` – 要访问的数据库
- `SSLCERTIFICATE` – Amazon Aurora 的 SSL 证书的完整路径

  对于 `ssl_ca`，请指定 SSL 证书。要下载 SSL 证书，请参阅 使用 SSL/TLS 加密与数据库集群的连接。

> **Note**
>
> 您不能使用自定义 Route 53 DNS 记录或 Aurora 自定义端点代替数据库集群端点来生成身份验证令牌。

此代码连接到 Aurora MySQL 数据库集群。

在运行此代码之前，请按照 Python 包索引中的说明安装 PyMySQL 驱动程序。

```
import pymysql
import sys
import boto3
import os

ENDPOINT="mysqlcluster.cluster-123456789012.us-east-1.rds.amazonaws.com"
PORT="3306"
USER="jane_doe"
REGION="us-east-1"
DBNAME="mydb"
os.environ['LIBMYSQL_ENABLE_CLEARTEXT_PLUGIN'] = '1'

#gets the credentials from .aws/credentials
session = boto3.Session(profile_name='default')
client = session.client('rds')

token = client.generate_db_auth_token(DBHostname=ENDPOINT, Port=PORT, DBUsername=USER,
 Region=REGION)

try:
    conn =
 pymysql.connect(auth_plugin_map={'mysql_clear_password':None},host=ENDPOINT,
 user=USER, password=token, port=PORT, database=DBNAME, ssl_ca='SSLCERTIFICATE',
 ssl_verify_identity=True, ssl_verify_cert=True)
    cur = conn.cursor()
    cur.execute("""SELECT now()""")
    query_results = cur.fetchall()
    print(query_results)
except Exception as e:
    print("Database connection failed due to {}".format(e))
```

此代码连接到 Aurora PostgreSQL 数据库集群。

在运行此代码之前，请按照 Psycopg 文档中的说明安装 psycopg2。

```
import psycopg2
import sys
import boto3
import os
```

```
ENDPOINT="postgresmycluster.cluster-123456789012.us-east-1.rds.amazonaws.com"
PORT="5432"
USER="jane_doe"
REGION="us-east-1"
DBNAME="mydb"

#gets the credentials from .aws/credentials
session = boto3.Session(profile_name='RDSCreds')
client = session.client('rds')

token = client.generate_db_auth_token(DBHostname=ENDPOINT, Port=PORT, DBUsername=USER,
 Region=REGION)

try:
    conn = psycopg2.connect(host=ENDPOINT, port=PORT, database=DBNAME, user=USER,
 password=token, sslrootcert="SSLCERTIFICATE")
    cur = conn.cursor()
    cur.execute("""SELECT now()""")
    query_results = cur.fetchall()
    print(query_results)
except Exception as e:
    print("Database connection failed due to {}".format(e))
```

如果您想通过代理连接到数据库集群，请参阅使用 IAM 身份验证连接到代理。

IAM 数据库身份验证故障排除

下面，您可以找到一些常见 IAM 数据库身份验证问题的故障排除思路，以及有关 IAM 数据库身份验证的 CloudWatch 日志和指标的信息。

将 IAM 数据库身份验证错误日志导出到 CloudWatch Logs

IAM 数据库身份验证错误日志存储在数据库主机上，您可以将这些日志导出到您的 CloudWatch Logs 账户。使用本页中的日志和补救方法来解决 IAM 数据库身份验证问题。

可以通过控制台、AWS CLI 和 RDS API 启用向 CloudWatch Logs 导出日志。有关控制台说明，请参阅 将数据库日志发布到 Amazon CloudWatch Logs。

要在从 AWS CLI 中创建数据库集群时将 IAM 数据库身份验证错误日志导出到 CloudWatch Logs，请使用以下命令：

```
aws rds create-db-cluster --db-cluster-identifier mydbinstance \
--region us-east-1 \
--engine postgres \
--engine-version 16 \
--master-username master \
--master-user-password password \
--publicly-accessible \
--enable-iam-database-authentication \
--enable-cloudwatch-logs-exports=iam-db-auth-error
```

要在从 AWS CLI 中修改数据库集群时将 IAM 数据库身份验证错误日志导出到 CloudWatch Logs，请使用以下命令：

```
aws rds modify-db-cluster --db-instance-identifier mydbcluster \
--region us-east-1 \
--cloudwatch-logs-export-configuration '{"EnableLogTypes":["iam-db-auth-error"]}'
```

要验证数据库集群是否正在将 IAM 数据库身份验证日志导出到 CloudWatch Logs，请检查 `describe-db-instances` 命令输出中的 `EnabledCloudwatchLogsExports` 参数是否设置为 `iam-db-auth-error`。

```
aws rds describe-db-cluster --region us-east-1 --db-cluster-identifier mydbcluster
            ...

             "EnabledCloudwatchLogsExports": [
                "iam-db-auth-error"
            ],
            ...
```

IAM 数据库身份验证 CloudWatch 指标

Amazon Aurora 向您的 Amazon CloudWatch 账户提供有关 IAM 数据库身份验证的近乎实时的指标。下表列出了使用 CloudWatch 时可用的 IAM 数据库身份验证指标：

指标	描述
`IamDbAuthConnectionRequests`	使用 IAM 数据库身份验证发出的连接请求总数。

指标	描述
IamDbAuthConnectionSuccess	成功的 IAM 数据库身份验证请求总数。
IamDbAuthConnectionFailure	失败的 IAM 数据库身份验证请求总数。
IamDbAuthConnectionFailureInvalidToken	由于令牌无效而失败的 IAM 数据库身份验证请求总数。
IamDbAuthConnectionFailureInsufficientPermissions	由于策略或权限不正确而失败的 IAM 数据库身份验证请求总数。
IamDbAuthConnectionFailureThrottling	由于 IAM 数据库身份验证节流而失败的 IAM 数据库身份验证请求总数。
IamDbAuthConnectionFailureServerError	由于 IAM 数据库身份验证功能中的内部服务器错误而失败的 IAM 数据库身份验证请求总数。

常见问题和解决方案

使用 IAM 数据库身份验证时可能会遇到以下问题。使用表中的修复步骤来解决问题：

错误	指标	原因	解决方案
[ERROR] Failed to authenticate the connection request for user *db_user* because the provided token is malformed or otherwise invalid. (Status Code: 400, Error Code: InvalidToken)	IamDbAuthConnectionFailure IamDbAuthConnectionFailureInvalidToken	连接请求中的 IAM 数据库身份验证令牌要么不是有效的 SigV4a 令牌，要么格式不正确。	在应用程序中检查令牌生成策略。在某些情况下，请确保使用有效的格式传递令牌。截断令牌（或字符串格式不正确）将使令牌失效。

错误	指标	原因	解决方案
[ERROR] Failed to authenticate the connection request for user *db_user* because the token age is longer than 15 minutes. (Status Code: 400, Error Code:ExpiredToken)	IamDbAuthConnectionFailure IamDbAuthConnectionFailureInvalidToken	IAM 数据库身份验证令牌已到期。令牌仅在 15 分钟内有效。	检查应用程序中的令牌缓存和/或令牌重用逻辑。您不应重用超过 15 分钟的令牌。
[ERROR] Failed to authorize the connection request for user *db_user* because the IAM policy assumed by the caller 'arn:aws:sts::123456789012:assumed-role/ <RoleName>/ <RoleSession>' is not authorized to perform `rds-db:connect` on the DB instance. (Status Code: 403, Error Code:NotAuthorized)	IamDbAuthConnectionFailure IamDbAuthConnectionFailureInsufficientPermissions	该错误可能是由于以下原因引起的： • 应用程序采用的 IAM 策略未授权执行 rds-db:connect 操作。 • 您为 *db_user* 采用的角色/策略不正确，无法连接到数据库。 • 您为 *db_user* 采用的策略是正确的，但是您没有连接到正确的数据库。	验证您在应用程序中采用的 IAM 角色和/或策略。请务必采用与连接到数据库相同的策略来生成令牌。

错误	指标	原因	解决方案
[ERROR] Failed to authorize the connection request for user *db_user* due to IAM DB authentication throttling. (Status Code: 429, Error Code: ThrottlingException)	IamDbAuthConnectionFailure IamDbAuthConnectionFailureThrottling	您在短时间内向数据库发出的连接请求过多。IAM 数据库身份验证节流限制为每秒 200 个连接。	降低使用 IAM 身份验证建立新连接的速率。考虑使用 RDS 代理来实现连接池，以便在应用程序中重用已建立的连接。
[ERROR] Failed to authorize the connection request for user *db_user* due to an internal IAM DB authentication error. (Status Code: 500, Error Code: InternalError)	IamDbAuthConnectionFailure IamDbAuthConnectionFailureThrottling	使用 IAM 数据库身份验证授权数据库连接时出现内部错误。	请联系 https://aws.amazon.com/premiumsupport/ 以便调查此问题。

对 Amazon Aurora 身份和访问权限问题进行故障排除

使用以下信息可帮助您诊断和修复在使用 Aurora 和 IAM 时可能遇到的常见问题。

主题

- 我无权在 Aurora 中执行操作
- 未授权我执行 iam:PassRole
- 我希望允许我 AWS 账户以外的人访问我的 Aurora 资源

我无权在 Aurora 中执行操作

如果 AWS Management Console 告诉您，您无权执行某个操作，则必须联系您的管理员寻求帮助。管理员是向您提供登录凭证的人。

当 `mateojackson` 用户尝试使用控制台查看有关 *`widget`* 的详细信息但不具有 `rds:`*`GetWidget`* 权限时，会发生以下示例错误。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 rds:GetWidget on resource: my-example-widget
```

在这种情况下，Mateo 请求他的管理员更新其策略，以允许他使用 `rds:`*`GetWidget`* 操作访问 *`my-example-widget`* 资源。

未授权我执行 iam:PassRole

如果您收到错误消息，提示您无权执行 `iam:PassRole` 操作，则必须联系您的管理员寻求帮助。管理员是向您提供登录凭证的人。请求该人员更新您的策略，以便允许您将角色传递给 Aurora。

有些 AWS 服务允许您将现有角色传递到该服务，而不是创建新服务角色或服务相关角色。为此，您必须具有将角色传递到服务的权限。

当名为 `marymajor` 的用户尝试使用控制台在 Aurora 中执行操作时，会发生以下示例错误。但是，服务必须具有服务角色所授予的权限才可执行操作。Mary 不具有将角色传递到服务的权限。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

在这种情况下，Mary 请求她的管理员来更新其策略，以允许她执行 `iam:PassRole` 操作。

我希望允许我 AWS 账户以外的人访问我的 Aurora 资源

您可以创建一个角色，以便其他账户中的用户或您组织外的人员可以使用该角色来访问您的资源。您可以指定谁值得信赖，可以担任角色。对于支持基于资源的策略或访问控制列表（ACL）的服务，您可以使用这些策略向人员授予对您的资源的访问权。

要了解更多信息，请参阅以下内容：

- 要了解 Aurora 是否支持这些功能，请参阅Amazon Aurora 如何与 IAM 协同工作。
- 要了解如何为您拥有的 AWS 账户中的资源提供访问权限，请参阅 IAM 用户指南中的为您拥有的另一个 AWS 账户中的 IAM 用户提供访问权限。

- 要了解如何为第三方 AWS 账户提供您的资源的访问权限，请参阅IAM 用户指南中的 为第三方拥有的 AWS 账户提供访问权限。
- 要了解如何通过联合身份验证提供访问权限，请参阅 IAM 用户指南中的为经过外部身份验证的用户（联合身份验证）提供访问权限。
- 要了解使用角色和基于资源的策略进行跨账户访问之间的差别，请参阅《IAM 用户指南》中的 IAM 角色与基于资源的策略有何不同。

Amazon Aurora 中的日志记录和监控

对于保持 Amazon Aurora 和您的 AWS 解决方案的可靠性、可用性和性能，监控发挥着非常重要的作用。您应该从 AWS 解决方案的各个部分收集监控数据，以便更轻松地调试多点故障（如果发生）。AWS 提供了多种工具来监控您的 Amazon Aurora 资源并对潜在事件做出响应：

Amazon CloudWatch 警报

使用 Amazon CloudWatch 警报，您可以在指定时间段内监控某个指标。如果指标超过给定阈值，则会向 Amazon SNS 主题或 AWS Auto Scaling 策略发送通知。CloudWatch 警报将不会调用操作，因为这些操作处于特定状态。而是必须在状态已改变并在指定的若干个时间段内保持不变后才调用。

AWS CloudTrail 日志

CloudTrail 提供了用户、角色或 AWS 服务在 Amazon Aurora 中所执行操作的记录。CloudTrail 将对 Amazon Aurora 的所有 API 调用作为事件捕获，包括来自控制台的调用和代码对 Amazon RDS API 操作的调用。通过使用 CloudTrail 收集的信息，您可以确定向 Amazon Aurora 发出了什么请求、发出请求的 IP 地址、何人发出的请求、请求的发出时间以及其他详细信息。有关更多信息，请参阅 监控 AWS CloudTrail 中的 Amazon Aurora API 调用。

增强监控

Amazon Aurora 针对运行数据库集群的操作系统（OS），实时提供操作系统的指标。您可以使用控制台查看数据库集群的指标，或者在您选择的监控系统中使用 Amazon CloudWatch Logs 的增强监控 JSON 输出。有关更多信息，请参阅 使用增强监控来监控操作系统指标。

Amazon RDS Performance Insights

Performance Insights 在现有 Amazon Aurora 监控功能的基础上进行了扩展，以便说明您的数据库的性能并帮助您分析影响性能的任何问题。利用 Performance Insights 控制面板，您可以可视化数据库负载并按等待状态、SQL 语句、主机或用户来筛选负载。有关更多信息，请参阅 在 Amazon Aurora 上使用性能详情监控数据库负载。

数据库日志

您可以使用 AWS Management Console、AWS CLI 或 RDS API 查看、下载和监控数据库日志。有关更多信息，请参阅 监控 Amazon Aurora 日志文件。

Amazon Aurora 建议

Amazon Aurora 可自动提供数据库资源建议。这些建议通过分析数据库集群配置、使用情况和性能数据来提供最佳实践准则。有关更多信息，请参阅 来自 Amazon Aurora 的建议。

Amazon Aurora 事件通知

Amazon Aurora 使用 Amazon Simple Notification Service（Amazon SNS）在发生 Amazon Aurora 事件时提供通知。这些通知可以采用AWS区域 Amazon SNS 支持的任何通知形式，例如电子邮件、文本消息或对 HTTP 终端节点的调用。有关更多信息，请参阅 使用 Amazon RDS 事件通知。

AWS Trusted Advisor

Trusted Advisor 凝聚了从为数十万 AWS 客户提供服务中总结的最佳实践。Trusted Advisor 可检查您的 AWS 环境，然后在有可能节省开支、提高系统可用性和性能或弥补安全漏洞时为您提供建议。所有 AWS 客户均有权访问五个 Trusted Advisor 检查。使用“商业”和“企业”支持计划的客户可以查看所有 Trusted Advisor 检查。

Trusted Advisor 提供与 Amazon Aurora 相关的下列检查：

- Amazon Aurora 闲置数据库实例
- Amazon Aurora 安全组访问风险
- Amazon Aurora 备份
- Amazon Aurora 多可用区
- Aurora 数据库实例可访问性

有关这些检查的更多信息，请参阅 Trusted Advisor 最佳实践（检查）。

数据库活动流

数据库活动流可通过控制 DBA 对数据库活动流的访问来保护您的数据库，使其免受内部威胁。因此，数据库活动流的收集、传输、存储和后续处理超出了管理数据库的 DBA 的访问权限。数据库活动流可为数据库提供安全保障，并满足合规性和法规要求。有关更多信息，请参阅使用数据库活动流监控 Amazon Aurora。

有关监控 Aurora 的更多信息，请参阅监控 Amazon Aurora 集群中的指标。

Amazon Aurora 的合规性验证

作为多个AWS合规性计划的一部分，第三方审计员将评估 Amazon Aurora 的安全性和合规性。其中包括 SOC、PCI、FedRAMP、HIPAA 及其他。

有关特定合规性计划范围内的 AWS 服务的列表，请参阅合规性计划范围内的 AWS 服务。有关一般信息，请参阅 AWS 合规性计划。

您可以使用 AWS Artifact 下载第三方审计报告。有关更多信息，请参阅在 AWS Artifact 中下载报告。

您在使用 Amazon Aurora 时的合规性责任由您的数据的敏感性、您组织的合规性目标以及适用的法律法规决定。AWS提供以下资源以帮助满足合规性要求：

- 安全性与合规性快速入门指南 – 这些部署指南讨论了架构注意事项，并提供了在AWS上部署基于安全性和合规性的基准环境的步骤。
- Amazon Web Services 上的 HIPAA 安全性和合规性架构设计 – 此白皮书介绍了公司如何使用 AWS 创建符合 HIPAA 标准的应用程序。
- AWS 合规性资源 – 可能适用于您的行业和位置的业务手册和指南集合。
- AWS Config – 此AWS服务评估您的资源配置对内部实践、行业指南和法规的遵循情况。
- AWS Security Hub – 此 AWS 服务 向您提供 AWS 中安全状态的全面视图。Security Hub 通过安全控件评估您的 AWS 资源并检查其是否符合安全行业标准和最佳实践。有关受支持服务及控件的列表，请参阅 Security Hub 控件参考。

Amazon Aurora 中的弹性

AWS全球基础设施围绕AWS区域和可用区构建。AWS区域提供多个在物理上独立且隔离的可用区，这些可用区通过延迟低、吞吐量高且冗余性高的网络连接在一起。利用可用区，您可以设计和操作在可用区之间无中断地自动实现故障转移的应用程序和数据库。与传统的单个或多个数据中心基础架构相比，可用区具有更高的可用性、容错性和可扩展性。

有关 AWS 区域和可用区的更多信息，请参阅 AWS 全球基础设施。

除了AWS全球基础设施之外，Aurora 还提供了相应功能来帮助支持您的数据弹性和备份需求。

备份与还原

Aurora 自动备份您的集群卷并将还原数据保留备份保留期的时长。Aurora 备份是连续且递增的，您可以快速还原到备份保留期内的任何时间点。在写入备份数据时，不会发生任何性能影响或数据库服务中断。在创建或修改数据库集群时，可指定备份保留期 (1 天到 35 天)。

如果希望备份的保留期超出备份保留期，还可为集群卷中的数据创建快照。Aurora 在整个备份保留期间，保留增量还原数据。因此，您只需为在备份保留期之外所要保留的数据创建快照。您可以从该快照创建新的数据库集群。

您可以从 Aurora 保留的备份数据或您保存的数据库集群快照创建新的 Aurora 数据库集群以恢复数据。您可以从备份保留期内任何时间点的备份数据快速创建新的数据库集群副本。备份保留期内的 Aurora 备份的持续和增量性质意味着您无需频繁创建数据快照来缩短还原时间。

有关更多信息，请参阅“备份和还原 Amazon Aurora 数据库集群”。

复制

Aurora 副本是 Aurora 数据库集群中的独立终端节点，最适合用于扩展读取操作以及提高可用性。对于数据库集群在AWS区域中所跨的多个可用区，最多可以分配 15 个 Aurora 副本。数据库集群卷由该数据库集群的多个数据副本组成。不过，对于主数据库实例和数据库集群中的 Aurora 副本，集群卷中的数据表示为单个逻辑卷。如果主数据库实例失败，Aurora 副本将提升为主数据库实例。

Aurora 还支持特定于 Aurora MySQL 和 Aurora PostgreSQL 的复制选项。

有关更多信息，请参阅“使用 Amazon Aurora 进行复制”。

故障转移

Aurora 跨一个AWS区域中的多个可用区将数据副本存储在数据库集群中。无论数据库集群中的数据库实例是否跨多个可用区，都会进行此存储。在跨可用区创建 Aurora 副本时，Aurora 会自动预置并同步

维护副本。主数据库实例将跨可用区同步复制到 Aurora 副本，以提供数据冗余、消除 I/O 冻结并在系统备份期间将延迟峰值降至最小。在计划内的系统维护期间，运行高度可用的数据库集群可以提高可用性，并帮助保护数据库以防发生故障和可用区中断。

有关更多信息，请参阅 Amazon Aurora 的高可用性。

Amazon Aurora 中的基础设施安全性

作为一项托管式服务，Amazon Relational Database Service 受 AWS 全球网络安全保护。有关 AWS 安全服务以及 AWS 如何保护基础设施的信息，请参阅 AWS 云安全。要按照基础架构安全最佳实践设计您的 AWS 环境，请参阅《安全性支柱 AWS Well-Architected Framework》中的 基础架构保护。

您可以使用 AWS 发布的 API 调用通过网络访问 Amazon RDS。客户端必须支持以下内容：

- 传输层安全性协议 (TLS) 我们要求使用 TLS 1.2，建议使用 TLS 1.3。
- 具有完全向前保密 (PFS) 的密码套件，例如 DHE（临时 Diffie-Hellman）或 ECDHE（临时椭圆曲线 Diffie-Hellman）。大多数现代系统（如 Java 7 及更高版本）都支持这些模式。

此外，必须使用访问密钥 ID 和与 IAM 主体关联的秘密访问密钥来对请求进行签名。或者，您可以使用 AWS Security Token Service（AWS STS）生成临时安全凭证来对请求进行签名。

此外，Aurora 提供了相应功能来帮助支持基础设施安全性。

安全组

安全组控制着流量在数据库集群内外拥有的访问权限。原定设置情况下，将对您的数据库集群关闭网络访问。您可以在安全组中指定规则，允许从 IP 地址范围、端口或安全组进行访问。配置传入规则后，会向与该安全组关联的所有数据库集群应用相同的规则。

有关更多信息，请参阅 使用安全组控制访问权限。

公开可用性

在基于 Amazon VPC 服务的虚拟私有云（VPC）内启动数据库实例时，可以打开或关闭该数据库实例的公共可访问性。要指定您创建的数据库实例是否具有解析为公共 IP 地址的 DNS 名称，请使用公共可用性 参数。通过使用此参数，您可以指定是否可以公开访问数据库实例。您可以通过修改 Public accessibility 参数修改数据库实例以打开或关闭公开可用性。

有关更多信息，请参阅“对互联网隐藏 VPC 中的数据库集群”。

> **Note**
>
> 如果您的数据库实例位于 VPC 中但不可公开访问，则您还可以使用AWS Site-to-Site VPN 连接或 AWS Direct Connect 连接从专用网络访问该实例。有关更多信息，请参阅 互联网络流量隐私。

Amazon RDS API 和接口 VPC 终端节点 (AWS PrivateLink)

您可以通过创建接口 VPC 终端节点 在 VPC 和 Amazon RDS API 终端节点之间建立私有连接。接口终端节点由 提供支持AWS PrivateLink

AWS PrivateLink 使您可以私下访问 Amazon RDS API 操作，而无需互联网网关、NAT 设备、VPN 连接或 AWS Direct Connect 连接。VPC 中的数据库实例不需要公有 IP 地址即可与 Amazon RDS API 端点进行通信，进而启动、修改或终止数据库实例和数据库集群。您的数据库实例也不需要公有 IP 地址即可使用任何可用的 RDS API 操作。您的 VPC 和 Amazon RDS 之间的流量不会脱离 Amazon 网络。

每个接口终端节点均由子网中的一个或多个弹性网络接口表示。有关弹性网络接口的更多信息，请参阅《Amazon EC2 用户指南》中的弹性网络接口。

有关 VPC 终端节点的更多信息，请参阅 Amazon VPC 用户指南中的接口 VPC 终端节点 (AWS PrivateLink)。有关 RDS API 操作的信息，请参阅 Amazon RDS API 参考。

您不需要接口 VPC 端点即可连接到数据库集群。有关更多信息，请参阅 在 VPC 中访问数据库集群的场景。

VPC 终端节点注意事项

在为 Amazon RDS API 终端节点设置接口 VPC 终端节点之前，请务必查看Amazon VPC 用户指南 中的接口终端节点属性和限制。

可以从使用 AWS PrivateLink 的 VPC 中获取所有与管理 Amazon Aurora 资源相关的 RDS API 操作。

RDS API 终端节点支持 VPC 终端节点策略。默认情况下，允许通过终端节点对 RDS API 操作进行完全访问。有关更多信息，请参阅 Amazon VPC 用户指南中的使用 VPC 终端节点控制对服务的访问。

可用性

Amazon RDS API 当前在以下AWS区域中支持 VPC 终端节点：

- US East (Ohio)
- 美国东部（弗吉尼亚州北部）
- 美国西部（加利福尼亚北部）
- 美国西部（俄勒冈州）

- 非洲（开普敦）
- 亚太地区（香港）
- Asia Pacific (Mumbai)
- 亚太地区（大阪）
- 亚太地区（首尔）
- 亚太地区（新加坡）
- 亚太地区（悉尼）
- 亚太地区（东京）
- 加拿大（中部）
- 加拿大西部（卡尔加里）
- 中国（北京）
- 中国（宁夏）
- 欧洲地区（法兰克福）
- 欧洲（苏黎世）
- 欧洲地区（爱尔兰）
- 欧洲地区（伦敦）
- 欧洲地区（巴黎）
- Europe (Stockholm)
- 欧洲地区（米兰）
- 以色列（特拉维夫）
- 中东（巴林）
- 南美洲（圣保罗）
- AWS GovCloud（美国东部）
- AWS GovCloud（美国西部）

为 Amazon RDS API 创建接口 VPC 终端节点

您可以使用 Amazon VPC 控制台或 AWS Command Line Interface (AWS CLI) 为 EBS 直接 Amazon RDS API 服务创建 VPC 终端节点。有关更多信息，请参阅 Amazon VPC 用户指南中的创建接口终端节点。

使用服务名称 `com.amazonaws.region.rds` 为 Amazon RDS API 创建 VPC 终端节点。

如果您为终端节点启用私有 DNS，则可以将其默认 DNS 名称用于AWS区域（例如 `rds.us-east-1.amazonaws.com`），从而通过 VPC 终端节点向 Amazon RDS 发出 API 请求（中国的AWS区域除外）。对于中国（北京）和中国（宁夏）AWS区域，您可以通过 VPC 终端节点分别使用 `rds-api.cn-north-1.amazonaws.com.cn` 和 `rds-api.cn-northwest-1.amazonaws.com.cn` 发出 API 请求。

有关更多信息，请参阅 Amazon VPC 用户指南中的通过接口终端节点访问服务。

为 Amazon RDS API 创建 VPC 终端节点策略

您可以为 VPC 终端节点附加控制对 Amazon RDS API 的访问的终端节点策略。该策略指定以下信息：

- 可执行操作的主体。
- 可执行的操作。
- 可对其执行操作的资源。

有关更多信息，请参阅 Amazon VPC 用户指南中的使用 VPC 终端节点控制对服务的访问。

示例：Amazon RDS API 操作的 VPC 终端节点策略

下面是用于 Amazon RDS API 的终端节点策略示例。当附加到终端节点时，此策略会向所有委托人授予对列出的针对所有资源的 Amazon RDS API 操作的访问权限。

```
{
   "Statement":[
      {
         "Principal":"*",
         "Effect":"Allow",
         "Action":[
            "rds:CreateDBInstance",
            "rds:ModifyDBInstance",
            "rds:CreateDBSnapshot"
         ],
         "Resource":"*"
      }
   ]
}
```

示例：拒绝来自指定 AWS 账户的所有访问的 VPC 终端节点策略

以下 VPC 端点策略会拒绝 AWS 账户 123456789012 所有使用端点访问资源的权限。此策略允许来自其他账户的所有操作。

```
{
  "Statement": [
    {
      "Action": "*",
      "Effect": "Allow",
      "Resource": "*",
      "Principal": "*"
    },
    {
      "Action": "*",
      "Effect": "Deny",
      "Resource": "*",
      "Principal": { "AWS": [ "123456789012" ] }
     }
   ]
}
```

Amazon Aurora 的安全最佳实践

使用 AWS Identity and Access Management（IAM）账户可控制对 Amazon RDS API 操作（特别是创建、修改或删除 Amazon Aurora 资源的操作）的访问权限。此类资源包括数据库集群、安全组和参数组。此外，使用 IAM 可控制执行常见管理任务的操作，例如备份和还原数据库集群。

- 为管理 Amazon Aurora 资源的每个人（包括您自己）创建一个单独的用户。请勿使用 AWS 根凭证管理 Amazon Aurora 资源。
- 授予每位用户执行其职责所需的最低权限集。
- 使用 IAM 组有效地管理适用于多个用户的权限。
- 定期轮换您的 IAM 凭证。
- 将 AWS Secrets Manager 配置为自动轮换 Amazon Aurora 的密钥。有关更多信息，请参阅 AWS Secrets Manager用户指南中的轮换 AWS Secrets Manager 密钥。您也可以从 AWS Secrets Manager 中以编程方式检索凭证。有关更多信息，请参阅 AWS Secrets Manager 用户指南 中的检索密钥值。

有关 Amazon Aurora 安全性的更多信息，请参阅 Amazon Aurora 中的安全性。有关 IAM 的更多信息，请参阅。AWS Identity and Access Management有关 IAM 最佳实践的信息，请参阅 IAM 最佳实践。

AWS Security Hub 使用安全控件来评估资源配置和安全标准，以帮助您遵守各种合规框架。有关使用 Security Hub 评估 Lambda 资源的更多信息，请参阅《AWS Security Hub 用户指南》中的 Amazon Relational Database Service 控件。

您可以监控 RDS 的使用情况，因为它与使用 Security Hub 的安全最佳实践有关。有关更多信息，请参阅什么是 AWS Security Hub？。

使用 AWS Management Console、AWS CLI 或 RDS API 更改主用户的密码。如果使用另一个工具（如 SQL 客户端）来更改主用户密码，则可能会无意中取消用户的权限。

Amazon GuardDuty 是一项持续的安全监控服务，可以分析和处理各种数据来源，包括 Amazon RDS 登录活动。它使用威胁情报源和机器学习来确定您的 AWS 环境中意外、可能未经授权、可疑的登录行为以及恶意活动。

当 Amazon GuardDuty RDS 保护检测到表示您的数据库中存在威胁的潜在可疑或异常登录尝试时，GuardDuty 会生成新的调查发现，其中包含有关可能被盗用的数据库的详细信息。有关更多信息，请参阅 使用适用于 Amazon Aurora 的 Amazon GuardDuty RDS 保护功能监控威胁。

使用安全组控制访问权限

VPC 安全组控制传入和传出数据库集群的流量拥有的访问权限。默认情况下，将为您的数据库集群关闭网络访问。您可以在安全组中指定规则，允许从 IP 地址范围、端口或安全组进行访问。配置传入规则后，会向与该安全组关联的所有数据库集群应用相同的规则。您最多可以在一个安全组中指定 20 个规则。

VPC 安全组概述

每个 VPC 安全组规则都使特定的源能够访问 VPC 中与该 VPC 安全组关联的数据库集群。源可以是一个地址范围 (例如，203.0.113.0/24) 或另一个 VPC 安全组。指定作为源的 VPC 安全组后，就可以允许从使用此源 VPC 安全组的所有实例 (通常为应用程序服务器) 中传入流量。VPC 安全组可能具有管理入站和出站流量的规则。但是，出站流量规则通常不适用于数据库集群。仅在数据库集群充当客户端时，出站流量规则才适用。您必须使用 Amazon EC2 API 或 VPC 控制台中的 Security Group（安全组）选项创建 VPC 安全组。

当您为 VPC 安全组创建允许访问 VPC 中的集群的规则时，必须为规则允许访问的每个地址范围指定一个端口。例如，如果您要对 VPC 中的实例开启 Secure Shell（SSH）访问，则为指定的地址范围创建一条允许访问 TCP 端口 22 的规则。

您可以为 VPC 中不同的实例配置允许访问不同端口的多个 VPC 安全组。例如，您可以创建一个允许访问您的 VPC 中的 Web 服务器的 TCP 端口 80 的 VPC 安全组。之后，您可以创建另一个 VPC 安全组，来允许访问您的 VPC 中的 Aurora MySQL 数据库实例的 TCP 端口 3306。

> **Note**
>
> 在 Aurora 数据库集群中，与数据库集群关联的 VPC 安全组也会与数据库集群中的所有数据库实例关联。如果您更改数据库集群或数据库实例的 VPC 安全组，更改将自动应用于数据库集群中的所有数据库实例。

有关 VPC 安全组的更多信息，请参阅 Amazon Virtual Private Cloud 用户指南 中的安全组。

> **Note**
>
> 如果您的数据库集群位于 VPC 中但不可公开访问，则您还可以使用 AWS Site-to-Site VPN 连接或 AWS Direct Connect 连接从专用网络访问它。有关更多信息，请参阅 互联网络流量隐私。

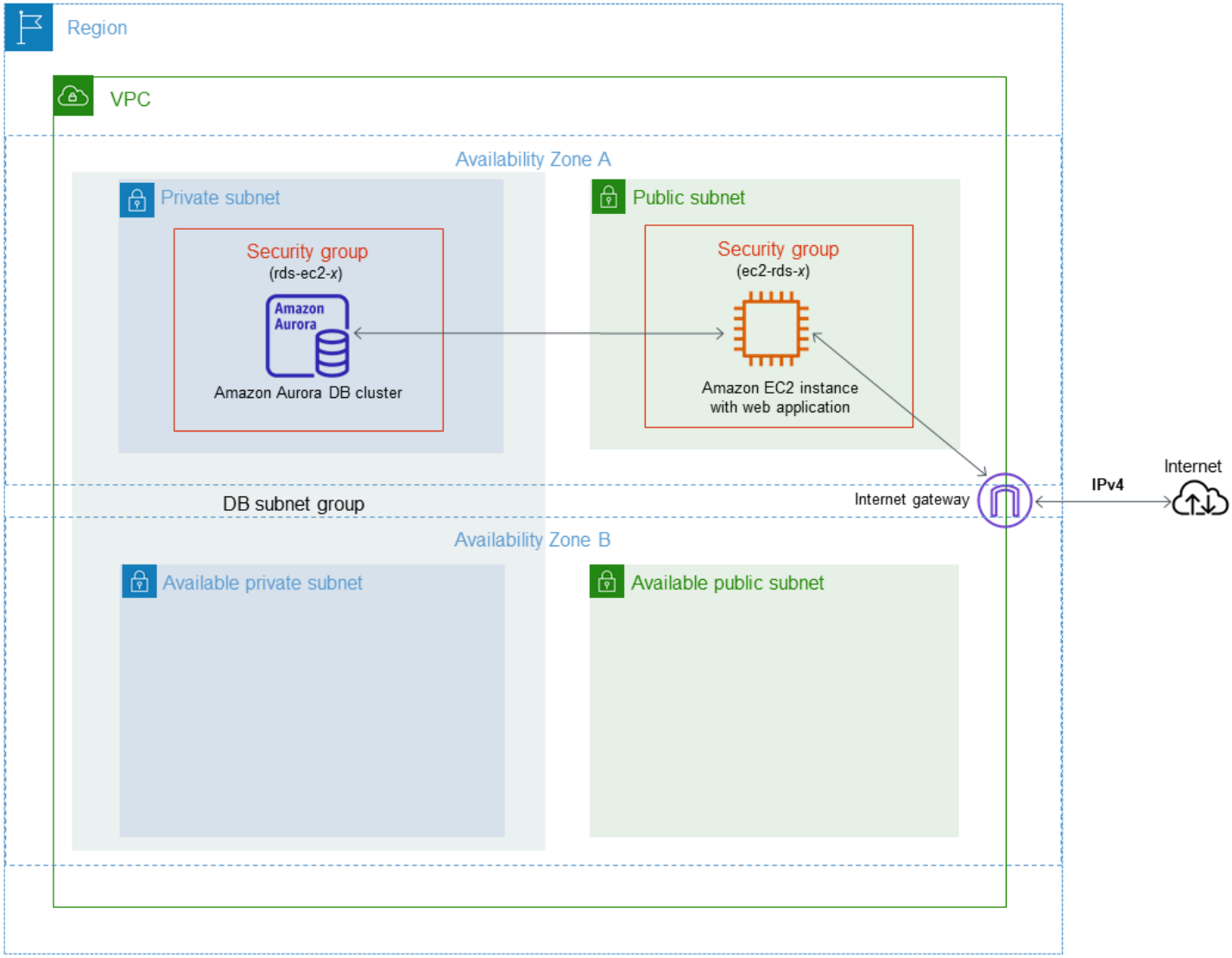
安全组情况

VPC 中的数据库集群的一种常见用法是，与运行在相同 VPC 中 Amazon EC2 实例上的应用程序服务器分享数据，这些数据可通过 VPC 外的客户端应用程序进行访问。对于此情况，使用 AWS Management Console上的 RDS 和 VPC 页面或 RDS 和 EC2 API 操作来创建必要的实例和安全组：

1. 创建一个 VPC 安全组 (例如，`sg-0123ec2example`)，然后定义使用客户端应用程序 IP 地址作为源的入站规则。通过此安全组，客户端应用程序可连接到使用此安全组的 VPC 中的 EC2 实例。
2. 创建一个适用于该应用程序的 EC2 实例，然后将该 EC2 实例添加到上一步中创建的 VPC 安全组 (`sg-0123ec2example`)。
3. 创建第二个 VPC 安全组 (例如，`sg-6789rdsexample`)，然后通过将步骤 1 中创建的 VPC 安全组 (`sg-0123ec2example`) 指定为源来创建一个新规则。

4. 创建一个新的数据库集群，然后将该数据库集群添加到在上一步骤中创建的 VPC 安全组（`sg-6789rdsexample`）。在创建数据库集群时，使用的端口号应与为您在步骤 3 中创建的 VPC 安全组（`sg-6789rdsexample`）规则指定的端口号相同。

下图说明了此情形。



有关针对此场景配置 VPC 的详细说明，请参阅教程：创建 VPC 以用于数据库集群（仅限 IPv4）。有关使用 VPC 的更多信息，请参阅 Amazon VPC 和 Amazon Aurora。

创建 VPC 安全组

您可以使用 VPC 控制台为数据库实例创建 VPC 安全组。有关创建安全组的信息，请参阅 Amazon Virtual Private Cloud 用户指南 中的通过创建安全组提供对 VPC 中数据库集群的访问和安全组。

将安全组与数据库集群关联

您可以使用 RDS 控制台的 Modify cluster (修改集群) 选项、`ModifyDBCluster` Amazon RDS API 或 `modify-db-cluster` AWS CLI 命令将安全组与数据库集群关联。

以下 CLI 示例关联特定 VPC 组并从数据库集群中移除数据库安全组

```
aws rds modify-db-cluster --db-cluster-identifier dbName --vpc-security-group-ids sg-ID
```

有关修改数据库集群的信息，请参阅修改 Amazon Aurora 数据库集群。

主用户账户权限

在创建一个新的数据库集群时，您使用的默认主用户将获得针对该数据库集群的特定权限。您无法在创建数据库集群之后更改主用户名。

> **Important**
>
> 我们强烈建议不要直接在应用程序中使用主用户。请遵守使用数据库用户的最佳实践，按照您的应用程序所需的最少权限创建用户。

> **Note**
>
> 如果您不小心删除了主用户的权限，要想恢复权限，您可以修改数据库集群并设置新的主用户密码。有关修改数据库集群的更多信息，请参阅修改 Amazon Aurora 数据库集群。

下表显示了主用户为每个数据库引擎获取的权限和数据库角色。

数据库引擎	系统权限	数据库角色
Aurora MySQL	版本 2： ALTER,ALTER ROUTINE, CREATE,CREATE ROUTINE,CREATE TEMPORARY TABLES,CREATE USER,CREATE VIEW,DELETE,DROP,	—

数据库引擎	系统权限	数据库角色
	EVENT,EXECUTE,GRANT OPTION,INDEX,INSERT, LOAD FROM S3,LOCK TABLES, PROCESS,REFERENCES , RELOAD,REPLICATI ON CLIENT , REPLICATION SLAVE ,SELECT, SELECT INTO S3,SHOW DATABASES , SHOW VIEW,TRIGGER, UPDATE	
	版本 3： ALTER,APPLICATION_PASSWORD_ADMIN , ALTER ROUTINE,CONNECTION_ADMIN , CREATE,CREATE ROLE,CREATE ROUTINE,CREATE TEMPORARY TABLES, CREATE USER,CREATE VIEW, DELETE,DROP,DROP ROLE, EVENT,EXECUTE,INDEX, INSERT,LOCK TABLES, PROCESS,REFERENCES , RELOAD,REPLICATION CLIENT , REPLICATI ON SLAVE ,ROLE_ADMIN , SET_USER_ ID ,SELECT,SHOW DATABASES ,SHOW VIEW,TRIGGER, UPDATE,XA_RECOVER_ADMIN 从 Aurora MySQL 版本 3.04.0 开始，主用户也将获得 SHOW_ROUTINE 权限。 从 Aurora MySQL 版本 3.09.0 开始，主用户也将获得 FLUSH_OPTIMIZER_COSTS 、FLUSH_STA TUS 、FLUSH_TABLES 和 FLUSH_USE R_RESOURCES 权限。	rds_superuser_role 有关 rds_superuser_role 的更多信息，请参阅基于角色的权限模型。
Aurora PostgreSQL	LOGIN,NOSUPERUSER , INHERIT,CREATEDB, CREATEROLE ,NOREPLICATION ,VALID UNTIL 'infinity'	RDS_SUPERUSER 有关 RDS_SUPERUSER 的更多信息，请参阅了解 PostgreSQL 角色和权限。

将服务相关角色用于 Amazon Aurora

将 AWS Identity and Access Management（IAM）服务相关角色用于 Amazon Aurora。服务相关角色是一种与 Amazon Aurora 直接关联的独特类型的 IAM 角色。服务相关角色由 Amazon Aurora 预定义，并包含该服务代表您调用其他AWS服务所需的一切权限。

服务相关角色可让您更轻松地使用 Amazon Aurora，因为您不必手动添加必要的权限。Amazon Aurora 定义其服务相关角色的权限，除非另外定义，否则只有 Amazon Aurora 可以代入该角色。定义的权限包括信任策略和权限策略，以及不能附加到任何其他 IAM 实体的权限策略。

只有在首先删除角色的相关资源后，才能删除角色。这将保护您的 Amazon Aurora 资源，因为您不会无意中删除对资源的访问权限。

有关支持服务相关角色的其他服务的信息，请参阅可与 IAM 搭配使用的 AWS 服务，并查找 Service-Linked Role（服务相关角色）列中为 Yes（是）的服务。选择是和链接，查看该服务的服务相关角色文档。

Amazon Aurora 的服务相关角色权限

Amazon Aurora 使用名为 AWSServiceRoleForRDS 的服务相关角色允许 Amazon RDS 代表您的数据库集群调用 AWS 服务。

AWSServiceRoleForRDS 服务相关角色信任以下服务以担任该角色：

- `rds.amazonaws.com`

此服务相关角色附加了一个名为 `AmazonRDSServiceRolePolicy` 的权限策略，授予其在您的账户中操作的权限。

有关此策略的更多信息，包括 JSON 策略文档，请参阅《AWS 托管式策略参考指南》中的 AmazonRDSServiceRolePolicy。

> **Note**
>
> 您必须配置权限，允许 IAM 实体（如用户、组或角色）创建、编辑或删除服务相关角色。如果您遇到以下错误消息：
> Unable to create the resource. Verify that you have permission to create service linked role。Otherwise wait and try again later.
> 确保您已启用以下权限：

```
{
    "Action": "iam:CreateServiceLinkedRole",
    "Effect": "Allow",
    "Resource": "arn:aws:iam::*:role/aws-service-role/rds.amazonaws.com/
AWSServiceRoleForRDS",
    "Condition": {
        "StringLike": {
            "iam:AWSServiceName":"rds.amazonaws.com"
        }
    }
}
```

有关更多信息，请参阅《IAM 用户指南》中的服务相关角色权限。

为 Amazon Aurora 创建服务相关角色

您无需手动创建服务相关角色。创建数据库集群时，Amazon Aurora 将为您创建服务相关角色。

> **Important**
>
> 如果在 2017 年 12 月 1 日（从此时开始支持服务相关角色）之前已使用 Amazon Aurora 服务，则 Amazon Aurora 已在您的账户中创建 AWSServiceRoleForRDS 角色。要了解更多信息，请参阅我的 AWS 账户中出现新角色。

如果您删除了此服务相关角色然后需要再次创建它，则可以使用相同的流程在您的账户中重新创建此角色。创建数据库集群时，Amazon Aurora 将再次为您创建服务相关角色。

为 Amazon Aurora 编辑服务相关角色

Amazon Aurora 不允许您编辑 AWSServiceRoleForRDS 服务相关角色。创建服务相关角色后，您将无法更改角色的名称，因为可能有多种实体引用该角色。但是可以使用 IAM 编辑角色描述。有关更多信息，请参阅《IAM 用户指南》中的编辑服务相关角色。

删除 Amazon Aurora 的服务相关角色

如果您不再需要使用某个需要服务相关角色的功能或服务，我们建议您删除该角色。这样您就没有未被主动监控或维护的未使用实体。但是，您必须先删除所有数据库集群才能删除服务相关角色。

清除服务相关角色

必须先确认服务相关角色没有活动会话并删除该角色使用的任何资源，然后才能使用 IAM 删除服务相关角色。

在 IAM 控制台中检查服务相关角色是否具有活动会话

1. 登录 AWS Management Console，打开 IAM 控制台：https://console.aws.amazon.com/iam/。
2. 在 IAM 控制台的导航窗格中，选择角色。然后选择 AWSServiceRoleForRDS 角色的名称（不是复选框）。
3. 在所选角色的摘要页面上，选择上次访问时间选项卡。
4. 在上次访问时间选项卡上，查看服务相关角色的近期活动。

> **Note**
>
> 如果您不确定 Amazon Aurora 是否正在使用 AWSServiceRoleForRDS 角色，可以尝试删除该角色。如果服务正在使用该角色，则删除操作会失败，并且您可以查看正在使用该角色的 AWS 区域。如果该角色已被使用，则您必须等待会话结束，然后才能删除该角色。您无法撤销服务相关角色对会话的权限。

如果您要删除 AWSServiceRoleForRDS 角色，必须先删除您的所有数据库集群。

删除所有集群

使用以下过程之一删除一个集群。重复过程删除每个集群。

删除一个集群 (控制台)

1. 通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在 Databases (数据库) 列表中，选择要删除的集群。
3. 对于 Cluster Actions (集群操作)，选择 Delete (删除)。
4. 选择删除。

删除一个集群 (CLI)

请参阅 AWS CLI Command Reference 中的 `delete-db-cluster`。

删除一个集群 (API)

请参阅 Amazon RDS API Reference 中的 DeleteDBCluster。

使用 IAM 控制台、IAM CLI 或 IAM API 删除 AWSServiceRoleForRDS 服务相关角色。有关更多信息，请参阅《IAM 用户指南》中的删除服务相关角色。

Amazon RDS 测试版的服务相关角色权限

Amazon Aurora 使用名为 AWSServiceRoleForRDSBeta 的服务相关角色，来支持 Amazon Aurora 代表 RDS 数据库资源调用 AWS 服务。

AWSServiceRoleForRDSBeta 服务相关角色信任以下服务来代入该角色：

- rds.amazonaws.com

此服务相关角色附加了一个名为 AmazonRDSBetaServiceRolePolicy 的权限策略，授予其在您的账户中操作的权限。有关更多信息，请参阅 AWS 托管式策略：AmazonRDSBetaServiceRolePolicy。

> **Note**
>
> 您必须配置权限，允许 IAM 实体（如用户、组或角色）创建、编辑或删除服务相关角色。如果您遇到以下错误消息：
> Unable to create the resource. Verify that you have permission to create service linked role。Otherwise wait and try again later.
> 确保您已启用以下权限：
>
> ```
> {
>     "Action": "iam:CreateServiceLinkedRole",
>     "Effect": "Allow",
>     "Resource": "arn:aws:iam::*:role/aws-service-role/custom.rds.amazonaws.com/
> AmazonRDSBetaServiceRolePolicy",
>     "Condition": {
>         "StringLike": {
>             "iam:AWSServiceName":"custom.rds.amazonaws.com"
>         }
>     }
> }
> ```
>
> 有关更多信息，请参阅《IAM 用户指南》中的服务相关角色权限。

Amazon RDS 预览版的服务相关角色

Amazon Aurora 使用名为 AWSServiceRoleForRDSPreview 的服务相关角色，来支持 Amazon Aurora 代表 RDS 数据库资源调用 AWS 服务。

AWSServiceRoleForRDSPreview 服务相关角色信任以下服务来代入该角色：

- rds.amazonaws.com

此服务相关角色附加了一个名为 AmazonRDSPreviewServiceRolePolicy 的权限策略，授予其在您的账户中操作的权限。有关更多信息，请参阅 AWS 托管式策略：AmazonRDSPreviewServiceRolePolicy。

> **Note**
>
> 您必须配置权限，允许 IAM 实体（如用户、组或角色）创建、编辑或删除服务相关角色。如果您遇到以下错误消息：
> Unable to create the resource. Verify that you have permission to create service linked role。Otherwise wait and try again later.
> 确保您已启用以下权限：
>
> ```
> {
>     "Action": "iam:CreateServiceLinkedRole",
>     "Effect": "Allow",
>     "Resource": "arn:aws:iam::*:role/aws-service-role/custom.rds.amazonaws.com/
> AmazonRDSPreviewServiceRolePolicy",
>     "Condition": {
>         "StringLike": {
>             "iam:AWSServiceName":"custom.rds.amazonaws.com"
>         }
>     }
> }
> ```
>
> 有关更多信息，请参阅《IAM 用户指南》中的服务相关角色权限。

Amazon VPC 和 Amazon Aurora

利用 Amazon Virtual Private Cloud（Amazon VPC），您可能能够在虚拟私有云（VPC）中启动 AWS 资源，如 Aurora 数据库集群。

使用 VPC 时，您的虚拟联网环境完全由您控制。您可以选择自己的 IP 地址范围、创建子网以及配置路由和访问控制列表。无需其他成本即可在 VPC 中运行数据库集群。

账户具有原定设置 VPC。除非您另行指定，否则所有新的数据库集群都将在原定设置 VPC 中创建。

主题

- 在 VPC 中使用数据库集群
- 在 VPC 中访问数据库集群的场景
- 教程：创建 VPC 以用于数据库集群（仅限 IPv4）
- 教程：创建 VPC 以用于数据库集群（双堆栈模式）

接下来，您可以查找与 Amazon Aurora 数据库集群相关的 VPC 功能的讨论。有关 Amazon VPC 的更多信息，请参阅 Amazon VPC 入门指南和 Amazon VPC 用户指南。

在 VPC 中使用数据库集群

您的数据库集群位于虚拟私有云（VPC）内。Amazon VPC 是一个虚拟网络，在逻辑上与AWS云中的其他虚拟网络相互隔离。使用 Amazon VPC，您可能能够将 AWS 资源（例如，Amazon Aurora 数据库集群或 Amazon EC2 实例），启动到 VPC 中。VPC 可以是您的账户附带的默认 VPC，也可以是您创建的 VPC。所有 VPC 均与您的 AWS 账户关联。

默认 VPC 具有可用来隔离 VPC 内资源的三个子网。默认 VPC 还具有一个互联网网关，可用来从 VPC 外部访问 VPC 内部的资源。

有关涉及 VPC 内的 Amazon Aurora 数据库集群的场景的列表，请参阅在 VPC 中访问数据库集群的场景。

主题

- 在 VPC 中使用数据库集群
- 使用数据库子网组
- 共享子网

- Amazon Aurora IP 寻址
- 对互联网隐藏 VPC 中的数据库集群
- 在 VPC 中创建数据库集群

在以下教程中，您可以学习创建可用于常见 Amazon Aurora 场景的 VPC：

- 教程：创建 VPC 以用于数据库集群（仅限 IPv4）
- 教程：创建 VPC 以用于数据库集群（双堆栈模式）

在 VPC 中使用数据库集群

下面是一些有关在 VPC 中使用数据库集群的提示：

- 您的 VPC 必须至少有两个子网。这些子网必须位于要部署数据库集群的 AWS 区域中两个不同的可用区。子网 是 VPC 的 IP 地址范围段，您可以指定子网，利用子网并根据安全和操作需要对数据库集群进行分组。
- 如果要让 VPC 中的数据库集群实现公开访问，请确保开启 VPC 属性 DNS hostnames（DNS 主机名）和 DNS resolution（DNS 解析）。
- 您的 VPC 必须具有您创建的数据库子网组。您可通过指定创建的子网来创建数据库子网组。Amazon Aurora 选择子网和该子网中的 IP 地址，以与数据库集群中的主数据库实例关联。主数据库实例使用包含该子网的可用区。
- 您的 VPC 必须具有允许访问数据库集群的 VPC 安全组。

  有关更多信息，请参阅 在 VPC 中访问数据库集群的场景。
- 每个子网必须包含足够大的 CIDR 数据块，以便在维护活动（包括故障转移和扩展计算）期间有可供 Amazon Aurora 使用的备用 IP 地址。例如，诸如 10.0.0.1.0/24 和 10.0.1.0/24 的范围通常足够大。
- VPC 的 instance tenancy 属性可能为 default 或 dedicated。所有默认 VPC 的“instance tenancy”属性设置为“default”，则默认的 VPC 可支持任何数据库实例类。

  如果您选择将数据库集群放在“instance tenancy”属性设置为“dedicated”的专用 VPC 中，则数据库集群的数据库实例类必须是已批准的 Amazon EC2 专用实例类型之一。例如，r5.large EC2 专用实例对应于 db.r5.large 数据库实例类。有关 VPC 中实例租期的信息，请参阅 Amazon Elastic Compute Cloud 用户指南 中的专用实例。

  有关可位于专用实例中的实例类型的更多信息，请参阅 Amazon EC2 定价页上的 Amazon EC2 专用实例。

> **Note**
>
> 当您将实例租赁属性设置为专用于数据库集群时，它不能保证数据库集群将在专属主机上运行。

使用数据库子网组

子网 是您指定的用来根据安全和操作需要对资源进行分组的 VPC 的 IP 地址范围段。数据库子网组 是您在 VPC 中创建并随后指定给数据库集群的子网（通常为私有子网）的集合。使用数据库子网组，您可以在使用 AWS CLI 或 RDS API 创建数据库集群时指定特定的 VPC。如果您使用控制台，则可以选择要使用的 VPC 和子网组。

每个数据库子网组应包含给定 AWS 区域中至少两个可用区的子网。在 VPC 中创建数据库集群时，为其选择数据库子网组。从数据库子网组中，Amazon Aurora 会选择子网和该子网中的 IP 地址（与数据库集群中的主数据库实例相关联）。数据库使用包含该子网的可用区。Aurora 始终从具有可用 IP 地址空间的子网中分配 IP 地址。

数据库子网组中的子网要么是公有子网，要么是私有子网。根据您为网络访问控制列表（网络 ACL）和路由表设置的配置，这些子网是公有子网或私有子网。要使数据库集群可公开访问，其数据库子网组中的所有子网必须均为公有子网。如果与可公开访问的数据库集群关联的子网从公有更改为私有，则可能会影响数据库集群的可用性。

要创建支持双堆栈模式的数据库子网组，请确保添加到该数据库子网组的每个子网都有一个与之关联的 Internet 协议版本 6 (IPv6) CIDR 块。有关更多信息，请参阅 Amazon VPC 用户指南中的 Amazon Aurora IP 寻址和迁移到 IPv6。

Amazon Aurora 在 VPC 中创建数据库集群时，它使用数据库子网组中的 IP 地址将网络接口分配给数据库集群。不过，我们强烈建议您使用域名系统（DNS）名称来连接数据库集群。之所以建议这样做，是因为底层 IP 地址在失效转移期间会发生变化。

> **Note**
>
> 对于在 VPC 中运行的每个数据库集群，请确保数据库子网组的每个子网中预留至少一个地址，以供 Amazon Aurora 用来执行恢复操作。

共享子网

您可以在共享 VPC 中创建数据库集群。

使用共享 VPC 时需要记住的一些注意事项：

- 您可以将数据库集群从共享 VPC 子网移动到非共享 VPC 子网，反之亦然。
- 共享 VPC 中的参与者必须在 VPC 中创建安全组，才能允许他们创建数据库集群。
- 共享 VPC 中的拥有者和参与者可以使用 SQL 查询访问数据库。但是，只有资源的创建者才能对该资源进行任何 API 调用。

Amazon Aurora IP 寻址

IP 地址使 VPC 中的资源能够相互通信以及与 Internet 上的资源进行通信。Amazon Aurora 同时支持 IPv4 和 IPv6 寻址协议。默认情况下，Amazon Aurora 和 Amazon VPC 使用 IPv4 寻址协议。您无法关闭这种行为。创建 VPC 时，请确保指定 IPv4 CIDR 块 (一系列私有 IPv4 地址)。您可以选择将 IPv6 CIDR 块分配给您的 VPC 和子网，并将来自该块的 IPv6 地址分配给您子网中的数据库集群。

对 IPv6 协议的支持扩展了支持的 IP 地址数量。通过使用 IPv6 协议，您可以确保有足够的可用地址来应对 Internet 的未来发展。新的和现有 RDS 资源可以在 VPC 内使用 IPv4 和 IPv6 地址。在应用程序不同部分使用的两个协议之间配置、保护和转换网络流量可能会产生运营开销。您可以对 Amazon RDS 资源的 IPv6 协议进行标准化，以简化网络配置。

主题

- IPv4 地址
- IPv6 地址
- 双堆栈模式

IPv4 地址

创建 VPC 时，您必须以 CIDR 块（如 `10.0.0.0/16`）的形式为 VPC 指定一个 IPv4 地址范围。数据库子网组 定义此 CIDR 块中可供数据库集群使用的 IP 地址范围。这些 IP 地址可以是私有地址，也可以是公有地址。

私有 IPv4 地址是指无法通过 Internet 访问的 IP 地址。您可以使用私有 IPv4 地址在同一 VPC 中的数据库集群和其他资源（如 Amazon EC2 实例）之间进行通信。每个数据库集群都有一个用于在 VPC 中通信的私有 IP 地址。

公有 IP 地址是指可通过 Internet 访问的 IPv4 地址。您可以使用公有地址在数据库集群和 Internet 上的资源（如 SQL 客户端）之间进行通信。您可以控制数据库集群是否接收公有 IP 地址。

有关说明如何创建可用于常见 Amazon Aurora 场景的 VPC（只有私有 IPv4 地址）的教程，请参阅 教程：创建 VPC 以用于数据库集群（仅限 IPv4）。

IPv6 地址

您可以选择向 VPC 和子网关联 IPv6 CIDR 块，然后将此块中的 IPv6 地址分配给 VPC 中的资源。每个 IPv6 地址都是全局唯一的。

我们将自动从 Amazon 的 IPv6 地址池中为您的 VPC 分配 IPv6 CIDR 块。您不能自行选择范围。

连接到 IPv6 地址时，请确保满足以下条件：

- 客户端配置为允许通过 IPv6 进行客户端到数据库的通信。
- 数据库实例使用的 RDS 安全组已正确配置，允许通过 IPv6 进行客户端到数据库的通信。
- 客户端操作系统堆栈允许 IPv6 地址上有流量，操作系统驱动程序和库已配置为选择正确的默认数据库实例终端节点（IPv4 或 IPv6）。

有关 IPv6 的更多信息，请参阅 Amazon VPC 用户指南中的 IP 寻址。

双堆栈模式

当数据库集群可以通过 IPv4 和 IPv6 寻址协议进行通信时，它在双堆栈模式下运行。因此，资源可以通过 IPv4 和/或 IPv6 与数据库集群进行通信。RDS 禁用互联网网关访问私有双堆栈模式数据库实例的 IPv6 端点。RDS 这样做是为了确保您的 IPv6 端点是私有的，并且只能从您的 VPC 内部对其进行访问。

主题

- 双堆栈模式和数据库子网组
- 使用双堆栈模式数据库实例
- 修改仅限 IPv4 的数据库集群以使用双堆栈模式
- 双堆栈网络数据库集群的可用性
- 双堆栈网络数据库集群的限制

有关说明如何创建您可用于常见 Amazon Aurora 场景的 VPC（具有 IPv4 和 IPv6 地址）的教程，请参阅 教程：创建 VPC 以用于数据库集群（双堆栈模式）。

双堆栈模式和数据库子网组

要使用双堆栈模式，请确保与数据库集群关联的数据库子网组中的每个子网都具有与之关联的 IPv6 CIDR 块。您可以创建新的数据库子网组或修改现有数据库子网组来满足此要求。当数据库集群处于双堆栈模式后，客户端可以与它正常连接。确保准确配置客户端安全防火墙和 RDS 数据库实例安全组，以允许通过 IPv6 的流量。要进行连接，客户端使用数据库集群主实例的端点。客户端应用程序可以指定连接到数据库时首选哪种协议。在双堆栈模式下，数据库集群会检测客户端的首选网络协议（IPv4 或 IPv6），并使用该协议进行连接。

如果某数据库子网组因子网删除或 CIDR 断开关联而停止支持双堆栈模式，则与该数据库子网组关联的数据库实例存在网络状态不兼容的风险。此外，创建新的双堆栈模式数据库集群时，您不能使用数据库子网组。

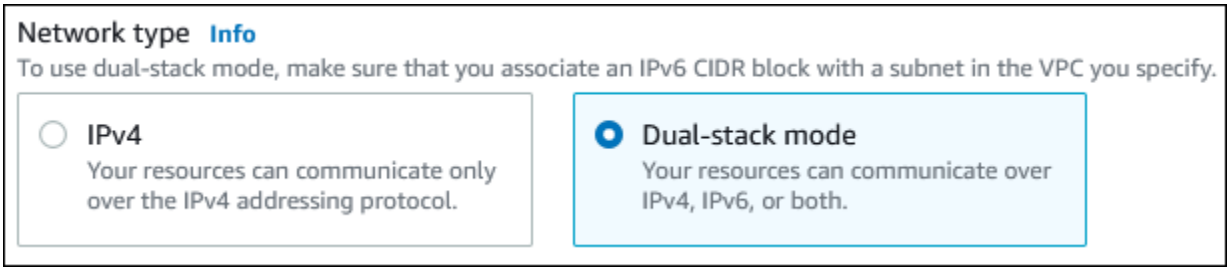
要使用 AWS Management Console 确定数据库子网组是否支持双堆栈模式，请查看数据库子网组的详细信息页面上的 Network type（网络类型）。要使用 AWS CLI 确定数据库子网组是否支持双堆栈模式，请运行 describe-db-subnet-groups 命令并查看输出中的 `SupportedNetworkTypes`。

只读副本被视为独立的数据库实例，并且可以具有与主数据库实例不同的网络类型。如果您更改只读副本的主数据库实例的网络类型，则只读副本不会受到影响。当您还原数据库实例时，可以将其还原为支持的任何网络类型。

使用双堆栈模式数据库实例

创建或修改数据库集群时，您可以指定双堆栈模式，以允许您的资源通过 IPv4 和/或 IPv6 与数据库集群进行通信。

当您使用 AWS Management Console 创建或修改数据库实例时，可以在 Network type（网络类型）部分中指定双堆栈模式。下图显示了控制台中的 Network type（网络类型）部分。



当您使用 AWS CLI 创建或修改数据库集群时，请将 `--network-type` 选项设置为 `DUAL` 以使用双堆栈模式。当您使用 RDS API 创建或修改数据库集群时，请将 `NetworkType` 参数设置为 `DUAL` 以使用双堆栈模式。当您修改数据库实例的网络类型时，可能会出现停机。如果指定的数据库引擎版本或数据库子网组不支持双堆栈模式，则返回 `NetworkTypeNotSupported` 错误。

有关创建数据库集群的更多信息，请参阅创建 Amazon Aurora 数据库集群。有关修改数据库集群的更多信息，请参阅 修改 Amazon Aurora 数据库集群。

要使用控制台确定数据库集群是否处于双堆栈模式，请查看数据库集群的 Connectivity & security（连接和安全）选项卡上的 Network type（网络类型）。

修改仅限 IPv4 的数据库集群以使用双堆栈模式

您可以修改仅限 IPv4 的数据库集群以使用双堆栈模式。为此，请更改数据库集群的网络类型。修改可能会导致停机。

建议您在维护时段内更改 Amazon Aurora 数据库集群的网络类型。目前，不支持将新实例的网络类型设置为双堆栈模式。您可以使用 `modify-db-cluster` 命令手动设置网络类型。

修改数据库集群以使用双堆栈模式之前，请确保其数据库子网组支持双堆栈模式。如果与数据库集群关联的数据库子网组不支持双堆栈模式，请在修改数据库集群时指定支持该模式的其他数据库子网组。修改数据库集群的数据库子网组可能会导致停机。

如果在将数据库集群更改为使用双堆栈模式之前，修改数据库集群的数据库子网组，请确保该数据库子网组在更改前后对数据库集群有效。

我们建议您只使用值为 `DUAL` 的 `--network-type` 参数运行 modify-db-cluster API，以将 Amazon Aurora 集群的网络更改为双堆栈模式。在同一 API 调用中将其他参数与 `--network-type` 参数一起添加可能会导致停机。

如果更改后无法连接到数据库集群，请确保已准确配置客户端和数据库安全防火墙和路由表，以允许流量流向选定网络（IPv4 或 IPv6）上的数据库。您可能还需要修改操作系统参数、库或驱动程序才能使用 IPv6 地址进行连接。

修改仅限 IPv4 的数据库集群以使用双堆栈模式

1. 修改数据库子网组以支持双堆栈模式，或者创建支持双堆栈模式的数据库子网组：

   a. 将 IPv6 CIDR 块与 VPC 关联。

      有关说明，请参阅《Amazon VPC 用户指南》中的将 IPv6 CIDR 块添加到 VPC。

   b. 将 IPv6 CIDR 块附加到数据库子网组中的所有子网。

      有关说明，请参阅《Amazon VPC 用户指南》中的将 IPv6 CIDR 块添加到子网。

   c. 确认数据库子网组支持双堆栈模式。

如果您使用 AWS Management Console，请选择数据库子网组，并确保 Supported network types（支持的网络类型）值为 Dual, IPv4（双，IPv4）。

如果您使用 AWS CLI，请运行 describe-db-subnet-groups 命令，并确保数据库实例的 `SupportedNetworkType` 值为 `Dual, IPv4`。

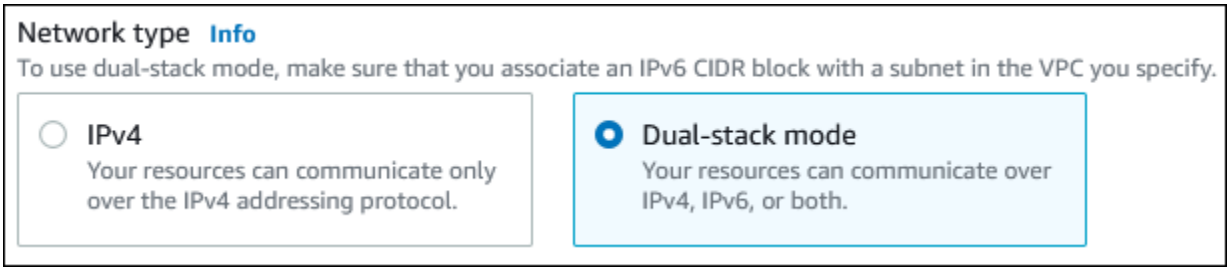
2. 修改与数据库集群关联的安全组以允许与数据库建立 IPv6 连接，或者新建允许 IPv6 连接的安全组。

   有关说明，请参阅 Amazon VPC 用户指南中的安全组规则。

3. 修改数据库集群以支持双堆栈模式。为此，请将 Network type（网络类型）设置为 Dual-stack mode（双堆栈模式）。

   如果您使用控制台，请确保以下设置正确：

   - Network type（网络类型）– Dual-stack mode（双堆栈模式）

   - DB subnet group（数据库子网组）– 在上一步中配置的数据库子网组
   - Security group（安全组）– 在上一步中配置的安全组

   如果您使用 AWS CLI，请确保以下设置正确：

   - `--network-type` – `dual`
   - `--db-subnet-group-name` – 在上一步中配置的数据库子网组
   - `--vpc-security-group-ids` – 在上一步中配置的 VPC 安全组

   例如：

   ```
   aws rds modify-db-cluster --db-cluster-identifier my-cluster --network-type "DUAL"
   ```

4. 确认数据库集群支持双堆栈模式。

如果您使用控制台，请对于数据库集群选择 Configuration（配置）选项卡。在该选项卡上，确保 Network type（网络类型）值为 Dual-stack mode（双堆栈模式）。

如果您使用 AWS CLI，请运行 describe-db-clusters 命令，并确保数据库集群的 `NetworkType` 值为 `dual`。

在写入器数据库实例端点上运行 `dig` 命令，以标识与其关联的 IPv6 地址。

```
dig db-instance-endpoint AAAA
```

使用写入器数据库实例端点（而不是 IPv6 地址）连接到数据库集群。

双堆栈网络数据库集群的可用性

双堆栈网络数据库集群在除以下区域外的所有 AWS 区域中都可用：

- 亚太地区（海得拉巴）
- 亚太地区（马来西亚）
- 亚太地区（墨尔本）
- 加拿大西部（卡尔加里）
- 欧洲（西班牙）
- 欧洲（苏黎世）
- 以色列（特拉维夫）
- 中东（阿联酋）

以下数据库引擎版本支持双堆栈网络数据库集群：

- Aurora MySQL 版本：
  - 3.02 及更高的 3 版本
  - 2.09.1 及更高的 2 版本

  有关 Aurora MySQL 版本的更多信息，请参阅 Aurora MySQL 版本注释。
- Aurora PostgreSQL 版本：
  - 15.2 及所有更高版本
  - 14.3 及更高的 14 版本

- 13.7 及更高的 13 版本

有关 Aurora PostgreSQL 版本的更多信息，请参阅 Aurora PostgreSQL 版本注释。

双堆栈网络数据库集群的限制

以下限制适用于双堆栈网络数据库集群：

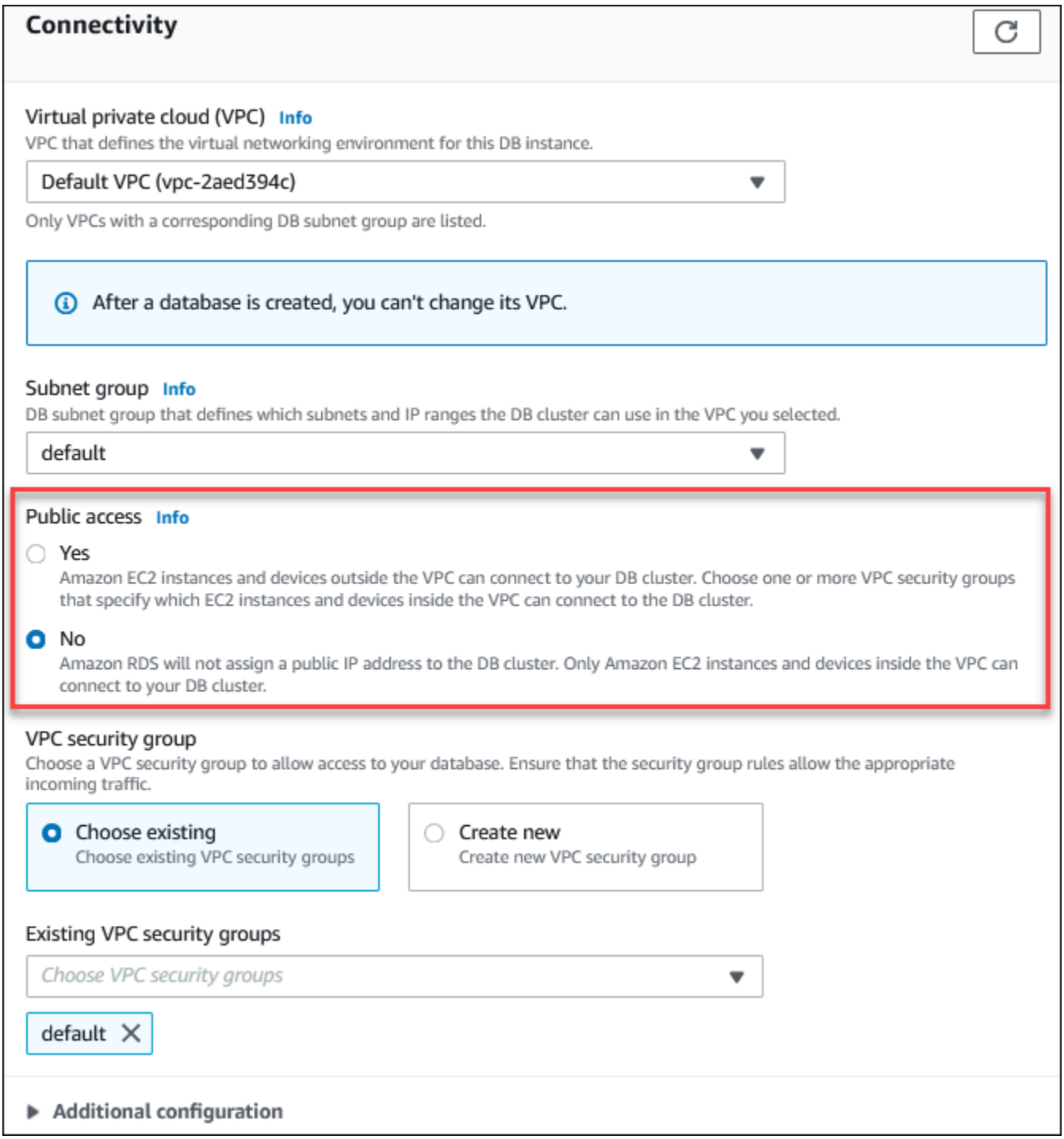
- 数据库集群不能仅使用 IPv6 协议。它们可以仅使用 IPv4，也可以同时使用 IPv4 和 IPv6 协议（双堆栈模式）。
- Amazon RDS 不支持本机 IPv6 子网。
- 使用双堆栈模式的数据库集群必须是私有的。它们不可公开访问。
- 不能将 RDS 代理与双堆栈模式数据库集群一起使用。

对互联网隐藏 VPC 中的数据库集群

一个常见的 Amazon Aurora 场景是具有一个 VPC，其中有一个 Amazon EC2 实例（具有一个面向公众的 Web 应用程序）和一个数据库集群（具有一个不能公开访问的数据库）。例如，您可创建包含公有子网和私有子网的 VPC。可将充当 Web 服务器的 EC2 实例部署在公有子网中。数据库集群部署在私有子网中。在此部署方案中，只有 Web 服务器才能访问数据库集群。有关此方案的说明，请参阅 VPC 中的数据库集群由同一 VPC 中的 Amazon EC2 实例访问。

当您在 VPC 中启动数据库集群时，该数据库集群具有用于 VPC 内流量的私有 IP 地址。此私有 IP 地址不可公开访问。您可以使用 Public access（公有访问权限）选项指定数据库集群除了私有 IP 地址之外是否还具有公有 IP 地址。如果将数据库集群指定为可公开访问，则其 DNS 端点解析为 VPC 内部的私有 IP 地址。它从 VPC 外部解析为公有 IP 地址。对数据库集群的访问最终由它使用的安全组控制。如果分配给数据库集群的安全组不包含允许公有访问的入站规则，则不允许该公有访问。此外，要使数据库集群可公开访问，其数据库子网组中的子网必须具有互联网网关。有关更多信息，请参阅无法连接到 Amazon RDS 数据库实例

您可以通过修改 Public access（公开访问）选项，修改数据库集群来开启或关闭公开可访问性。下图显示了其他连接配置部分中的公开访问选项。要设置此选项，请打开连接部分中的其他连接配置部分。



有关修改数据库实例以设置公开访问选项的信息，请参阅修改数据库集群中的数据库实例。

在 VPC 中创建数据库集群

以下过程帮助您在 VPC 中创建数据库集群。要使用默认 VPC，可以从步骤 2 开始，并使用已经为您创建的 VPC 和数据库子网组。如果您想创建其他 VPC，则可创建新的 VPC。

> **Note**
>
> 如果要让 VPC 中的数据库集群实现公开访问，则必须通过启用 VPC 属性 DNS hostnames（DNS 主机名）和 DNS resolution（DNS 解析）更新 VPC 的 DNS 信息。有关更新 VPC 实例的 DNS 信息的信息，请参阅更新对 VPC 的 DNS 支持。

执行以下步骤可在 VPC 中创建数据库实例：

- 步骤 1：创建 VPC
- 步骤 2：创建数据库子网组
- 步骤 3：创建 VPC 安全组
- 步骤 4：在 VPC 中创建数据库实例

步骤 1：创建 VPC

创建一个 VPC，该 VPC 具有的子网位于至少两个可用区内。您在创建数据库子网组时将使用这些子网。如果您拥有默认 VPC，则系统会在该 AWS 区域中的每个可用区中自动为您创建子网。

有关更多信息，请参阅创建包含公有子网和私有子网的 VPC，或参阅《Amazon VPC 用户指南》中的创建 VPC。

步骤 2：创建数据库子网组

数据库子网组是您为 VPC 创建、然后为数据库集群指定的子网（通常为私有子网）的集合。数据库子网组可让您在使用 AWS CLI 或 RDS API 创建数据库集群时指定特定的 VPC。如果您使用控制台，您就只能选择要使用的 VPC 和子网。每个数据库子网组必须至少包含给定 AWS 区域中至少两个可用区的一个子网。作为最佳实践，每个数据库子网组应至少包含 AWS 区域中每个可用区的一个子网。

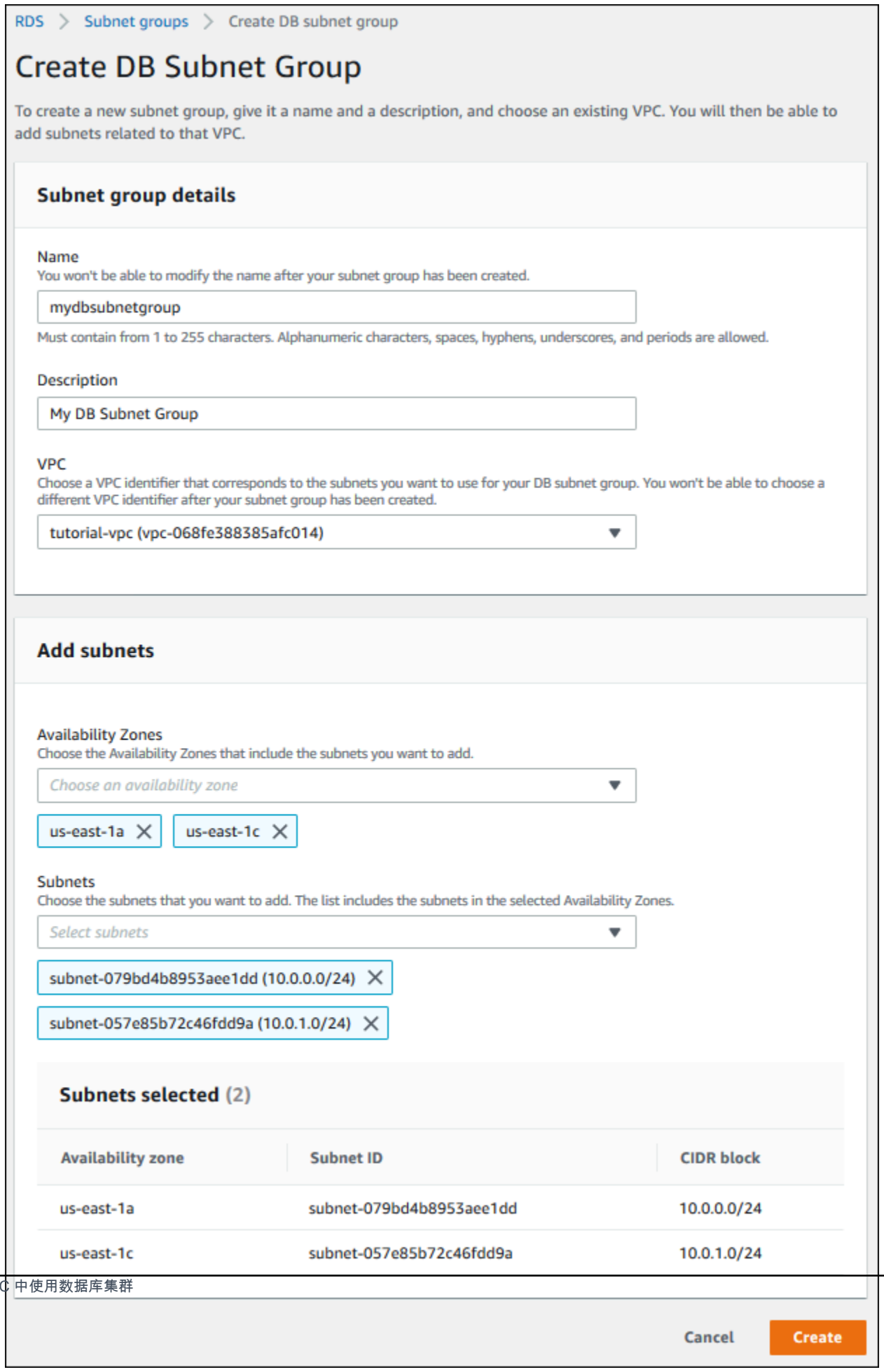
为使数据库集群可公开访问，数据库子网组中的子网必须具有互联网网关。有关子网的互联网网关的更多信息，请参阅《Amazon VPC 用户指南》中的使用互联网网关将子网连接到互联网。

在 VPC 中创建数据库集群时，您可以选择数据库子网组。Amazon Aurora 会选择要与数据库集群关联的子网和该子网中的 IP 地址。如果不存在数据库子网组，Amazon Aurora 会在您创建数据库集群时创建默认子网组。Amazon Aurora 使用该 IP 地址创建弹性网络接口并将其关联到您的数据库集群。数据库集群使用包含该子网的可用区。

在此步骤中，您创建一个数据库子网组，然后添加为 VPC 创建的子网。

创建数据库子网组

1. 通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择子网组。
3. 选择创建数据库子网组。
4. 对于名称，键入您的数据库子网组的名称。
5. 对于描述，键入您的数据库子网组的描述。
6. 对于 VPC，选择默认 VPC 或您创建的 VPC。
7. 在添加子网部分中，从可用区中选择包含子网的可用区，然后从子网中选择子网。


RDS > Subnet groups > Create DB subnet group
Create DB Subnet Group
To create a new subnet group, give it a name and a description, and choose an existing VPC. You will then be able to add subnets related to that VPC.
Subnet group details
Name
You won't be able to modify the name after your subnet group has been created.
mydbsubnetgroup
Must contain from 1 to 255 characters. Alphanumeric characters, spaces, hyphens, underscores, and periods are allowed.
Description
My DB Subnet Group
VPC
Choose a VPC identifier that corresponds to the subnets you want to use for your DB subnet group. You won't be able to choose a different VPC identifier after your subnet group has been created.
tutorial-vpc (vpc-068fe388385afc014)
Add subnets
Availability Zones
Choose the Availability Zones that include the subnets you want to add.
Choose an availability zone
us-east-1a
us-east-1c
Subnets
Choose the subnets that you want to add. The list includes the subnets in the selected Availability Zones.
Select subnets
subnet-079bd4b8953aee1dd (10.0.0.0/24)
subnet-057e85b72c46fdd9a (10.0.1.0/24)
Subnets selected (2)
Availability zone
Subnet ID
CIDR block
us-east-1a
subnet-079bd4b8953aee1dd
10.0.0.0/24
us-east-1c
subnet-057e85b72c46fdd9a
10.0.1.0/24
Cancel
Create

8. 选择创建。

   您的新数据库子网组显示在 RDS 控制台的数据库子网组列表中。可选择该数据库子网组，在窗口底部的详细信息窗格中查看详细信息，其中包括与该组关联的所有子网。

步骤 3：创建 VPC 安全组

创建数据库集群前，您可以创建要与数据库集群相关联的 VPC 安全组。如果您不创建 VPC 安全组，则可以在创建数据库集群时使用默认安全组。有关如何为数据库集群创建安全组的说明，请参阅为私有数据库集群创建 VPC 安全组，或参阅《Amazon VPC 用户指南》中的使用安全组控制到资源的流量。

步骤 4：在 VPC 中创建数据库实例

在此步骤中，创建一个数据库集群，并使用在之前的步骤中创建的 VPC 名称、数据库子网组和 VPC 安全组。

> **Note**
>
> 如果要让 VPC 中的数据库集群实现公开访问，必须启用 VPC 属性 DNS hostnames（DNS 主机名）和 DNS resolution（DNS 解析）。有关更多信息，请参阅《Amazon VPC 用户指南》中的 VPC 的 DNS 属性。

有关如何创建数据库集群的详细信息，请参阅创建 Amazon Aurora 数据库集群。

在 Connectivity（连接）部分出现提示时，输入 VPC 名称、数据库子网组和 VPC 安全组。

> **Note**
>
> Aurora 数据库集群当前不支持更新 VPC。

在 VPC 中访问数据库集群的场景

Amazon Aurora 支持以下在 VPC 中访问数据库集群的场景：

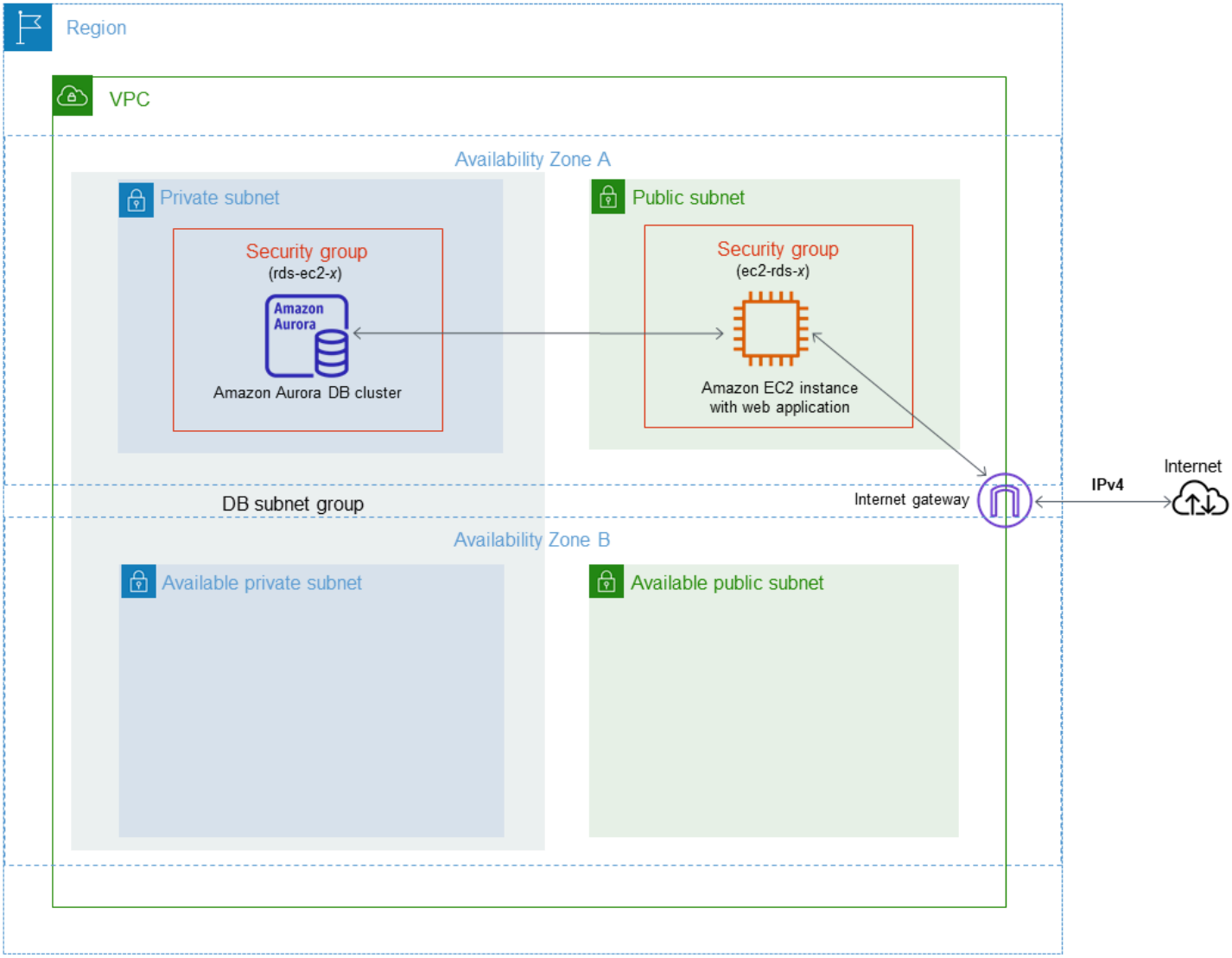
- 同一 VPC 中的 Amazon EC2 实例
- 不同 VPC 中的 EC2 实例
- 通过 Internet 访问的客户端应用程序

- 私有网络

VPC 中的数据库集群由同一 VPC 中的 Amazon EC2 实例访问

VPC 中的数据库集群的常见用途是与在相同 VPC 的 Amazon EC2 实例中运行的应用程序服务器共享数据。

下图说明了此情形。



管理相同 VPC 中 EC2 实例与数据库集群之间的访问的最简单方法是执行以下操作：

- 创建数据库集群将位于其中的 VPC 安全组。此安全组可用于限制对数据库集群的访问权限。例如，您可以为该安全组创建自定义规则。该规则可能允许使用您创建数据库集群时分配给它的端口以及您用来访问数据库集群的 IP 地址（用于开发或其他目的）进行 TCP 访问。

- 创建您的 EC2 实例（Web 服务器和客户端）将位于其中的 VPC 安全组。如果需要，此安全组可允许使用 VPC 的路由表从 Internet 访问 EC2 实例。例如，您可设置此安全组的规则以允许通过端口 22 对 EC2 实例进行 TCP 访问。
- 为数据库集群的安全组创建自定义规则，这些规则允许从您为 EC2 实例创建的安全组进行连接。这些规则将允许安全组的任何成员访问数据库集群。

在单独的可用区中还有一个额外的公有和私有子网。RDS 数据库子网组需要位于至少两个可用区中的一个子网。额外的子网使将来很容易切换到多可用区数据库实例部署。

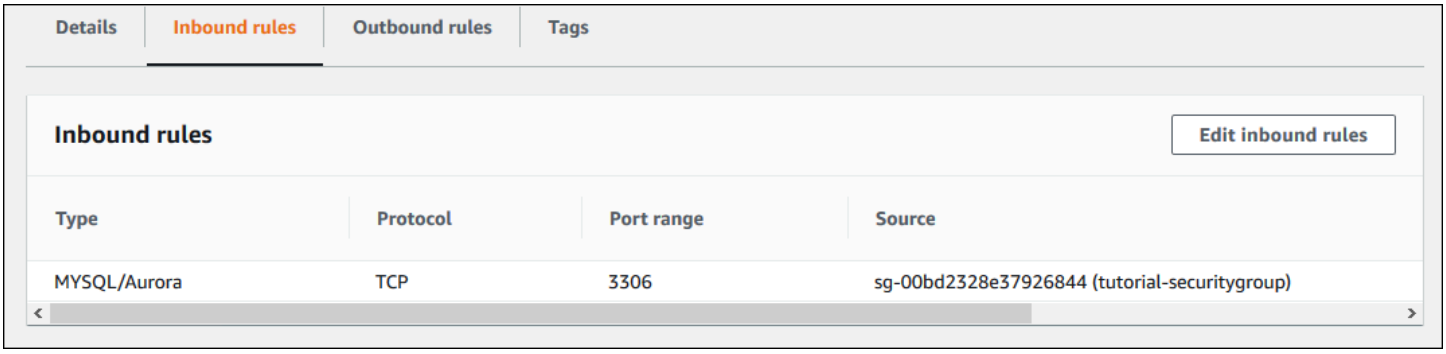
有关说明如何为此方案创建包含公有子网和私有子网的 VPC 的教程，请参阅教程：创建 VPC 以用于数据库集群（仅限 IPv4）。

> **Tip**
>
> 创建数据库集群时，您可以在 Amazon EC2 实例和数据库集群之间自动设置网络连接。有关更多信息，请参阅 配置与 EC2 实例的自动网络连接 。

要在 VPC 安全组中创建允许从另一安全组连接的规则，请执行以下操作：

1. 登录到AWS Management Console并打开 Amazon VPC 控制台，网址：https://console.aws.amazon.com/vpc。
2. 在导航窗格中，选择 Security groups（安全组）。
3. 选择或创建要允许另一个安全组的成员访问的安全组。在前面的场景中，这是您用于数据库集群的安全组。选择 Inbound rules（入站规则）选项卡，然后选择 Edit inbound rules（编辑入站规则）。
4. 在 Edit inbound rules（编辑入站规则）页面上，选择 Add rule（添加规则）。
5. 对于类型，选择与创建数据库集群时使用的端口相对应的条目，如 MYSQL/Aurora。
6. 在源框中，开始键入安全组的 ID，其中列出了匹配的安全组。选择您希望其成员可以访问此安全组保护的资源的安全组。在前面的方案中，这是您用于 EC2 实例的安全组。
7. 如果需要，可通过在源框中创建类型为所有 TCP 的规则以及安全组，对 TCP 协议重复这些步骤。如果您打算使用 UDP 协议，请在 Source（源）框中创建 Type（类型）为 All UDP（所有 UDP）的规则以及安全组。
8. 选择保存规则。

以下屏幕显示了包含其来源的安全组的入站规则。

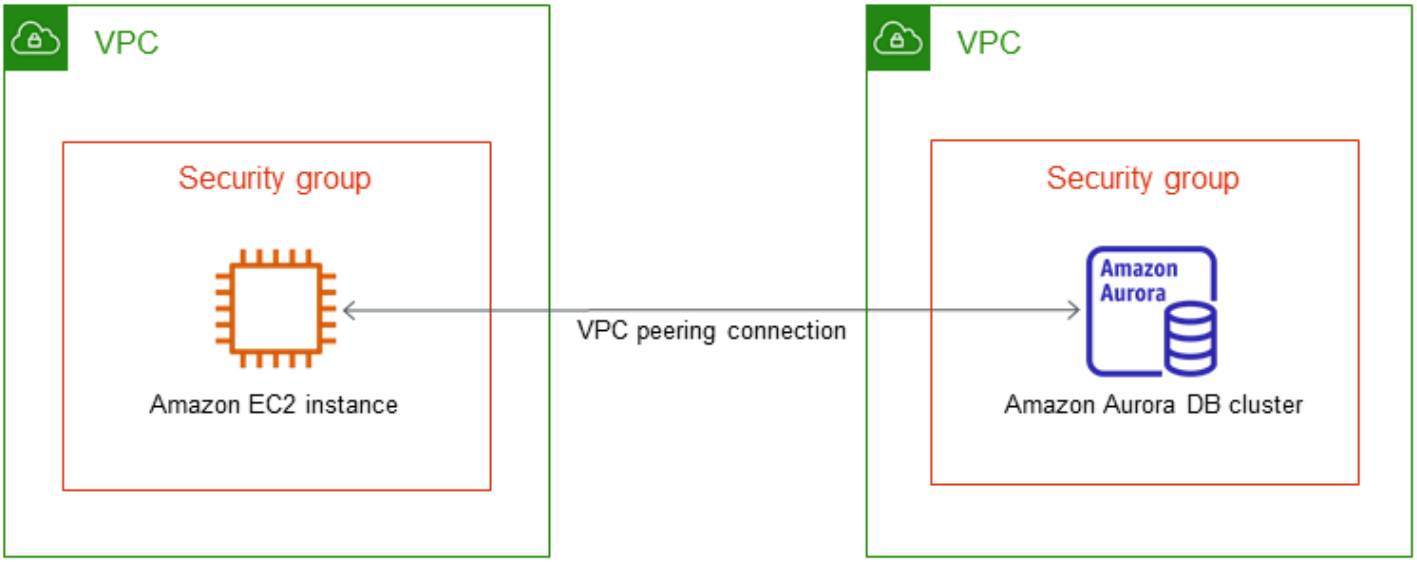


有关从 EC2 实例连接到数据库集群的更多信息，请参阅连接到 Amazon Aurora 数据库集群。

VPC 中的数据库集群由另一 VPC 中的 EC2 实例访问

当您的数据库集群与您用来访问它的 EC2 实例位于不同的 VPC 中时，可使用 VPC 对等连接来访问数据库集群。

下图说明了此情形。

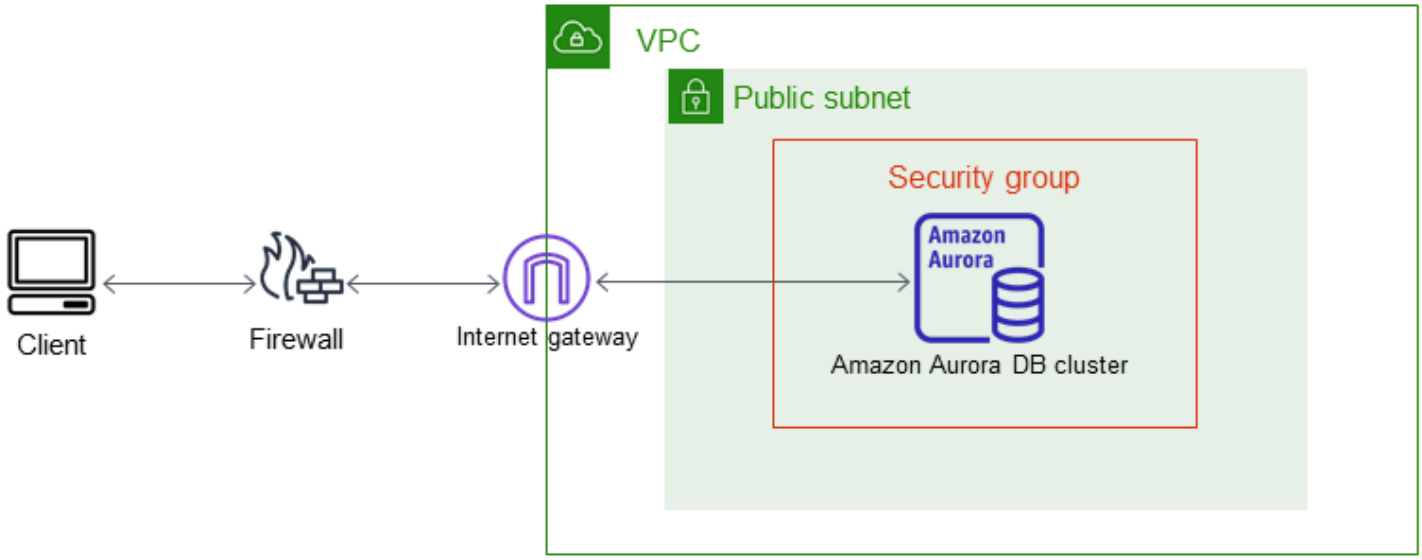


VPC 对等连接是两个 VPC 之间的网络连接，通过此连接，您可以使用私有 IP 地址在这两个 VPC 之间路由流量。这两个 VPC 中的资源可以彼此通信，就像它们在同一网络中一样。您可以在自己的 VPC 之间、自己的 VPC 与另一个 AWS 账户中的 VPC 或与其他 AWS 区域中的 VPC 之间创建 VPC 对等连接。要了解有关 VPC 对等的更多信息，请参阅 Amazon Virtual Private Cloud 用户指南 中的 VPC 对等。

VPC 中的数据库集群由客户端应用程序通过互联网访问

要从客户端应用程序通过互联网访问 VPC 中的数据库集群，您可配置包含单个公有子网的 VPC 以及一个互联网网关，以实现通过互联网通信。

下图说明了此情形。



推荐以下配置：

- 大小为 /16 的 VPC (例如，CIDR：10.0.0.0/16)。此大小提供了 65536 个私有 IP 地址。
- 大小为 /24 的子网 (例如，CIDR：10.0.0.0/24)。此大小提供了 256 个私有 IP 地址。
- 与 VPC 和子网关联的 Amazon Aurora 数据库集群。Amazon RDS 将子网内的 IP 地址分配给您的数据库集群。
- 将 VPC 连接到 Internet 和其他 AWS 产品的互联网网关。
- 与数据库集群关联的安全组。安全组的入站规则允许客户端应用程序访问数据库集群。

有关在 VPC 中创建数据库集群的信息，请参阅在 VPC 中创建数据库集群。

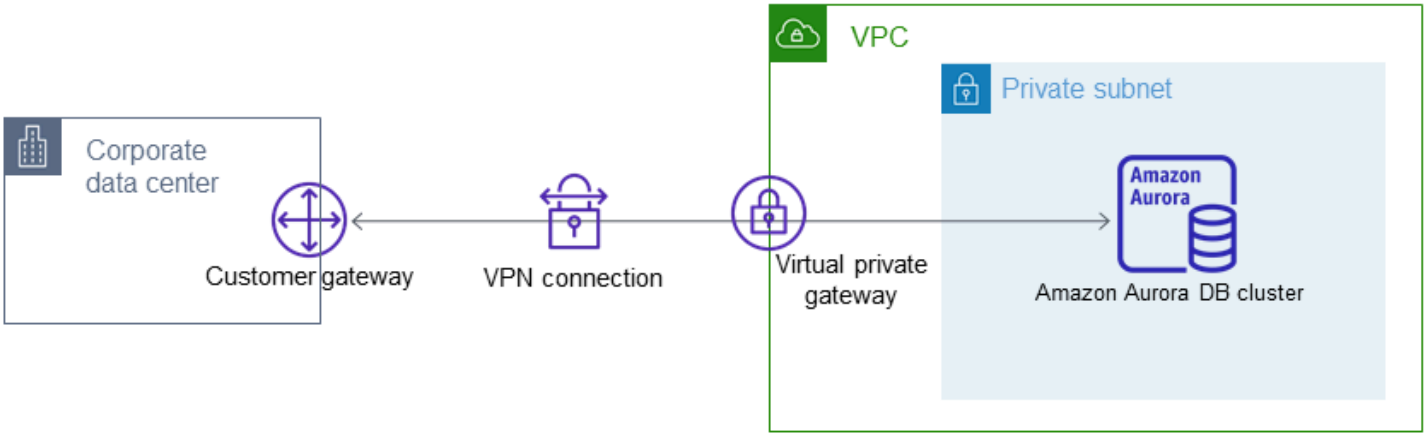
VPC 中由私有网络访问的数据库集群

如果您的数据库集群不可公开访问，可通过以下选项从私有网络进行访问：

- 一个AWS Site-to-Site VPN 连接。有关更多信息，请参阅什么是 AWS Site-to-Site VPN？
- AWS Direct Connect 连接。有关更多信息，请参阅什么是 AWS Direct Connect？

- AWS Client VPN 连接。有关更多信息，请参阅什么是 AWS Client VPN？

下图显示了一个具有AWS Site-to-Site VPN 连接的场景。

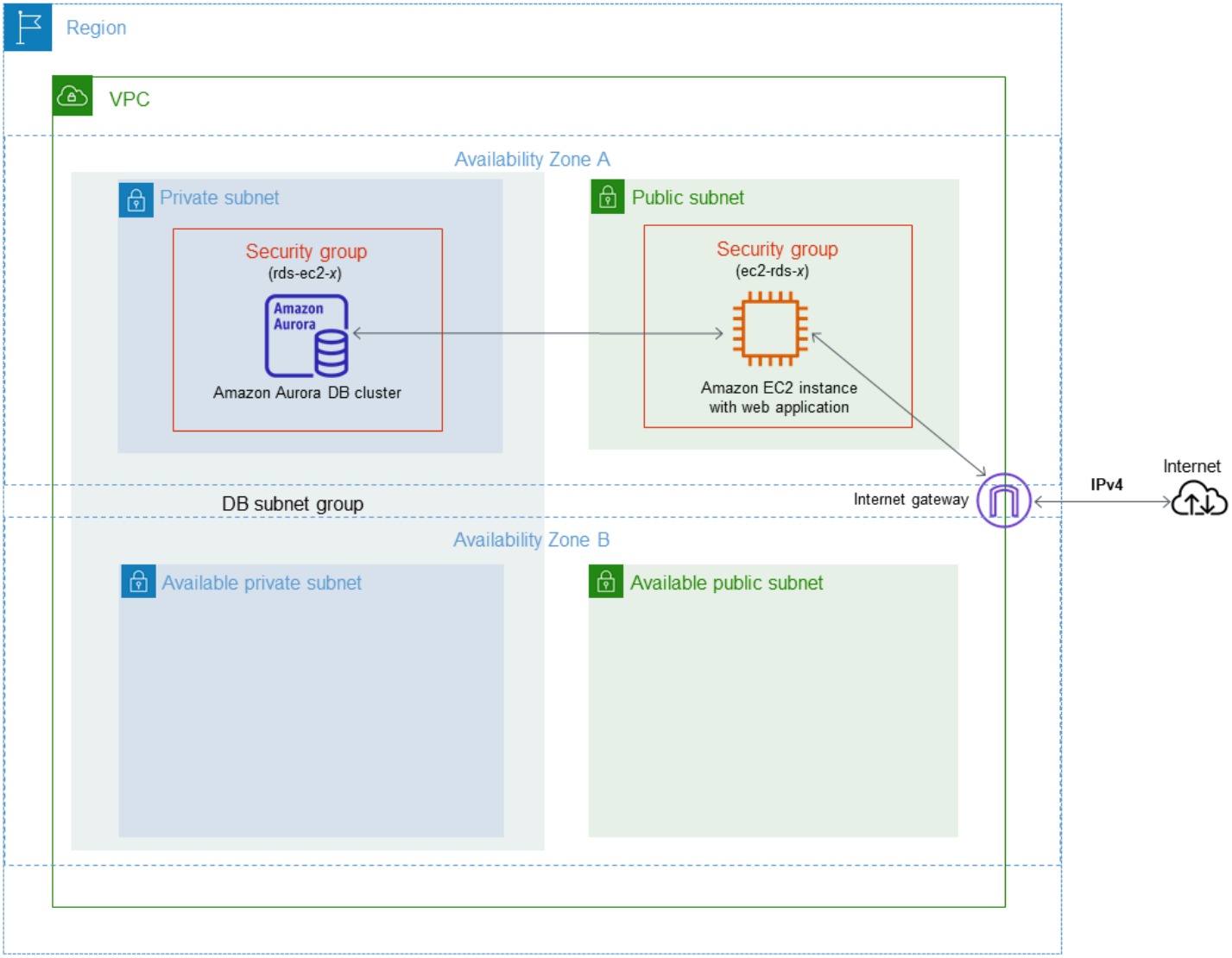


有关更多信息，请参阅 互联网络流量隐私。

教程：创建 VPC 以用于数据库集群（仅限 IPv4）

一种常见的场景包括虚拟私有云（VPC）中基于 Amazon VPC 服务的数据库集群。此 VPC 与在同一 VPC 中运行的 Web 服务器共享数据。在本教程中，针对此场景创建 VPC。

下图说明了此情形。有关其他方案的信息，请参阅在 VPC 中访问数据库集群的场景。



数据库集群只需对 Web 服务器可用，而无需对公共互联网可用。因此，请创建包含公有子网和私有子网的 VPC。Web 服务器托管在公有子网中，以便它可访问公共互联网。数据库集群托管于私有子网中。Web 服务器可以连接到数据库集群，因为它托管在同一 VPC 内。但是，数据库集群不可用于公共互联网，从而提高了安全性。

本教程在单独的可用区中配置额外的公有和私有子网。本教程未使用这些子网。RDS 数据库子网组需要位于至少两个可用区中的一个子网。借助额外的子网，可以更轻松地配置多个 Aurora 数据库实例。

本教程介绍为 Amazon Aurora 数据库集群配置 VPC。有关向您说明如何为此 VPC 方案创建 Web 服务器的教程，请参阅教程：创建 Web 服务器和 Amazon Aurora 数据库集群。有关 Amazon VPC 的更多信息，请参阅 Amazon VPC 入门指南和 Amazon VPC 用户指南。

> **Tip**
>
> 创建数据库集群时，您可以在 Amazon EC2 实例和数据库集群之间自动设置网络连接。网络配置类似于本教程中描述的配置。有关更多信息，请参阅配置与 EC2 实例的自动网络连接。

创建包含公有子网和私有子网的 VPC

使用以下步骤创建包含公有和私有子网的 VPC。

创建 VPC 和子网

1. 通过 https://console.aws.amazon.com/vpc/ 打开 Amazon VPC 控制台。
2. 在 AWS Management Console 的右上角，选择要在其中创建 VPC 的区域。此示例使用 美国西部（俄勒冈）区域。
3. 在左上角，选择 VPC Dashboard（VPC 控制面板）。要开始创建 VPC，请选择 Create VPC（创建 VPC）。
4. 对于 VPC Settings（VPC 设置）下的 Resources to create（要创建的资源），选择 VPC and more（VPC 及更多）。
5. 对于 VPC settings（VPC 设置），请设置以下值：
   - Name tag auto-generation（名称标签自动生成）– **tutorial**
   - IPv4 CIDR block（IPv4 CIDR 块）– **10.0.0.0/16**
   - IPv6 CIDR block（IPv6 CIDR 块）– No IPv6 CIDR block（无 IPv6 CIDR 块）
   - Tenancy（租赁）– Default（原定设置）
   - Number of Availability Zones (AZs) [可用区（AZ）数量] – 2
   - Customize AZs（自定义可用区）– 保留原定设置值。
   - Number of public subnet（公有子网的数量）– 2
   - Number of private subnets（私有子网的数量）– 2
   - Customize subnets CIDR blocks（自定义子网 CIDR 块）– 保留原定设置值。
   - NAT gateways ($) [NAT 网关（$）]– None（无）

- VPC endpoints（VPC 端点）– None（无）
- DNS options（DNS 选项）– 保留原定设置值。

6. 选择创建 VPC。

为公共 Web 服务器创建 VPC 安全组

接下来创建安全组以便公共访问。要连接到 VPC 中的公有 EC2 实例，请将入站规则添加到 VPC 安全组。这些规则允许流量从互联网进行连接。

创建 VPC 安全组

1. 通过以下网址打开 Amazon VPC 控制台：https://console.aws.amazon.com/vpc/。
2. 依次选择 VPC 控制面板)、安全组和创建安全组。
3. 在创建安全组页面上，设置以下值：
   - 安全组名称：**tutorial-securitygroup**
   - 说明：**Tutorial Security Group**
   - VPC：选择您之前创建的 VPC，例如：vpc-***identifier*** (tutorial-vpc)
4. 将入站规则添加到安全组。
   a. 确定要用来通过 Secure Shell（SSH）连接到 VPC 中的 EC2 实例的 IP 地址。要确定您的公有 IP 地址，请在新的浏览器窗口或标签页中，使用 https://checkip.amazonaws.com 上的服务。IP 地址的一个示例为 `203.0.113.25/32`。

      在许多情况下，您可能通过互联网服务提供商（ISP）进行连接，或者在不使用静态 IP 地址的情况下从防火墙之后进行连接。如果是这样，请找出客户端计算机使用的 IP 地址范围。

      > **Warning**
      > 如果您使用 `0.0.0.0/0` 进行 SSH 访问，则所有 IP 地址可能能够使用 SSH 访问您的公有实例。在测试环境下短时间内，此方法尚可接受，但它对于生产环境并不安全。在生产环境中，将仅向特定 IP 地址或地址范围授权使用 SSH 访问您的实例。

   b. 在入站规则部分中，选择添加规则。
   c. 为新入站规则设置以下值，以允许 SSH 访问 Amazon EC2 实例。如果这样做，您就可以连接到 Amazon EC2 实例，以便安装 Web 服务器和其他实用程序。您还可以连接到 EC2 实例，以便上载 Web 服务器的内容。

- 类型： **SSH**
- 源：步骤 a 中的 IP 地址或范围，例如：**203.0.113.25/32**。

d. 选择 添加规则。

e. 为新入站规则设置以下值以允许针对 Web 服务器的 HTTP 访问：

- 类型： **HTTP**
- 源：**0.0.0.0/0**

5. 请选择 Create security group（创建安全组）以创建安全组。

请记下安全组 ID，因为本教程的后面将需要它。

为私有数据库集群创建 VPC 安全组

要保持您的数据库集群为私有，请创建第二个安全组进行私有访问。要连接到 VPC 中的私有数据库集群，请将入站规则添加到 VPC 安全组，以仅支持来自 Web 服务器的流量。

创建 VPC 安全组

1. 通过以下网址打开 Amazon VPC 控制台：https://console.aws.amazon.com/vpc/。
2. 依次选择 VPC 控制面板)、安全组和创建安全组。
3. 在创建安全组页面上，设置以下值：

- 安全组名称：**tutorial-db-securitygroup**
- 说明：**Tutorial DB Instance Security Group**
- VPC：选择您之前创建的 VPC，例如：vpc-***identifier*** (tutorial-vpc)

4. 将入站规则添加到安全组。

a. 在入站规则部分中，选择添加规则。

b. 为新入站规则设置以下值，以允许 Amazon EC2 实例中端口 3306 上的 MySQL 流量。如果这样做，您就可以从 Web 服务器连接到数据库集群。这样，您就可以从 Web 应用程序将数据存储和检索到数据库。

- 类型：**MySQL/Aurora**
- Source（源）：您在本教程的前面部分创建的 tutorial-securitygroup 安全组的标识符，例如 sg-9edd5cfb。

5. 请选择 Create security group（创建安全组）以创建安全组。

创建数据库子网组

数据库子网组 是您在 VPC 中创建并随后指定给数据库集群的子网集合。通过数据库子网组，您可能能够在创建数据库集群时指定特定的 VPC。

创建数据库子网组

1. 在 VPC 中识别数据库的私有子网。

   a. 通过 https://console.aws.amazon.com/vpc/ 打开 Amazon VPC 控制台。

   b. 选择 VPC Dashboard（VPC 控制面板），然后选择 Subnets（子网）。

   c. 记下名为 tutorial-subnet-private1-us-west-2a 和 tutorial-subnet-private2-us-west-2b 的子网的子网 ID。

      创建数据库子网组时需要子网 ID。

2. 通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。

   确保您连接到 Amazon RDS 控制台，而不是 Amazon VPC 控制台。

3. 在导航窗格中，选择子网组。

4. 选择 Create DB subnet group（创建数据库子网组）。

5. 在创建数据库子网组页面的子网组详细信息中设置以下值：

   - 名称：**tutorial-db-subnet-group**
   - 说明：**Tutorial DB Subnet Group**
   - VPC：tutorial-vpc (vpc-***identifier***)

6. 在添加子网部分中，选择可用区和子网。

   对于本教程，请为 Availability Zones（可用区）选择 us-west-2a 和 us-west-2b。对于 Subnets（子网），选择您在上一步中确定的私有子网。

7. 选择创建。

   您的新数据库子网组显示在 RDS 控制台的数据库子网组列表中。可选择该数据库子网组，以在窗口底部的详细信息窗格中查看详细信息。这些详细信息包括与该组关联的所有子网。

> **Note**
>
> 如果您已创建此 VPC 以完成 教程：创建 Web 服务器和 Amazon Aurora 数据库集群，请按照 创建 Amazon Aurora 数据库集群 中的说明创建数据库集群。

删除 VPC

为本教程创建 VPC 和其他资源后，如果不再需要 VPC 和其他资源，可以将其删除。

> **Note**
>
> 如果您在为本教程创建的 VPC 中添加了资源，则可能需要先删除这些资源，然后才能删除 VPC。例如，这些资源可能包括 Amazon EC2 实例或 Amazon RDS 数据库集群。有关更多信息，请参阅 Amazon VPC 用户指南中的您的 VPC 的安全性。

删除 VPC 和相关资源

1. 删除数据库子网组。
   a. 通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
   b. 在导航窗格中，选择子网组。
   c. 选择要删除的数据库子网组，例如 tutorial-db-subnet-group。
   d. 选择 Delete (删除)，然后在确认窗口中选择 Delete (删除)。
2. 记下 VPC ID。
   a. 通过以下网址打开 Amazon VPC 控制台：https://console.aws.amazon.com/vpc/。
   b. 选择 VPC 控制面板，然后选择 VPC。
   c. 在列表中，确定您创建的 VPC，例如 tutorial-vpc。
   d. 记下所创建 VPC 的 VPC ID。在后面的步骤中，您需要此 VPC ID。
3. 删除安全组。
   a. 通过以下网址打开 Amazon VPC 控制台：https://console.aws.amazon.com/vpc/。
   b. 选择 VPC 控制面板，然后选择安全组。
   c. 选择 Amazon RDS 数据库实例的安全组，例如 tutorial-db-securitygroup。

d. 对于 Actions（操作），请选择 Delete security groups（删除安全组），然后在确认页面上选择 Delete（删除）。

e. 在安全组页面上，选择 Amazon EC2 实例的安全组，例如 tutorial-securitygroup。

f. 对于 Actions（操作），请选择 Delete security groups（删除安全组），然后在确认页面上选择 Delete（删除）。

4. 删除 VPC

a. 通过以下网址打开 Amazon VPC 控制台：https://console.aws.amazon.com/vpc/。

b. 选择 VPC 控制面板，然后选择 VPC。

c. 选择要删除的 VPC，例如 tutorial-vpc。

d. 对于 Actions (操作)，请选择 Delete VPC (删除 VPC)。

确认页面显示与 VPC 关联的其他资源，这些资源也将被删除，包括与其关联的子网。

e. 在确认页面上，输入 **`delete`** 并选择 Delete (删除)。

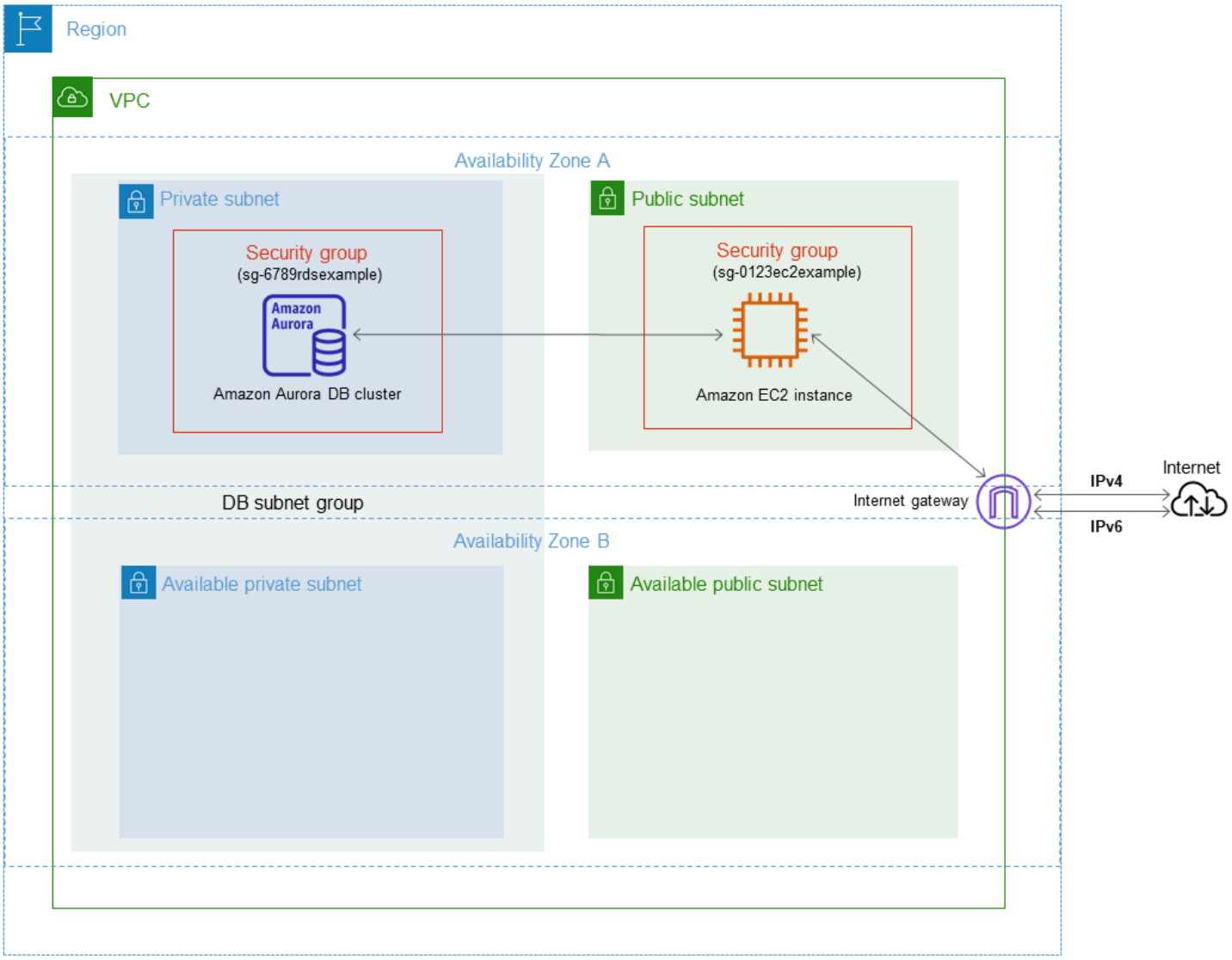
教程：创建 VPC 以用于数据库集群（双堆栈模式）

一种常见的场景包括虚拟私有云（VPC）中基于 Amazon VPC 服务的数据库集群。此 VPC 与在同一 VPC 中运行的公有 Amazon EC2 实例共享数据。

在本教程中，您将为此场景创建 VPC，该 VPC 与在双堆栈模式下运行的数据库一起使用。双堆栈模式，用于启用基于 IPv6 寻址协议的连接。有关 IP 地址的更多信息，请参阅 Amazon Aurora IP 寻址。

大多数区域支持双堆栈网络集群。有关更多信息，请参阅双堆栈网络数据库集群的可用性。要查看双堆栈模式的限制，请参阅 双堆栈网络数据库集群的限制。

下图说明了此情形。



有关其他方案的信息，请参阅在 VPC 中访问数据库集群的场景。

数据库集群只需对 Amazon EC2 实例可用，而无需对公共互联网可用。因此，请创建包含公有子网和私有子网的 VPC。Amazon EC2 实例托管在公有子网中，以便它可访问公共 Internet。数据库集群托管于私有子网中。Amazon EC2 实例可以连接到数据库集群，因为它托管在同一 VPC 内。但是，数据库集群不可用于公共互联网，从而提高了安全性。

本教程在单独的可用区中配置额外的公有和私有子网。本教程未使用这些子网。RDS 数据库子网组需要位于至少两个可用区中的一个子网。借助额外的子网，可以轻松地配置多个 Aurora 数据库实例。

要创建使用双堆栈模式的数据库集群，请为 Network type（网络类型）设置指定 Dual-stack mode（双堆栈模式）。您也可以使用相同的设置修改数据库集群。有关创建数据库集群的更多信息，请参阅创建 Amazon Aurora 数据库集群。有关修改数据库集群的更多信息，请参阅 修改 Amazon Aurora 数据库集群。

本教程介绍为 Amazon Aurora 数据库集群配置 VPC。有关 Amazon VPC 的更多信息，请参阅 Amazon VPC 用户指南。

创建包含公有子网和私有子网的 VPC

使用以下步骤创建包含公有和私有子网的 VPC。

创建 VPC 和子网

1. 通过以下网址打开 Amazon VPC 控制台：https://console.aws.amazon.com/vpc/。
2. 在 AWS Management Console 的右上角，选择要在其中创建 VPC 的区域。此示例使用美国东部（俄亥俄州）区域。
3. 在左上角，选择 VPC Dashboard（VPC 控制面板）。要开始创建 VPC，请选择 Create VPC（创建 VPC）。
4. 对于 VPC Settings（VPC 设置）下的 Resources to create（要创建的资源），选择 VPC and more（VPC 及更多）。
5. 对于剩下的 VPC settings（VPC 设置），请设置这些值：
   - Name tag auto-generation（名称标签自动生成）– **`tutorial-dual-stack`**
   - IPv4 CIDR block（IPv4 CIDR 块）– **`10.0.0.0/16`**
   - IPv6 CIDR block（IPv6 CIDR 块）– Amazon-provided IPv6 CIDR block（Amazon 提供的 IPv6 CIDR 块）
   - Tenancy（租赁）– Default（原定设置）
   - Number of Availability Zones (AZs) [可用区（AZ）数量] – 2

- Customize AZs（自定义可用区）– 保留原定设置值。
- Number of public subnet（公有子网的数量）– 2
- Number of private subnets（私有子网的数量）– 2
- Customize subnets CIDR blocks（自定义子网 CIDR 块）– 保留原定设置值。
- NAT gateways ($) [NAT 网关（$）]– None（无）
- Egress only internet gateway（仅限出口的互联网网关）– No（否）
- VPC endpoints（VPC 端点）– None（无）
- DNS options（DNS 选项）– 保留原定设置值。

> **Note**
>
> Amazon RDS 至少需要位于两个不同可用区中的两个子网，才能支持多可用区数据库实例部署。本教程创建了单可用区部署，但这项要求使得将来可以轻松转换为多可用区数据库实例部署。

6. 选择创建 VPC。

为公有 Amazon EC2 实例创建 VPC 安全组

接下来创建安全组以便公共访问。要连接到 VPC 中的公有 EC2 实例，请将入站规则添加到 VPC 安全组，以允许流量从互联网连接。

创建 VPC 安全组

1. 通过以下网址打开 Amazon VPC 控制台：https://console.aws.amazon.com/vpc/。
2. 依次选择 VPC 控制面板)、安全组和创建安全组。
3. 在创建安全组页面上，设置以下值：
   - 安全组名称：**`tutorial-dual-stack-securitygroup`**
   - 说明：**`Tutorial Dual-Stack Security Group`**
   - VPC：选择您之前创建的 VPC，例如：vpc-***`identifier`*** (tutorial-dual-stack-vpc)
4. 将入站规则添加到安全组。
   a. 确定要用来通过 Secure Shell（SSH）连接到 VPC 中的 EC2 实例的 IP 地址。

互联网协议版本 4（IPv4）地址的示例为 `203.0.113.25/32`。互联网协议版本 6（IPv6）地址范围的示例为 `2001:db8:1234:1a00::/64`。

在许多情况下，您可能通过互联网服务提供商（ISP）进行连接，或者在不使用静态 IP 地址的情况下从防火墙之后进行连接。如果是这样，请找出客户端计算机使用的 IP 地址范围。

> **Warning**
>
> 如果对 IPv4 使用 `0.0.0.0/0` 或对 IPv6 使用 `::0`，则使所有 IP 地址可能能够使用 SSH 访问您的公有实例。在测试环境下短时间内，此方法尚可接受，但它对于生产环境并不安全。在生产环境中，请仅授权特定 IP 地址或地址范围访问您的实例。

b. 在入站规则部分中，选择添加规则。

c. 为新入站规则设置以下值，以允许安全外壳 (SSH) 访问 Amazon EC2 实例。如果这样做，则可以连接到 EC2 实例来安装 SQL 客户端和其他应用程序。指定一个 IP 地址，以便您可以访问 EC2 实例：

- 类型：**SSH**
- Source（源）：步骤 a 中的 IP 地址或范围。IPv4 IP 地址的一个示例为 **203.0.113.25/32**。IPv6 IP 地址的一个示例为 **2001:DB8::/32**。

5. 请选择 Create security group（创建安全组）以创建安全组。

请记下安全组 ID，因为本教程的后面将需要它。

为私有数据库集群创建 VPC 安全组

要保持您的数据库集群为私有，请创建第二个安全组进行私有访问。要连接到 VPC 中的私有数据库集群，请将入站规则添加到 VPC 安全组。它们仅允许来自 Amazon EC2 实例的流量。

创建 VPC 安全组

1. 通过以下网址打开 Amazon VPC 控制台：https://console.aws.amazon.com/vpc/。
2. 依次选择 VPC 控制面板)、安全组和创建安全组。
3. 在创建安全组页面上，设置以下值：

- 安全组名称：**tutorial-dual-stack-db-securitygroup**
- 说明：**Tutorial Dual-Stack DB Instance Security Group**

- VPC：选择您之前创建的 VPC，例如：vpc-***identifier*** (tutorial-dual-stack-vpc)

4. 将入站规则添加到安全组：

   a. 在入站规则部分中，选择添加规则。

   b. 为新入站规则设置以下值，以允许 Amazon EC2 实例中端口 3306 上的 MySQL 流量。如果这样做，您就可以从 EC2 实例连接到您的数据库集群。这样做意味着您可以将数据从 EC2 实例发送到数据库。

      - Type（类型）：MySQL/Aurora
      - Source（源）：您在本教程的前面部分创建的 tutorial-dual-stack-securitygroup 安全组的标识符，例如 sg-9edd5cfb。

5. 要创建安全组，请选择创建安全组。

创建数据库子网组

数据库子网组 是您在 VPC 中创建并随后指定给数据库集群的子网集合。使用数据库子网组，您可以在创建数据库集群时指定特定的 VPC。要创建与 DUAL 兼容的数据库子网组，所有子网都必须与 DUAL 兼容。要与 DUAL 兼容，子网必须具有与之关联的 IPv6 CIDR。

创建数据库子网组

1. 在 VPC 中识别数据库的私有子网。

   a. 通过以下网址打开 Amazon VPC 控制台：https://console.aws.amazon.com/vpc/。

   b. 选择 VPC Dashboard（VPC 控制面板），然后选择 Subnets（子网）。

   c. 记下名为 tutorial-dual-stack-subnet-private1-us-west-2a 和 tutorial-dual-stack-subnet-private2-us-west-2b 的子网的子网 ID。

      创建数据库子网组时需要子网 ID。

2. 通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。

   确保您连接到 Amazon RDS 控制台，而不是 Amazon VPC 控制台。

3. 在导航窗格中，选择子网组。

4. 选择 Create DB subnet group（创建数据库子网组）。

5. 在创建数据库子网组页面的子网组详细信息中设置以下值：

   - 名称：**tutorial-dual-stack-db-subnet-group**

- 说明：**`Tutorial Dual-Stack DB Subnet Group`**
- VPC：tutorial-dual-stack-vpc (vpc-***`identifier`***)

6. 在 Add subnets（添加子网）部分中，选择 Availability Zones（可用区）和 Subnets（子网）选项的值。

   对于本教程，请为 Availability Zones（可用区）选择 us-east-2a 和 us-east-2b。对于 Subnets（子网），选择您在上一步中确定的私有子网。

7. 选择创建。

您的新数据库子网组显示在 RDS 控制台的数据库子网组列表中。您可以选择数据库子网组以查看其详细信息。其中包括支持的寻址协议、与该组关联的所有子网以及数据库子网组支持的网络类型。

在双堆栈模式下创建 Amazon EC2 实例

要创建 Amazon EC2 实例，请按照《Amazon EC2 用户指南》中的使用新启动实例向导启动实例中的说明操作。

在 Configure Instance Details（配置实例详细信息）页面上，设置以下值并将其他值保留为其原定设置值：

- 网络 – 选择同时具有公有子网和私有子网的现有 VPC，如在创建包含公有子网和私有子网的 VPC中创建的 tutorial-dual-stack-vpc（vpc-***`identifier`***）。
- Subnet（子网）– 选择一个现有的公有子网，如在为公有 Amazon EC2 实例创建 VPC 安全组中创建的 subnet-***`identifier`*** | tutorial-dual-stack-subnet-public1-us-east-2a | us-east-2a。
- Auto-assign Public IP（自动分配公有 IP）– 选择 Enable（启用）。
- Auto-assign IPv6 IP（自动分配 IPv6 IP）– 选择 Enable（启用）。
- Firewall (security groups) [防火墙（安全组）] – 选择 Select an existing security group（选择现有安全组）。
- Common security groups（常用安全组）– 选择一个现有的安全组，例如在为公有 Amazon EC2 实例创建 VPC 安全组中创建的 `tutorial-securitygroup`。确保您选择的安全组包括 Secure Shell (SSH) 和 HTTP 访问的入站规则。

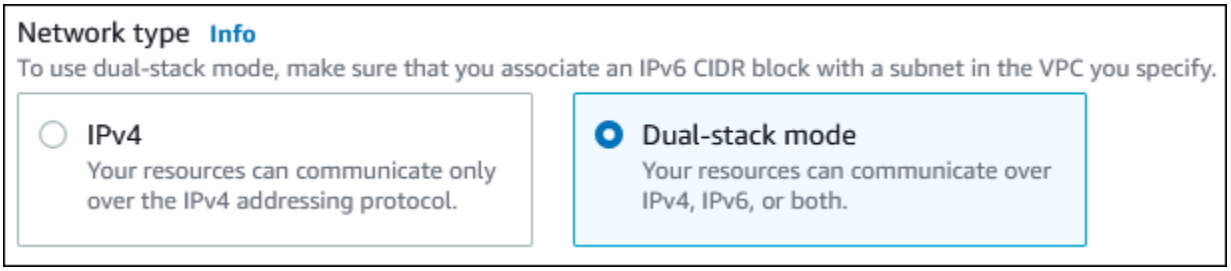
在双堆栈模式下创建数据库集群

在此步骤中，您将创建在双堆栈模式下运行的数据库集群。

创建数据库实例

1. 登录 AWS Management Console 并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在控制台的右上角，选择要在其中创建数据库集群的 AWS 区域。此示例使用美国东部（俄亥俄州）区域。
3. 在导航窗格中，选择 Databases（数据库）。
4. 选择创建数据库。
5. 在 Create database（创建数据库）页面上，确保选择了 Standard Create（标准创建）选项，然后选择 Aurora MySQL 数据库引擎类型。
6. 在 Connectivity（连接）部分中，设置以下值：

   - Network type（网络类型）– 选择 Dual-stack mode（双堆栈模式）。

   - Virtual Private Cloud (VPC) [虚拟私有云（VPC）]– 选择具有公有子网和私有子网的现有 VPC，如在创建包含公有子网和私有子网的 VPC中创建的 tutorial-dual-stack-vpc (vpc-*identifier*)。

     VPC 的子网必须位于不同的可用区中。

   - DB subnet group（数据库子网组）– VPC 的数据库子网组，如在创建数据库子网组中创建的 tutorial-dual-stack-db-subnet-group。
   - Public access（公有访问权限）– 选择 No（否）。
   - VPC security group (firewall) [VPC 安全组（防火墙）] – 选择 Choose existing（选择现有）。
   - Existing VPC security groups（现有 VPC 安全组）– 选择为私有访问配置的现有 VPC 安全组，如 为私有数据库集群创建 VPC 安全组 中创建的 tutorial-dual-stack-db-securitygroup。

     通过选择与其他每个安全组关联的 X 来删除该安全组，如默认安全组。

   - Availability Zone（可用区）– 选择 us-west-2a。

     为避免跨可用区的流量，请确保数据库实例和 EC2 实例位于同一个可用区内。

7. 对于其余部分，请指定数据库集群设置。有关每项设置的信息，请参阅 Aurora 数据库集群的设置。

连接到 Amazon EC2 实例和数据库集群

在双堆栈模式下创建 Amazon EC2 实例和数据库集群后，您可以使用 IPv6 协议连接到每个实例。要使用 IPv6 协议连接到 Amazon EC2 实例，请按照《Amazon EC2 用户指南》中的连接到 Linux 实例中的说明操作。

要从 Aurora MySQL 实例连接到 Aurora MySQL 数据库集群，请按照连接到 Aurora MySQL 数据库集群中的说明进行操作。

删除 VPC

为本教程创建 VPC 和其他资源后，如果不再需要 VPC 和其他资源，可以将其删除。

如果您在为本教程创建的 VPC 中添加了资源，则可能需要先删除这些资源，然后才能删除 VPC。资源示例包括 Amazon EC2 实例或数据库集群。有关更多信息，请参阅 Amazon VPC 用户指南中的您的 VPC 的安全性。

删除 VPC 和相关资源

1. 删除数据库子网组：
    a. 通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
    b. 在导航窗格中，选择子网组。
    c. 选择要删除的数据库子网组，例如 tutorial-db-subnet-group。
    d. 选择 Delete (删除)，然后在确认窗口中选择 Delete (删除)。
2. 记下 VPC ID：
    a. 通过以下网址打开 Amazon VPC 控制台：https://console.aws.amazon.com/vpc/。
    b. 选择 VPC 控制面板，然后选择 VPC。
    c. 在列表中，标识您创建的 VPC，例如 tutorial-dual-stack-vpc。
    d. 记下所创建 VPC 的 VPC ID 值。在后续步骤中，您将需要此 VPC ID。
3. 删除安全组：
    a. 通过以下网址打开 Amazon VPC 控制台：https://console.aws.amazon.com/vpc/。
    b. 选择 VPC 控制面板，然后选择安全组。

c. 选择 Amazon RDS 数据库实例的安全组，例如 tutorial-dual-stack-db-securitygroup。
d. 对于 Actions（操作），请选择 Delete security groups（删除安全组），然后在确认页面上选择 Delete（删除）。
e. 在 Security Groups（安全组）页面上，选择 Amazon EC2 实例的安全组，例如 tutorial-dual-stack-securitygroup。
f. 对于 Actions（操作），请选择 Delete security groups（删除安全组），然后在确认页面上选择 Delete（删除）。

4. 删除 NAT 网关：

a. 通过以下网址打开 Amazon VPC 控制台：https://console.aws.amazon.com/vpc/。
b. 选择 VPC 控制面板，然后选择 NAT 网关。
c. 选择您创建的 VPC 的 NAT 网关。使用 VPC ID 标识正确的 NAT 网关。
d. 对于 Actions（操作），请选择 Delete NAT gateway（删除 NAT 网关）。
e. 在确认页面上，输入 **delete** 并选择删除。

5. 删除 VPC：

a. 通过以下网址打开 Amazon VPC 控制台：https://console.aws.amazon.com/vpc/。
b. 选择 VPC 控制面板，然后选择 VPC。
c. 选择要删除的 VPC，例如 tutorial-dual-stack-vpc。
d. 对于 Actions (操作)，请选择 Delete VPC (删除 VPC)。

   确认页面显示与 VPC 关联的其他资源，这些资源也将被删除，包括与其关联的子网。
e. 在确认页面上，输入 **delete** 并选择 Delete (删除)。

6. 释放弹性 IP 地址：

a. 通过以下网址打开 Amazon EC2 控制台：https://console.aws.amazon.com/ec2/。
b. 选择 EC2 控制面板，然后选择弹性 IP。
c. 选择要释放的弹性 IP 地址。
d. 对于 Actions（操作），请选择 Release Elastic IP addresses（释放弹性 IP 地址）。
e. 在确认页面上，请选择释放。

Amazon Aurora 的配额和限制

接下来，您可以查找 Amazon Aurora 的资源配额和命名约束的说明。

主题

- Amazon Aurora 中的配额
- Amazon Aurora 中的命名约束
- Amazon Aurora 大小限制

Amazon Aurora 中的配额

每个AWS区域的每个AWS账户都有关于可以创建的 Amazon Aurora 资源数量的配额。达到某一资源的配额时，再进行创建该资源的调用就会失败并引发异常。

下表列出了每个 AWS 区域的资源及其配额。

名称	默认值	可调整	描述
每个数据库安全组的授权	每个受支持的区域：20 个	否	每个数据库安全组的安全组授权数
自定义引擎版本	每个受支持的区域：40 个	是	当前区域中此账户中允许的自定义引擎版本的最大数目
数据库集群参数组	每个受支持的区域：50 个	否	数据库集群参数组的最大数目
数据库集群	每个受支持的区域：40 个	是	当前区域中的此账户中允许的 Aurora 集群的最大数目
数据库实例	每个受支持的区域：40 个	是	当前区域中此账户中允许的数据库实例的最大数目

名称	默认值	可调整	描述
数据库子网组	每个受支持的区域：50 个	是	数据库子网组的最大数目
数据 API HTTP 请求体大小	每个受支持的区域：4MB	否	HTTP 请求正文允许的最大大小。
数据 API 最大并发集群密钥对数	每个受支持的区域：30 个	否	当前 AWS 区域中对此账户的并发数据 API 请求中 Aurora Serverless v1 数据库集群和密钥的唯一对的最大数量。
数据 API 最大并发请求数	每个受支持的区域：500 个	否	针对 Aurora Serverless v1 数据库集群的数据 API 请求的最大数量，这些请求使用相同密钥且可以同时处理。当处理中的请求完成时，将对其他请求进行排队和处理。
Data API 最大结果集大小	每个受支持的区域：1MB	否	数据 API 可以返回的数据库结果集的最大大小。
数据 API 的 JSON 响应字符串的最大大小	每个受支持的区域：10 MB	否	RDS 数据 API 返回的简化 JSON 响应字符串的最大大小。
每秒 Data API 请求数	每个受支持的区域：每秒 1000 个	否	当前 AWS 区域中此账户每秒允许的针对数据 API 的最大请求数。此限额仅适用于 Amazon Aurora Serverless v1 集群。

名称	默认值	可调整	描述
事件订阅	每个受支持的区域：20 个	是	事件订阅的最大数目
每个数据库集群的 IAM 角色	每个受支持的区域：5 个	是	与数据库集群关联的 IAM 角色的最大数目
每个数据库实例的 IAM 角色	每个受支持的区域：5 个	是	与数据库实例关联的 IAM 角色的最大数目
集成	每个受支持的区域：100 个	否	当前 AWS 区域内此账户中支持的最大集成数量
手动数据库集群快照	每个受支持的区域：100 个	是	手动数据库集群快照的最大数目
手动数据库实例快照数	每个受支持的区域：100 个	是	手动数据库实例快照的最大数目
选项组	每个受支持的区域：20 个	是	选项组的最大数目
参数组	每个受支持的区域：50 个	是	参数组的最大数目
代理	每个受支持的区域：20 个	是	当前 AWS 区域中此账户中允许的代理的最大数目
每个主数据库实例的只读副本数	每个受支持的区域：15 个	是	每个主数据库实例的只读副本的最大数量。Amazon Aurora 的此限额无法调整。
预留数据库实例	每个受支持的区域：40 个	是	当前 AWS 区域中此账户中允许的预留数据库实例的最大数目

名称	默认值	可调整	描述
每个安全组的规则数	每个受支持的区域：20 个	否	每个数据库安全组的规则的最大数目
安全组	每个受支持的区域：25 个	是	数据库安全组的最大数目
安全组 (VPC)	每个受支持的区域：5 个	否	每个 Amazon VPC 的数据库安全组的最大数目
每个数据库子网组的子网数	每个受支持的区域：20 个	否	每个数据库子网组的子网的最大数目
每个资源的标签	每个受支持的区域：50 个	否	每个 Amazon RDS 资源的标签的最大数目
所有数据库实例的总存储空间	每个受支持的区域：100 GB	是	EBS 卷上所有 Amazon RDS 数据库实例的最大总存储空间（以 GB 为单位）此限额不适用于 Amazon Aurora，每个数据库集群的最大集群容量为 128TiB。

Note

默认情况下，您最多可以有 40 个数据库实例。RDS 数据库实例、Aurora 数据库实例、Amazon Neptune 实例和 Amazon DocumentDB 实例适用于此配额。
如果您的应用程序需要更多数据库实例，则可以通过打开Service Quotas 控制台请求其他数据库实例。在导航窗格中，选择 AWS 服务。选择 Amazon Relational Database Service (Amazon RDS) (Amazon 关系数据库服务 (Amazon RDS))，选择配额，然后按照说明请求增加配额。有关更多信息，请参阅 Service Quotas 用户指南中的请求增加配额。
由 AWS Backup 管理的备份被视为手动数据库集群快照，但不计入手动集群快照限额。有关 AWS Backup 的更多信息，请参阅 AWS Backup 开发人员指南。

如果您使用任何 RDS API 操作并超过每秒调用数的默认限额，Amazon RDS API 会发出如下所示的错误。

ClientError: An error occurred (ThrottlingException) when calling the *API_name* operation: Rate exceeded.

此处，请减少每秒调用数。配额旨在涵盖大多数使用案例。如果需要更高的配额，可以使用以下任一选项请求增加配额：

- 在控制台中，打开服务配额控制台。
- 从 AWS CLI 中，使用 request-service-quota-increase AWS CLI 命令。

有关更多信息，请参阅 Service Quotas 用户指南。

Amazon Aurora 中的命名约束

Amazon Aurora 中的命名约束如下所示：

- 数据库集群标识符：
  - 必须包含 1–63 个字母数字字符或连字符。
  - 第一个字符必须是字母。
  - 不能以连字符结尾，也不能包含两个连续连字符。
  - 对于每个 AWS 区域的每个 AWS 账户的所有数据库实例必须是唯一的。
- 初始数据库名称 – 数据库名称约束在 Aurora MySQL 与 Aurora PostgreSQL 之间不同。有关更多信息，请参阅创建每个数据库集群时的可用设置。
- 主用户名 – 主用户名约束因每个数据库引擎而异。有关更多信息，请参阅创建数据库集群时的可用设置。
- 主密码：
  - 数据库主用户的密码可以包括除 /、'、"、@ 或空格之外的任意可打印 ASCII 字符。
  - 密码可能包含以下数量的可打印 ASCII 字符，具体取决于数据库引擎：
    - Aurora MySQL：8–41
    - Aurora PostgreSQL：8–99
- 数据库参数组：
  - 必须包含 1–255 个字母数字字符。
  - 第一个字符必须是字母。

  - 允许使用连字符，但名称不能以连字符结束或包含两个连续的连字符。
- 数据库子网组：
  - 必须包含 1–255 个字符。
  - 允许使用字母数字字符、空格、连字符、下划线和句点。

Amazon Aurora 大小限制

存储大小限制

对于以下引擎版本，Aurora 集群卷可以增大到 128 tebibytes (TiB) 的最大大小：

- 所有可用的 Aurora MySQL 版本 3 版本；Aurora MySQL 版本 2，版本 2.09 及更高版本
- 所有可用的 Aurora PostgreSQL 版本

对于较低的引擎版本，Aurora 集群卷的最大大小为 64 TiB。有关更多信息，请参阅 Aurora 存储如何自动调整大小。

要监控剩余存储空间，可以使用 `AuroraVolumeBytesLeftTotal` 指标。有关更多信息，请参阅 Amazon Aurora 的集群级指标。

SQL 表大小限制

对于 Aurora MySQL 数据库集群，最大表大小为 64TiB。对于 Aurora PostgreSQL 数据库集群，最大表大小为 32 TiB。我们建议您遵循表设计最佳实践，例如大型表的分区。

表空间 ID 限制

Aurora MySQL 的最大表空间 ID 为 2147483647。如果您经常创建和删除表，请确保知道您的表空间 ID 并计划使用逻辑转储。有关更多信息，请参阅 使用 mysqldump 从 MySQL 逻辑迁移到 Amazon Aurora MySQL。

Amazon Aurora 故障排除

利用以下部分帮助排查您使用 Amazon RDS 和 Amazon Aurora 中的数据库实例时遇到的问题。

主题

- 无法连接到 Amazon RDS 数据库实例
- Amazon RDS 安全性问题
- 重置数据库实例拥有者密码
- Amazon RDS 数据库实例中断或重新引导
- Amazon RDS 数据库参数更改未生效
- Amazon Aurora 中的可用内存问题
- Amazon Aurora MySQL 复制问题

有关使用 Amazon RDS API 调试问题的信息，请参阅对 Aurora 上的应用程序进行故障排除。

无法连接到 Amazon RDS 数据库实例

当您无法连接到数据库实例时，常见原因如下：

- 入站规则 – 由本地防火墙强制执行的访问规则与您授权用于访问数据库实例的 IP 地址可能不匹配。该问题很可能是由安全组中的入站规则所致。

  默认情况下，数据库实例不允许访问。通过与 VPC 关联的安全组授予访问权限，该安全组允许流量进出数据库实例。如有必要，请将针对您具体情况的入站和出站规则添加到安全组。您可以指定一个 IP 地址、IP 地址范围或另一个 VPC 安全组。

  > **Note**
  >
  > 添加新入站规则时，您可以为源选择我的 IP，以允许从浏览器中检测到的 IP 地址访问数据库实例。

  有关设置安全组的更多信息，请参阅通过创建安全组提供对 VPC 中数据库集群的访问。

> **Note**
>
> 不允许来自 169.254.0.0/16 范围内的 IP 地址的客户端连接。这是自动私有 IP 寻址范围 (APIPA)，它用于本地链路寻址。

- 公开可用性 – 要从 VPC 外部连接到您的数据库实例（例如通过使用客户端应用程序），实例必须具有向其分配的公有 IP 地址。

  要使实例公开可用，请修改它，在公开可用性下选择是。有关更多信息，请参阅 对互联网隐藏 VPC 中的数据库集群。
- 端口 – 由于您的本地防火墙限制，无法使用您在创建数据库实例时指定的端口来发送和接收通信。如需确定您的网络是否允许指定端口用于入站和出站通信，请咨询您的网络管理员。
- 可用性 – 对于新创建的数据库实例，数据库实例具有 `creating` 状态，直到该数据库实例可供使用。当状态变为 `available` 时，您可以连接到该数据库实例。根据数据库实例的大小，可能要用最长 20 分钟，实例才可用。
- 互联网网关 – 对于希望可公开访问的数据库实例，其数据库子网组中的子网必须具有互联网网关。

  为子网配置互联网网关

  1. 登录AWS Management Console并通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
  2. 在导航窗格中，选择 Databases (数据库)，然后选择数据库实例的名称。
  3. 在 Connectivity & security (连接和安全) 选项卡中，记录 VPC 下 VPC ID 的值以及 Subnets (子网) 下子网 ID 的值。
  4. 通过以下网址打开 Amazon VPC 控制台：https://console.aws.amazon.com/vpc/。
  5. 在导航窗格中，选择 Internet Gateways（互联网网关）。验证是否有 Internet 网关附加到您的 VPC。否则，选择 Create Internet Gateway 以创建 Internet 网关。选择 Internet 网关，然后选择 Attach to VPC 并按照说明将其附加到您的 VPC。
  6. 在导航窗格中，选择 Subnets，然后选择您的子网。
  7. 在 Route Table 选项卡上，验证 `0.0.0.0/0` 的路由是否为目的地以及您的 VPC 的 Internet 网关是否为目标。

     如果您使用实例的 IPv6 地址连接到实例，请检查是否有一个路由可以将所有 IPv6 流量 (`::/0`) 指向 Internet 网关。否则请执行以下操作：

     a. 选择路由表的 ID (rtb-xxxxxxxx) 以导航到路由表。

b. 在 Routes（路由）选项卡上，选择 Edit routes（编辑路由）。选择 Add route (添加路由)，将 `0.0.0.0/0` 用作目的地并将 Internet 网关用作目标。

   对于 IPv6，选择 Add route (添加路由)，将 `::/0` 用作目的地并将 Internet 网关用作目标。

c. 选择 Save routes（保存路由）。

此外，如果您尝试连接到 IPv6 端点，请确保客户端 IPv6 地址范围已获得连接到数据库实例的授权。

有关更多信息，请参阅 在 VPC 中使用数据库集群。

测试与数据库实例的连接

您可以使用常见 Linux 或 Microsoft Windows 工具测试与数据库实例的连接。

从 Linux 或 Unix 终端，您可以通过输入以下命令来测试连接。将 *DB-instance-endpoint* 替换为端点，并将 *port* 替换为数据库实例的端口。

```
nc -zv DB-instance-endpoint port
```

例如，下面显示了示例命令和返回值。

```
nc -zv postgresql1.c6c8mn7fake0.us-west-2.rds.amazonaws.com 8299

  Connection to postgresql1.c6c8mn7fake0.us-west-2.rds.amazonaws.com 8299 port [tcp/
vvr-data] succeeded!
```

Windows 用户可使用 Telnet 测试与数据库实例的连接。Telnet 操作不支持除测试连接之外的用途。如果连接成功，则该操作不返回任何消息。如果连接失败，则您将收到一条错误消息，诸如以下内容。

```
C:\>telnet sg-postgresql1.c6c8mntfake0.us-west-2.rds.amazonaws.com 819

  Connecting To sg-postgresql1.c6c8mntfake0.us-west-2.rds.amazonaws.com...Could not
 open
  connection to the host, on port 819: Connect failed
```

如果 Telnet 操作返回成功值，则您的安全组配置正确。

> **Note**
>
> Amazon RDS 不接受 Internet 控制消息协议 (ICMP) 流量，包括 ping。

排查连接身份验证错误

在某些情况下，您可以连接到数据库实例，但会收到身份验证错误。在这些情况下，您可能需要重置数据库实例的主用户密码。您可以通过修改 RDS 实例来做到这一点。

Amazon RDS 安全性问题

为避免出现安全性问题，切勿为用户账户使用您的 AWS 账户根用户电子邮件地址和密码。最佳做法是使用根用户来创建用户，并将这些用户分配给数据库用户账户。还可使用根用户创建其他用户账户（如有必要）。

有关创建用户的信息，请参阅在 AWS 账户 中创建 IAM 用户。有关在 AWS IAM Identity Center 中创建用户的信息，请参阅管理 IAM Identity Center 中的身份。

错误消息“无法检索账户属性，某些控制台功能可能受损。”

您可能会由于几个原因而收到此错误。这可能是因为您的账户缺少权限，或者您的账户尚未正确设置。如果您的账户是新账户，可能该账户还没准备好。如果是现有账户，您的访问策略中可能缺少执行特定操作的权限（如，创建数据库实例）。要修复该问题，您的管理员需要为您的账户提供必要的角色。有关更多信息，请参阅 IAM 文档。

重置数据库实例拥有者密码

如果您被锁定在数据库集群之外，则可以以主用户身份登录。然后，您可以重置其他管理用户或角色的凭证。如果您无法以主用户身份登录，AWS 账户所有者可以重置主用户密码。有关可能需要重置哪些管理账户或角色的详细信息，请参阅主用户账户权限。

您可以使用 Amazon RDS 控制台、AWS CLI 命令 modify-db-instance，或者使用 ModifyDBInstance API 操作更改数据库实例密码。有关修改数据库集群中的数据库实例的更多信息，请参阅 修改数据库集群中的数据库实例。

Amazon RDS 数据库实例中断或重新引导

重新启动数据库实例时可能会发生数据库实例中断。在将数据库实例置于某种禁止对其访问的状态，以及重新启动数据库时会出现这种情况。当您手动重启数据库实例时，可能会发生重启。当您更改需要在重启后才生效的数据库实例设置时，也可能会发生重启。

当您更改需要重启的设置或手动引起重启时，才会发生数据库实例重启。如果您更改某个设置并请求此更改立即生效，则可能会立即发生重启。或者，可能在数据库实例的维护时段内发生重启。

在出现下列情况之一时，会立即发生数据库实例重启：

- 您将数据库实例的备份保留期从 0 更改为非零值或从非零值更改为 0。然后，将 Apply Immediately（立即应用）设置为 `true`。
- 更改数据库实例类，并将立即应用设置为 `true`。

在维护时段内，如果出现下列情况之一，则会发生数据库实例重启：

- 将数据库实例的备份保留期从 0 更改为非零值或从非零值更改为 0 并将立即应用设置为 `false`。
- 更改数据库实例类，并将立即应用设置为 `false`。

当您更改数据库参数组中的静态参数时，在与该参数组关联的数据库实例重启之前，此更改将不会生效。更改需要手动重新启动。在维护时段内，数据库实例不会自动重新启动。

Amazon RDS 数据库参数更改未生效

在某些情况下，您可能会更改数据库参数组中的参数，但未看到更改生效。如果是这样，您可能需要重新启动与数据库参数组关联的数据库实例。当您更改动态参数时，更改将立即生效。当您更改静态参数时，在与该参数组关联的数据库实例重启之前，此更改将不会生效。

您可以使用 RDS 控制台重启数据库实例。或者，您可以显式调用 `RebootDBInstance` API 操作。如果数据库实例处于多可用区部署中，则可以在不进行失效转移的情况下重启。在静态参数更改后重新引导关联的数据库实例的要求，可帮助缓解影响 API 调用的参数误配置的风险。调用 `ModifyDBInstance` 以更改数据库实例类就是这样的一个例子。有关更多信息，请参阅 在 Amazon Aurora 中修改数据库参数组中的参数。

Amazon Aurora 中的可用内存问题

可用内存 是数据库实例上可供数据库引擎使用的总随机存取内存 (RAM)。这是可用操作系统 (OS) 内存以及可用缓冲区和页面缓存的总和。数据库引擎使用主机上的大部分内存，但操作系统进程也使用一些 RAM。当前分配给数据库引擎或操作系统进程使用的内存不包括在可用内存中。当数据库引擎内存不足时，数据库实例可以使用通常用于缓冲和缓存的临时空间。如前所述，这个临时空间包含在可用内存中。

您可以使用 Amazon CloudWatch 中的 `FreeableMemory` 指标来监控可用内存。有关更多信息，请参阅 Amazon Aurora 的监控工具。

如果数据库实例始终可用内存不足或使用交换空间，则考虑纵向扩展至更大的数据库实例类。有关更多信息，请参阅Amazon Aurora 数据库实例类。

您还可以更改内存设置。例如，在 Aurora MySQL 上，您可以调整 `innodb_buffer_pool_size` 参数的大小。默认情况下，此参数设置为物理内存的 75%。有关更多 MySQL 故障排除提示，请参阅如何排查 Amazon RDS for MySQL 数据库中的可用内存不足问题？

对于 Aurora Serverless v2，`FreeableMemory` 表示当 Aurora Serverless v2 数据库实例已扩展到其最大容量时，可用的未使用内存量。您可能已将实例缩减到相对较低的容量，但它仍然报告较高的 `FreeableMemory` 值，因为实例可以纵向扩展。该内存目前不可用，但如果您需要，您可以获取它。

对于当前容量低于最大容量的每个 Aurora 容量单位 (ACU)，`FreeableMemory` 增加大约 2 GiB。因此，在数据库实例扩展到它可以达到的最大值之前，此指标不会接近零。

如果此指标接近值 `0`，则数据库实例已纵向扩展到它能达到的最大容量。它已接近其可用内存的极限。考虑增大集群的最大 ACU 设置。如果读取器数据库实例上的这个指标接近值 `0`，请考虑向集群添加额外的读取器数据库实例。这样，工作负载的只读部分可以分散到更多数据库实例上，从而减少每个读取器数据库实例的内存使用。有关更多信息，请参阅 适用于 Aurora Serverless v2 的重要 Amazon CloudWatch 指标。

对于 Aurora Serverless v1，您可以更改容量范围以使用更多 ACU。有关更多信息，请参阅 修改 Aurora Serverless v1 数据库集群。

Amazon Aurora MySQL 复制问题

一些 MySQL 复制问题也适用于 Aurora MySQL。您可以诊断并纠正这些问题。

主题

- 诊断并解决只读副本之间的滞后

- 诊断并解决 MySQL 读取复制故障
- 复制已停止错误
- 只读副本复制未能初始化元数据结构

诊断并解决只读副本之间的滞后

在创建一个 MySQL 只读副本且该只读副本可用后，Amazon RDS 首先将复制自只读副本创建操作启动以来对源数据库实例所做的更改。在此期间，只读副本的复制滞后时间将大于 0。您可以在 Amazon CloudWatch 中通过查看 Amazon RDS `AuroraBinlogReplicaLag` 指标来监控此滞后时间。

`AuroraBinlogReplicaLag` 指标报告 MySQL `Seconds_Behind_Master` 命令的 `SHOW REPLICA STATUS` 字段的值。有关更多信息，请参阅 MySQL 文档中的 SHOW REPLICA STATUS 语句。

当 `AuroraBinlogReplicaLag` 指标达到 0 时，即表示副本已赶上源数据库实例进度。如果 `AuroraBinlogReplicaLag` 指标返回 -1，则副本可能为未激活状态。要纠正复制错误，请参阅诊断并解决 MySQL 读取复制故障。`AuroraBinlogReplicaLag` 值为 -1 还可能意味着 `Seconds_Behind_Master` 值无法确定或为 `NULL`。

> **Note**
>
> 以前的 Aurora MySQL 版本使用的是 `SHOW SLAVE STATUS`，而不是 `SHOW REPLICA STATUS`。如果您使用 Aurora MySQL 版本 1 或 2，那么请使用 `SHOW SLAVE STATUS`。将 `SHOW REPLICA STATUS` 用于 Aurora MySQL 版本 3 及更高版本。

`AuroraBinlogReplicaLag` 指标将在网络中断期间或在维护时段内应用修补程序时返回 -1。在这种情况下，在再次检查 `AuroraBinlogReplicaLag` 指标之前，需等待网络连接恢复或维护时段结束。

MySQL 只读复制技术是异步的。因此，预计源数据库实例上的 `BinLogDiskUsage` 指标和只读副本上的 `AuroraBinlogReplicaLag` 指标偶尔会增加。例如，请考虑对源数据库实例并行进行大量写入操作的情况。同时，对只读副本的写入操作会使用单个 I/O 线程序列化。这种情况可能会导致源实例与只读副本之间的滞后。

有关只读副本和 MySQL 的更多信息，请参阅 MySQL 文档中的复制实施详细信息。

您可降低对源数据库实例的更新与对只读副本的后续更新之间的滞后，方式如下：

- 将只读副本的数据库实例类的存储大小设置为与源数据库实例的类似。

- 确保源数据库实例和只读副本使用的数据库参数组中的参数设置兼容。有关更多信息和示例，请参阅下一部分中有关 max_allowed_packet 参数的讨论。
- 禁用查询缓存。对于经常修改的表，使用查询缓存可能会加大副本滞后，因为缓存已锁定且会频繁刷新。如果是这样的话，您可能会发现在禁用查询缓存的情况下副本滞后较少。您可通过在数据库实例的数据库参数组中将 query_cache_type parameter 设置为 0 来禁用查询缓存。有关查询缓存的更多信息，请参阅查询缓存配置。
- 为 MySQL 的 InnoDB 预热只读副本上的缓冲池。例如，假设您有少量经常更新的表，并且您正在使用 InnoDB 或 XtraDB 表架构。在这种情况下，将这些表转储到只读副本上。这样做将促使数据库引擎从磁盘扫描这些表的行，然后将它们缓存到缓冲池中。此方法可以减少副本滞后。下面是一个示例。

  对于 Linux、macOS 或 Unix：

```
PROMPT> mysqldump \
    -h <endpoint> \
    --port=<port> \
    -u=<username> \
    -p <password> \
    database_name table1 table2 > /dev/null
```

  对于 Windows：

```
PROMPT> mysqldump ^
    -h <endpoint> ^
    --port=<port> ^
    -u=<username> ^
    -p <password> ^
    database_name table1 table2 > /dev/null
```

诊断并解决 MySQL 读取复制故障

Amazon RDS 会监控只读副本的复制状态。如果由于任何原因停止复制，则 RDS 将只读副本实例的 Replication State（复制状态）字段更新为 Error。您可通过查看复制错误字段，检查 MySQL 引擎引发的关联错误的详细信息。还生成指示只读副本状态的事件，包括 RDS-EVENT-0045、RDS-EVENT-0046 和 RDS-EVENT-0057。有关这些事件和事件订阅的详细信息，请参阅 使用 Amazon RDS 事件通知。如果返回 MySQL 错误消息，则在 MySQL 错误消息文档中查看错误。

可导致复制出错的常见情况包括：

- 只读副本的 max_allowed_packet 参数的值小于源数据库实例的 max_allowed_packet 参数。

  max_allowed_packet 参数是您可以在数据库参数组中设置的自定义参数。max_allowed_packet 参数用于指定可在数据库上运行的数据操作语言 (DML) 的最大大小。在某些情况下，源数据库实例的 max_allowed_packet 值可能大于只读副本的 max_allowed_packet 值。如果是这样，复制过程可能会引发错误并停止复制。最常见的错误是 packet bigger than 'max_allowed_packet' bytes。通过将源和只读副本设置为使用具有相同 max_allowed_packet 参数值的数据库参数组，即可更正此错误。
- 对只读副本上的表进行写入操作。如果是在只读副本上创建索引，则需要将 read_only 参数设置为 0 才能创建索引。如果对只读副本上的表进行写入操作，则会中断复制。
- 使用 MyISAM 等非事务性存储引擎。只读副本需要使用事务性存储引擎。只有以下存储引擎支持复制：适用于 MySQL 或 MariaDB 的 InnoDB。

  您可以使用以下命令将 MyISAM 表转换为 InnoDB：

  ```
  alter table <schema>.<table_name> engine=innodb;
  ```
- 使用不安全的不确定性查询，如 SYSDATE()。有关更多信息，请参阅 MySQL 文档中的确定二进制日志记录中的安全和不安全语句。

下列步骤可帮助您纠正复制错误：

- 如果您遇到逻辑错误并且可安全跳过该错误，则可执行跳过当前的复制错误中所述的步骤。您的 Aurora MySQL 数据库实例必须运行包括 mysql_rds_skip_repl_error 过程的版本。有关更多信息，请参阅 mysql_rds_skip_repl_error。
- 如果您遇到二进制日志（binlog）位置问题，则可使用更改副本重放位置。您可以使用 Aurora MySQL 版本 1 和 2 的 mysql.rds_next_master_log 命令来执行此操作。您可以使用 Aurora MySQL 版本 3 及更高版本的 mysql.rds_next_source_log 命令来执行此操作。您的 Aurora MySQL 数据库实例必须运行支持此命令的版本以更改副本重放位置。有关版本信息，请参阅 mysql_rds_next_master_log。
- 如果您遇到因高 DML 负载导致的临时性能问题，则可在只读副本的数据库参数组中将 innodb_flush_log_at_trx_commit 参数设置为 2。这样做有助于只读副本保持同步，但这会临时降低原子性、一致性、隔离和持久性（ACID）属性。
- 您可以删除只读副本，使用相同的数据库实例标识符创建实例。这样，端点将保持与旧只读副本的端点相同。

如果复制错误得到纠正，则复制状态将更改为正在复制。有关更多信息，请参阅解决 MySQL 只读副本问题。

复制已停止错误

调用 mysql.rds_skip_repl_error 命令时，您可能会收到一条错误消息，指出副本已关闭或禁用。

出现该错误消息是由于复制已停止且无法重新启动。

如果您需要跳过大量错误，复制滞后时间可能会超出二进制日志文件的默认保留期。在这种情况下，您可能会遇到一个因二进制日志文件在副本上进行重放之前被清除而引发的严重错误。此清除会导致复制停止，而您将无法再调用 mysql.rds_skip_repl_error 命令以跳过复制错误。

您可以通过增加二进制日志文件在复制源上保留的小时数来缓解该问题。在增加二进制日志保留时间后，您可以重新启动复制进程，并根据需要调用 mysql.rds_skip_repl_error 命令。

若要设置二进制日志保留时间，请使用 mysql_rds_set_configuration 过程。指定“二进制日志保留小时数”的配置参数以及要在数据库集群上保留二进制日志文件的小时数（最多 2160 个小时，即 90 天）。Aurora MySQL 的默认值为 24（1 天）。以下示例将 binlog 文件的保留期设置为 48 个小时。

```
CALL mysql.rds_set_configuration('binlog retention hours', 48);
```

只读副本复制未能初始化元数据结构

在您尝试启动复制时，收到了以下错误消息：

```
Read Replica Replication Error - SQLError: 13124, reason: Replica failed to initialize
 applier metadata structure from the repository
```

当副本的元数据结构有问题时，就会出现此错误。要修复元数据结构，您必须创建新副本。

为了防止以后出现此问题，请执行以下操作之一：

- 如果可能，请在副本上禁用多线程。从 MySQL 8.0.27 开始，默认情况下启用了多线程。
- 如果您需要在副本上使用多线程，建议您使用基于 GTID 的复制。有关更多信息，请参阅 使用基于 GTID 的复制。

Amazon RDS API 参考

除了 AWS Management Console 和 AWS Command Line Interface（AWS CLI）之外，Amazon RDS 还提供了一个 API。您可以使用 API 自动执行管理 Amazon RDS 中的数据库实例和其他对象的任务。

- 有关 API 操作的字母顺序列表，请参阅操作。
- 有关数据类型的字母顺序列表，请参阅数据类型。
- 有关常用查询参数的列表，请参阅常用参数。
- 有关错误代码的描述，请参阅常见错误。

有关 AWS CLI 的更多信息，请参阅适用于 Amazon RDS 的 AWS Command Line Interface 参考。

主题

- 使用查询 API
- 对 Aurora 上的应用程序进行故障排除

使用查询 API

以下各节简要讨论用于查询 API 的参数和请求身份验证。

有关查询 API 工作原理的一般信息，请参阅 Amazon EC2 API Reference 中的 查询请求。

查询参数

HTTP 基于查询的请求是指使用 HTTP 动作 GET 或 POST 的 HTTP 请求，查询参数的名称为 `Action`。

每个查询请求必须包括一些通用参数，以处理操作的身份验证和选择事宜。

有些操作会使用参数列表。这些列表都是使用 `param.`*n* 表示法指定的。*n* 值是从 1 开始的整数。

有关 Amazon RDS 区域和端点的信息，请转至《Amazon Web Services 一般参考》的“区域和端点”部分中的 Amazon Relational Database Service（RDS）。

查询请求身份验证

您只能通过 HTTPS 发送查询请求，并且每个查询请求中必须包含签名。您必须使用 AWS 签名版本 4 或者签名版本 2。有关更多信息，请参阅签名版本 4 签名流程和签名版本 2 签名流程。

对 Aurora 上的应用程序进行故障排除

Amazon RDS 提供具体的描述性错误，以帮助您在与 Amazon RDS API 互动时排查问题。

主题

- 检索错误
- 故障排除技巧

有关对 Amazon RDS 数据库实例进行故障排除的信息，请参阅 Amazon Aurora 故障排除。

检索错误

通常，在您花费任何时间处理错误结果之前，您都会希望您的应用程序检查某个请求是否生成错误。查明是否出现错误的最简单方法是寻找 Amazon RDS API 中做出响应的 `Error` 节点。

XPath 语法提供了一种搜索 `Error` 节点存在的简单方法。它还提供了一种相对简单的方法来检索错误代码和消息。下面的代码片段采用 Perl 和 XML::XPath 模块来确定在请求期间是否出现错误。如果出现了错误，那么代码会刊载第一个错误代码和响应信息。

```
use XML::XPath;
    my $xp = XML::XPath->new(xml =>$response);
    if ( $xp->find("//Error") )
    {print "There was an error processing your request:\n", " Error code: ",
    $xp->findvalue("//Error[1]/Code"), "\n", " ",
    $xp->findvalue("//Error[1]/Message"), "\n\n"; }
```

故障排除技巧

我们建议采用下列流程来诊断和解决 Amazon RDS API 的问题：

- 通过检查 http://status.aws.amazon.com 来验证 Amazon RDS 在您的目标 AWS 区域中正常运行。
- 检查您的请求结构。

  每个 Amazon RDS 操作都在 Amazon RDS API Reference 中有一个参考页面。复查您正在使用的参数是否正确。要了解关于潜在错误内容的意见，请查看示例请求或用户场景，以查看这些示例是否执行类似操作。
- 检查 AWS re:Post。

Amazon RDS 有一个开发社区，您可以在其中搜索他人在开发过程中遇到的问题以及解决方案。如要查看主题，请转至 AWS re:Post。

文档历史记录

当前 API 版本：2014-10-31

下表介绍了 Amazon Aurora 用户指南 的重要更改。如需对此文档更新的通知，您可以订阅 RSS 源。有关 Amazon Relational Database Service (Amazon RDS) 的信息，请参阅 Amazon Relational Database Service 用户指南。

> **Note**
>
> 在 2018 年 8 月 31 日之前，在 Amazon Relational Database Service 用户指南 中介绍了 Amazon Aurora。有关早期的 Aurora 文档历史记录，请参阅 Amazon Relational Database Service 用户指南 中的文档历史记录。

您可以在数据库的新增功能？页面上筛选新的 Amazon Aurora 特征。对于 Products (产品)，请选择 Amazon Aurora。然后使用 **`global database`** 或 **`Serverless`** 之类的关键词进行搜索。

变更	说明	日期
更新现有策略	Amazon RDS 从 `AWSServiceRoleForRDSBeta` 服务相关角色的 `AmazonRDSBetaServiceRolePolicy` 中删除了 `sns:Publish` 权限。有关更多信息，请参阅 Amazon RDS 对 AWS 托管式策略的更新。	2024 年 8 月 7 日
更新现有策略	Amazon RDS 从 `AWSServiceRoleForRDSPreview` 服务相关角色的 `AmazonRDSPreviewServiceRolePolicy` 中删除了 `sns:Publish` 权限。有关更多信息，请参阅 Amazon RDS 对 AWS 托管式策略的更新。	2024 年 8 月 7 日

适用于 MySQL 的 AWS ODBC 驱动程序已正式发布	适用于 MySQL 的 Amazon Web Services（AWS）ODBC 驱动程序是专为实现 Aurora MySQL 的高可用性而设计的客户端驱动程序。有关更多信息，请参阅使用适用于 MySQL 的 Amazon Web Services（AWS）ODBC 驱动程序连接到 Aurora MySQL。	2024 年 7 月 18 日
适用于 Aurora Serverless v2 的 RDS 数据 API 在更多区域中推出	RDS 数据 API 现已在其它几个 AWS 区域中面向 Aurora PostgreSQL 推出。有关 RDS 数据 API 的区域支持的信息，请参阅适用于 Aurora PostgreSQL Serverless v2 和预调配的数据 API。	2024 年 7 月 9 日
AWS Python 驱动程序现已正式发布	Amazon Web Services（AWS）驱动程序设计为高级 Python 包装器。这款包装器是对开源 Psycopg 驱动程序的各项功能的补充和扩展。有关更多信息，请参阅使用 AWS 驱动程序连接到 Aurora 数据库集群。	2024 年 5 月 23 日
零 ETL 集成功能现已在中国区推出	零 ETL 集成现已在中国（北京）和中国（宁夏）这两个 AWS 区域推出。有关更多信息，请参阅与 Amazon Redshift 进行零 ETL 集成。	2024 年 5 月 21 日

RDS 代理现已在更多区域推出	RDS 代理现已在亚太地区（海得拉巴）、亚太地区（墨尔本）、中东（阿联酋）、以色列（特拉维夫）、加拿大西部（卡尔加里）和欧洲（苏黎世）区域推出。有关 RDS 代理的更多信息，请参阅使用 Amazon RDS 代理。	2024 年 5 月 21 日
Amazon RDS Extended Support	现在，创建或还原 Aurora MySQL 版本 2 或 3 或 Aurora PostgreSQL 版本 11 数据库会自动将该数据库注册到 Amazon RDS Extended Support，这样您的现有应用程序就可以继续按原样运行。您可以选择退出 RDS Extended Support，以避免在数据库引擎的 Aurora 标准支持终止日期后支付费用。有关更多信息，请参阅使用 Amazon RDS Extended Support。	2024 年 3 月 21 日
零 ETL 集成的数据筛选	Amazon RDS 支持在数据库和表级别进行数据筛选，以实现与 Amazon Redshift 的零 ETL 集成。有关更多信息，请参阅 Data filtering for Aurora zero-ETL integrations with Amazon Redshift。	2024 年 3 月 20 日

Aurora MySQL 与 Amazon Bedrock 集成	现在，您可以将 Amazon Aurora MySQL 数据库与 Amazon Bedrock 集成，为生成式人工智能应用程序提供支持。有关更多信息，请参阅 Using Amazon Aurora machine learning with Aurora MySQL。	2024 年 3 月 8 日
新 AWS 托管式策略	Amazon RDS 添加了一个名为 `AmazonRDSCustomInstanceProfileRolePolicy` 的新托管式策略，允许 RDS Custom 通过 EC2 实例配置文件执行自动化操作和数据库管理任务。有关更多信息，请参阅 Amazon RDS 对 AWS 托管式策略的更新。	2024 年 2 月 27 日
Amazon RDS 在以色列（特拉维夫）区域支持 AWS Secrets Manager	Amazon RDS 在以色列（特拉维夫）区域支持 Secrets Manager。有关更多信息，请参阅使用 Amazon RDS 和 AWS Secrets Manager 管理密码。	2024 年 2 月 21 日
Amazon RDS Extended Support	当您的数据库集群和全局集群中的 Aurora MySQL 和 Aurora PostgreSQL 主要引擎版本到达 Aurora 标准支持终止日期时，Amazon RDS 将自动启用 Amazon RDS Extended Support。有关更多信息，请参阅使用 Amazon RDS 扩展支持。	2024 年 2 月 15 日

Aurora PostgreSQL 16.1 支持 Babelfish for Aurora PostgreSQL 4.0.0	Aurora PostgreSQL 16.1 支持 Babelfish 4.0.0。有关新功能的列表，请参阅 16.1。有关每个 Babelfish 版本支持的特征列表，请参阅各版本 Babelfish 支持的功能。有关使用情况信息，请参阅使用 Babelfish for Aurora PostgreSQL。	2024 年 1 月 31 日
更新为默认 CA 证书	默认 CA 证书设置为 `rds-ca-rsa2048-g1` 。有关更多信息，请参阅使用 SSL/TLS 加密与数据库集群的连接。	2024 年 1 月 26 日
RDS 代理在欧洲（西班牙）区域推出	RDS 代理现已在欧洲（西班牙）区域推出。有关 RDS 代理的更多信息，请参阅使用 Amazon RDS 代理。	2024 年 1 月 8 日
RDS 数据 API 与 Aurora PostgreSQL Serverless v2 和预调配	现在，您可以将 RDS 数据 API 与 Aurora PostgreSQL Serverless v2 和预调配数据库集群结合使用。借助 RDS 数据 API，您可以通过安全的 HTTP 端点访问您的 Aurora 集群并运行 SQL 语句，而无需使用数据库驱动程序或管理连接。有关更多信息，请参阅使用 RDS 数据 API。	2023 年 12 月 21 日

Aurora PostgreSQL 与 Amazon Bedrock 集成	现在，您可以将 Amazon Aurora PostgreSQL 数据库与 Amazon Bedrock 集成，为生成式人工智能应用程序提供支持。有关更多信息，请参阅将 Amazon Aurora 机器学习与 Aurora PostgreSQL 结合使用。	2023 年 12 月 21 日
Amazon Aurora 在加拿大西部（卡尔加里）区域推出	Amazon Aurora 现已在加拿大西部（卡尔加里）区域推出。有关更多信息，请参阅区域和可用区。	2023 年 12 月 20 日
Amazon RDS 支持查看和响应建议	Amazon Aurora 建议现在包括基于阈值的主动建议和基于机器学习的被动建议。有关更多信息，请参阅 Amazon Aurora 的建议。	2023 年 12 月 19 日
Aurora PostgreSQL 与 Amazon Redshift 的零 ETL 集成（预览版）	现在，您可以使用 Aurora PostgreSQL 源数据库集群创建与 Amazon Redshift 的零 ETL 集成。对于预览版，您必须在美国东部（俄亥俄州）（us-east-2）AWS 区域的 Amazon RDS 数据库预览环境中创建所有集成。有关更多信息，请参阅使用 Aurora 与 Amazon Redshift 的零 ETL 集成。	2023 年 11 月 28 日
Amazon Aurora PostgreSQL 支持全局数据库写入转发	现在，您可以在基于 Aurora PostgreSQL 的全局数据库中的辅助集群上启用写入转发。有关更多信息，请参阅在 Aurora PostgreSQL 全局数据库中使用写入转发。	2023 年 11 月 9 日

Aurora PostgreSQL 支持优化型读取	使用 Aurora 优化型读取功能，您可以实现更快的 Aurora PostgreSQL 查询处理。有关更多信息，请参阅使用 Aurora 优化型读取提高 Aurora PostgreSQL 的查询性能。	2023 年 11 月 8 日
Amazon RDS 将 Performance Insights 指标导出到 Amazon CloudWatch	Performance Insights 允许您将预配置或自定义的指标控制面板导出到 Amazon CloudWatch。导出的指标控制面板可在 CloudWatch 控制台中查看。您还可以导出选定的 Performance Insights 指标小组件，并在 CloudWatch 控制台中查看指标数据。有关更多信息，请参阅将 Performance Insights 指标导出到 CloudWatch。	2023 年 11 月 8 日
Aurora MySQL 与 Amazon Redshift 的零 ETL 集成正式推出	与 Amazon Redshift 的零 ETL 集成现已对 Aurora MySQL 正式推出。有关更多信息，请参阅使用 Aurora 与 Amazon Redshift 的零 ETL 集成。	2023 年 11 月 7 日
Aurora PostgreSQL 支持 RDS 蓝绿部署	现在，您可以从 Aurora PostgreSQL 数据库集群创建蓝绿部署。有关更多信息，请参阅使用 Amazon RDS 蓝绿部署进行数据库更新。	2023 年 10 月 26 日

Aurora MySQL 支持使用 AWS KMS keys（SSE-KMS）进行服务器端加密	在 Aurora MySQL 3.05 及更高版本中，您可以使用 SSE-KMS（包括 AWS 托管式密钥和客户管理密钥）对从 Amazon S3 加载或保存到 Amazon S3 的数据进行服务器端加密。有关更多信息，请参阅将数据从 Amazon S3 桶中的文本文件加载到 Amazon Aurora MySQL 数据库集群以及将数据从 Amazon Aurora MySQL 数据库集群保存到 Amazon S3 桶中的文本文件。	2023 年 10 月 25 日
Aurora MySQL 优化缩短了数据库重启时间	在 Aurora MySQL 3.05 及更高版本中，我们引入了缩短数据库重启时间的优化措施。与未进行优化相比，这些优化措施可减少多达 65% 的停机时间，并且在重启后数据库工作负载的中断也会减少。有关更多信息，请参阅进行优化以缩短数据库重启时间。	2023 年 10 月 25 日
对 AWS 托管式策略的更新	`AmazonRDSPerformanceInsightsReadOnly` 和 `AmazonRDSPerformanceInsightsFullAccess` 托管式策略现在将 `Sid`（语句 ID）作为标识符包括在策略语句中。有关更多信息，请参阅 Amazon RDS 对 AWS 托管式策略的更新。	2023 年 10 月 23 日

Amazon RDS 向 Amazon CloudWatch 发布了 Performance Insights 计数器指标	CloudWatch 控制台中的 DB_PERF_INSIGHTS 指标数学函数允许您查询 Amazon RDS 以获取性能详情计数器指标。有关更多信息，请参阅创建 CloudWatch 警报以监控 Amazon Aurora。	2023 年 9 月 20 日
Amazon Aurora 支持通过 AWS Backup 进行时间点故障恢复（PITR）	您现在可以在 AWS Backup 中管理 Aurora 自动（连续）备份，并从其中还原到指定时间。有关更多信息，请参阅使用 AWS Backup 将数据库集群还原到指定时间。	2023 年 9 月 7 日
Amazon RDS 扩展支持	Amazon Aurora 宣布，在 Aurora 标准支持终止日期后，将能够在您的数据库实例中继续运行 Aurora MySQL 和 Aurora PostgreSQL 主要引擎版本。有关更多信息，请参阅使用 Amazon RDS 扩展支持。	2023 年 9 月 1 日
Amazon Aurora MySQL 扩展了对 Percona XtraBackup 的支持	现在，您可以将 MySQL 8.0 数据库物理迁移到 Aurora MySQL 版本 3 数据库集群。有关更多信息，请参阅使用 Percona XtraBackup 和 Amazon S3 从 MySQL 进行物理迁移。	2023 年 8 月 24 日

Aurora 全球数据库支持全球数据库故障转移	Aurora 全球数据库现在支持托管式全球故障转移，让您能够更轻松地从真实的区域性灾难或完全的服务级别中断中恢复。要了解有关此特征的更多信息，请参阅对 Aurora 全球数据库执行托管式计划内故障转移。这项以前称为“托管式计划内故障转移”的特征现在称为“切换”。有关切换的相关信息，请参阅对 Amazon Aurora 全球数据库执行切换。	2023 年 8 月 21 日
更新 AWS 托管式策略权限	AmazonRDSFullAccess 托管式策略具有新权限，允许您生成、查看和删除时间段内的性能分析报告。有关更多信息，请参阅 Amazon RDS 对 AWS 托管式策略的更新。	2023 年 8 月 17 日
更新 AWS 托管式策略权限	向 AmazonRDSPerformanceInsightsReadOnly 托管式策略添加新权限和添加新的托管式策略 AmazonRDSPerformanceInsightsFullAccess ，可让您生成时间段内的数据库负载分析报告。有关更多信息，请参阅 Amazon RDS 对 AWS 托管式策略的更新。	2023 年 8 月 16 日
Amazon RDS 支持使用 Performance Insights 进行数据库负载时间段分析	Performance Insights 允许您创建特定时间段的性能分析报告。该报告提供已确定的见解和解决性能问题的建议。有关更多信息，请参阅分析时间段内的数据库负载。	2023 年 8 月 16 日

Amazon Aurora 支持保留数据库集群的自动备份	现在，您可以保留已删除的 Aurora 集群的自动备份，并将其还原到指定的时间点。有关更多信息，请参阅保留自动备份。	2023 年 8 月 4 日
Amazon Aurora 在以色列（特拉维夫）区域推出	Amazon Aurora 现已在以色列（特拉维夫）区域推出。有关更多信息，请参阅区域和可用区。	2023 年 8 月 1 日
Amazon Aurora MySQL 支持本地（集群内）写入转发	现在，您可以将写入操作从读取器数据库实例转发到 Aurora MySQL 数据库集群中的写入器数据库实例。有关更多信息，请参阅在 Amazon Aurora MySQL 数据库集群中使用写入转发。	2023 年 7 月 31 日
Amazon Aurora 在更多 AWS 区域中支持 Aurora Serverless v2	您现在可以在亚太地区（墨尔本）AWS 区域创建 Aurora Serverless v2 数据库集群。有关 Aurora Serverless v2 的更多信息，请参阅使用 Aurora Serverless v2。	2023 年 6 月 28 日
Amazon Aurora 推出与 Amazon Redshift 的零 ETL 集成（预览版）	零 ETL 集成提供了一个完全托管式解决方案，使事务数据在写入 Aurora MySQL 数据库集群后的几秒钟内即可在 Amazon Redshift 中可用。有关更多信息，请参阅使用 Aurora 与 Amazon Redshift 的零 ETL 集成。	2023 年 6 月 28 日

Amazon RDS 在 Performance Insights 控制面板中提供组合的性能详情和 CloudWatch 指标视图	Amazon RDS 现在在 Performance Insights 控制面板中提供 Performance Insights 和 CloudWatch 指标的合并视图。有关更多信息，请参阅使用性能详情控制面板查看组合指标。	2023 年 5 月 24 日
Amazon Aurora 支持 db.r7g 实例类	现在可以使用 db.r7g 实例类来创建 Aurora 数据库集群。有关更多信息，请参阅 Aurora 数据库实例类。	2023 年 5 月 11 日
Amazon Aurora 支持新的数据库集群存储配置	使用 Aurora I/O-Optimized，您只需为数据库集群的使用量和存储空间付费，而无需为读取和写入 I/O 操作支付额外费用。有关更多信息，请参阅 Amazon Aurora 数据库集群的存储配置。	2023 年 5 月 11 日
Amazon Aurora 在更多 AWS 区域中支持 Aurora Serverless v2	现在，您可以在以下 AWS 区域中创建 Aurora Serverless v2 数据库集群：亚太地区（海得拉巴）、欧洲（西班牙）、欧洲（苏黎世）和中东地区（阿联酋）。有关 Aurora Serverless v2 的更多信息，请参阅使用 Aurora Serverless v2。	2023 年 5 月 4 日
Aurora Serverless v1 支持转换为预调配	您可以将 Aurora Serverless v1 数据库集群直接转换为预调配的数据库集群。有关更多信息，请参阅将 Aurora Serverless v1 数据库集群转换为预调配。	2023 年 4 月 27 日

Aurora Serverless v1 支持 Amazon Aurora PostgreSQL 版本 13	现在，您可以创建运行 Aurora PostgreSQL 版本 13 的 Aurora Serverless v1 数据库集群。有关更多信息，请参阅 Aurora Serverless v1。	2023 年 4 月 27 日
Amazon Aurora 在中国区域对 AWS Secrets Manager 的支持	Amazon Aurora 在中国（北京）和中国（宁夏）区域支持 Secrets Manager。有关更多信息，请参阅使用 Amazon Aurora 和 AWS Secrets Manager 管理密码。	2023 年 4 月 20 日
Amazon Aurora 支持向主题订阅用户发布带有标签的事件	发送到 Amazon Simple Notification Service（Amazon SNS）或 Amazon EventBridge 的 Amazon Aurora 事件通知现在在消息正文中包含事件标签。这些标签提供受服务事件影响的资源数据。有关更多信息，请参阅 Amazon RDS 事件通知标签和属性。	2023 年 4 月 17 日
IAM 服务相关角色权限更新	`AmazonRDSFullAccess` 和 `AmazonRDSReadOnlyAccess` 策略现在授予额外的权限，以允许在 RDS 控制台中显示 Amazon DevOps Guru 调查发现。有关更多信息，请参阅 Amazon RDS 对 AWS 托管式策略的更新。	2023 年 3 月 30 日
Amazon Aurora 在亚太地区（墨尔本）区域支持全局数据库	现在，您可以在亚太地区（墨尔本）区域创建 Aurora 全局数据库。有关 Aurora 全局数据库的信息，请参阅使用 Amazon Aurora 全局数据库。	2023 年 3 月 22 日

更新 AWS 托管式策略权限	AmazonRDSFullAccess 和 AmazonRDSReadOnlyAccess 策略现在会向 Amazon CloudWatch 授予额外的权限。有关更多信息，请参阅 Amazon RDS 对 AWS 托管式策略的更新。	2023 年 3 月 16 日
RDS 代理在中国区域推出	RDS 代理现已在中国（北京）和中国（宁夏）区域推出。有关 RDS 代理的更多信息，请参阅使用 Amazon RDS 代理。	2023 年 3 月 15 日
Amazon Aurora 在中国区域支持 Aurora Serverless v2	Aurora Serverless v2 现已在中国（北京）和中国（宁夏）区域推出。有关更多信息，请参阅 Aurora Serverless v2。	2023 年 3 月 15 日
RDS 代理已在亚太地区（雅加达）区域推出	RDS 代理现已在亚太地区（雅加达）区域推出。有关 RDS 代理的更多信息，请参阅使用 Amazon RDS 代理。	2023 年 3 月 8 日
Amazon Aurora MySQL 支持 Kerberos 身份验证	现在，当用户连接到您的 Aurora MySQL 数据库集群时，您可以使用 Kerberos 身份验证来验证用户的身份。有关更多信息，请参阅对 Aurora MySQL 使用 Kerberos 身份验证。	2023 年 3 月 8 日

Amazon Aurora 在其他 AWS 区域中支持全局数据库	您现在可以在以下区域中创建 Aurora 全局数据库：非洲（开普敦）、亚太地区（香港）、亚太地区（海得拉巴）、亚太地区（雅加达）、欧洲地区（米兰）、欧洲（西班牙）、欧洲（苏黎世）、中东（巴林）和中东（阿联酋）。有关 Aurora 全局数据库的信息，请参阅使用 Amazon Aurora 全局数据库。	2023 年 3 月 6 日
Amazon Aurora 在其他 AWS 区域中支持复制数据库集群快照	现在，您可以在以下区域中复制数据库集群快照：非洲（开普敦）、亚太地区（香港）、亚太地区（海得拉巴）、亚太地区（雅加达）、亚太地区（墨尔本）、欧洲地区（米兰）、欧洲（西班牙）、欧洲（苏黎世）、中东（巴林）和中东（阿联酋）。有关复制数据库集群快照的信息，请参阅复制数据库集群快照。	2023 年 3 月 6 日
Amazon DevOps Guru for RDS 支持主动见解	Amazon DevOps Guru for RDS 发布主动见解及建议，以帮助您在预测 Aurora 数据库中的问题发生之前解决它们。有关更多信息，请参阅 DevOps Guru for RDS 工作原理。	2023 年 2 月 28 日
Amazon Aurora MySQL 版本 1 已弃用	Aurora MySQL 版本 1（与 MySQL 5.6 兼容）已弃用。有关更多信息，请参阅 Amazon Aurora 主要版本可用时间。	2023 年 2 月 28 日

Aurora Serverless v1 支持设置数据库集群维护时段	现在，您可以为 Aurora Serverless v1 数据库集群设置维护时段。有关更多信息，请参阅调整首选数据库集群维护时段。	2023 年 2 月 27 日
Amazon Aurora 在亚太地区（海得拉巴）、欧洲（西班牙）和中东（阿联酋）区域中支持数据库活动流。	有关更多信息，请参阅数据库活动流。	2023 年 1 月 27 日
Amazon Aurora 在亚太地区（墨尔本）区域中推出	Amazon Aurora 现已在亚太地区（墨尔本）区域中推出。有关更多信息，请参阅区域和可用区。	2023 年 1 月 23 日
在创建数据库集群期间指定证书颁发机构（CA）	现在，您可以在创建数据库集群期间指定将哪个 CA 用于数据库集群的服务器证书。有关更多信息，请参阅证书颁发机构。	2023 年 1 月 5 日
Aurora MySQL 3.* 支持回溯	Aurora MySQL 3.* 版本现在可以快速从用户错误中恢复，例如删除错误的表或错误的行。使用回溯可以将数据库回退到以前的某个时间点，无需从备份还原，即使是大型数据库也只需要几秒钟时间。有关详细信息，请参阅回溯 Aurora 数据库集群。	2023 年 1 月 4 日
在其他 AWS 区域中使用可用的 Amazon RDS 蓝绿部署	蓝绿部署特征现已在中国（北京）和中国（宁夏）区域推出。有关更多信息，请参阅使用 Amazon RDS 蓝绿部署进行数据库更新。	2022 年 12 月 22 日

IAM 服务相关角色权限更新	AmazonRDSServiceRolePolicy 策略现在会向 AWS Secrets Manager 授予额外的权限。有关更多信息，请参阅 Amazon RDS 对 AWS 托管式策略的更新。	2022 年 12 月 22 日
Amazon Aurora 与 AWS Secrets Manager 集成以进行密码管理	Aurora 可以在 Secrets Manager 中管理数据库集群的主用户密码。有关更多信息，请参阅使用 Amazon Aurora 和 AWS Secrets Manager 管理密码。	2022 年 12 月 22 日
Amazon Aurora 在更多 AWS 区域中支持 Aurora Serverless v2	Aurora Serverless v2 现已在非洲（开普敦）和欧洲（米兰）区域推出。有关更多信息，请参阅 Aurora Serverless v2。	2022 年 12 月 21 日
Aurora PostgreSQL 通过 PostgreSQL 14 支持 RDS 代理	现在，您可以使用 Aurora PostgreSQL 14 数据库集群来创建 RDS 代理。有关 RDS 代理的更多信息，请参阅使用 Amazon RDS 代理。	2022 年 12 月 13 日
Amazon Aurora 提醒您注意 Amazon DevOps Guru 检测到的最近异常情况	控制台的数据库详细信息页面提醒您注意当前和过去 24 小时内发生的异常情况。有关更多信息，请参阅 DevOps Guru for RDS 工作原理。	2022 年 12 月 13 日
Amazon RDS 代理支持全局数据库	您现在可以将 RDS 代理与 Aurora Global Database 一起使用。有关更多信息，请参阅将 RDS 代理与 Aurora Global Database 一起使用。	2022 年 12 月 7 日

Aurora PostgreSQL 数据库集群支持适用于 PostgreSQL 的可信语言扩展	适用于 PostgreSQL 的可信语言扩展是一个开源开发套件，它允许您构建高性能 PostgreSQL 扩展并在您的 Aurora PostgreSQL 数据库集群上安全地运行它们。有关更多信息，请参阅使用适用于 PostgreSQL 的可信语言扩展。	2022 年 11 月 30 日
Amazon GuardDuty RDS 保护可监控访问威胁	当您开启 GuardDuty RDS 保护时，GuardDuty 会使用来自 Aurora 数据库的 RDS 登录事件，监控这些事件，并对这些事件进行分析，以了解潜在的内部威胁或外部行为者。当 GuardDuty RDS 保护检测到潜在威胁时，GuardDuty 会生成新的调查结果，其中包含有关可能受感染的数据库的详细信息。有关更多信息，请参阅使用 GuardDuty RDS 保护监控威胁。	2022 年 11 月 30 日
使用 Amazon RDS 蓝绿部署进行数据库更新	您可以在暂存环境中对数据库集群进行更改，并在不影响生产数据库集群的情况下测试更改。准备就绪后，可以将暂存环境提升为新的生产环境，最大限度地减少停机时间。有关更多信息，请参阅使用 Amazon RDS 蓝绿部署进行数据库更新。	2022 年 11 月 27 日
Amazon Aurora 在亚太地区（海得拉巴）区域中推出	Amazon Aurora 现已在亚太地区（海得拉巴）区域中推出。有关更多信息，请参阅区域和可用区。	2022 年 11 月 22 日

Amazon Aurora 在欧洲（西班牙）区域中推出	Amazon Aurora 现已在欧洲（西班牙）区域中推出。有关更多信息，请参阅区域和可用区。	2022 年 11 月 16 日
Amazon Aurora 在欧洲（苏黎世）区域中推出	Amazon Aurora 现已在欧洲（苏黎世）区域中推出。有关更多信息，请参阅区域和可用区。	2022 年 11 月 9 日
Amazon Aurora 支持将数据从数据库集群导出到 Amazon S3	现在，您可以将 Aurora 集群数据直接导出到 S3，无需先创建快照。有关更多信息，请参阅将数据库集群数据导出到 Amazon S3。	2022 年 10 月 27 日
Amazon Aurora MySQL 支持更快地导出到 Amazon S3	现在，对于 MySQL 5.7 和 8.0 兼容的 Aurora MySQL 集群，在将数据库集群快照数据导出到 S3 时，速度可提高多达 10 倍。有关更多信息，请参阅将数据库集群数据导出到 Amazon S3。	2022 年 10 月 20 日
Amazon Aurora 支持自动设置 Aurora 数据库集群和 EC2 实例之间的连接	您可以使用 AWS Management Console设置现有 Aurora 数据库集群与 EC2 实例之间的连接。有关更多信息，请参阅自动连接 EC2 实例和 Aurora 数据库集群。	2022 年 10 月 14 日

适用于 PostgreSQL 的 AWS JDBC 驱动程序已正式发布	适用于 PostgreSQL 的 AWS JDBC 驱动程序是专为 Aurora PostgreSQL 而设计的客户端驱动程序。适用于 PostgreSQL 的 AWS JDBC 驱动程序现已正式发布。有关更多信息，请参阅连接适用于 PostgreSQL 的 AWS JDBC 驱动程序。	2022 年 10 月 6 日
Amazon Aurora 支持就地升级与 MySQL 5.7 兼容的 Aurora MySQL	您可以执行就地升级，以将与 MySQL 5.7 兼容的现有 Aurora MySQL 集群更改为与 MySQL 8.0 兼容的 Aurora MySQL 集群。有关更多信息，请参阅从 Aurora MySQL 2.x 升级到 3.x。	2022 年 9 月 26 日
Performance Insights 显示前 25 个 SQL 查询	在 Performance Insights 控制面板中，Top SQL（主要 SQL）选项卡将显示对数据库负载影响最大的 25 个 SQL 查询。有关详细信息，请参阅 Top SQL（主要 SQL）选项卡概览。	2022 年 9 月 13 日
Aurora MySQL 支持新的数据库实例类	您现在可以将 db.r6i 数据库实例类用于 Aurora MySQL 数据库集群。有关更多信息，请参阅数据库实例类。	2022 年 9 月 13 日
Amazon Aurora 在中东（阿联酋）区域中推出	Amazon Aurora 现已在中东（阿联酋）区域中推出。有关更多信息，请参阅区域和可用区。	2022 年 8 月 30 日

Amazon Aurora 支持自动设置与 EC2 实例的连接	创建 Aurora 数据库集群时，可以使用 AWS Management Console在 Amazon Elastic Compute Cloud 实例和新数据库集群之间设置连接。有关更多信息，请参阅使用 EC2 实例配置自动网络连接。	2022 年 8 月 22 日
Amazon Aurora 支持双堆栈模式	数据库集群现在可以在双堆栈模式下运行。在双堆栈模式下，资源可通过 IPv4 和/或 IPv6 与数据库集群进行通信。有关更多信息，请参阅 Amazon Aurora IP 寻址。	2022 年 8 月 17 日
Amazon Aurora 支持就地升级与 PostgreSQL 兼容的 Aurora Serverless v1	您可以对与 PostgreSQL 10 兼容的 Aurora Serverless v1 集群执行就地升级，以将现有集群更改为与 PostgreSQL 11 兼容的 Aurora Serverless v1 集群。有关就地升级过程，请参阅修改 Aurora Serverless v1 数据库集群。	2022 年 8 月 8 日
Performance Insights 支持亚太地区（雅加达）区域	以前，您无法在亚太地区（雅加达）区域使用 Performance Insights。这一限制已删除。有关更多信息，请参阅 Performance Insights 的 AWS 区域支持。	2022 年 7 月 21 日
Amazon Aurora 支持新的数据库实例类	您现在可以将 db.r6i 数据库实例类用于 Aurora PostgreSQL 数据库集群。有关更多信息，请参阅数据库实例类。	2022 年 7 月 14 日

RDS Performance Insights 支持额外的保留期	以前，Performance Insights 仅提供两个保留期：7 天（默认）或 2 年（731 天）。现在，如果您需要将性能数据保留超过 7 天，则可以指定 1-24 个月。有关更多信息，请参阅性能详情的定价和数据留存。	2022 年 7 月 1 日
Amazon Aurora 支持就地升级与 MySQL 兼容的 Aurora Serverless v1	您可以对与 MySQL 5.6 兼容的 Aurora Serverless v1 集群执行就地升级，以将现有集群更改为与 MySQL 5.7 兼容的 Aurora Serverless v1 集群。有关就地升级过程，请参阅修改 Aurora Serverless v1 数据库集群。	2022 年 6 月 16 日
Aurora 支持在 RDS 控制台中开启 Amazon DevOps Guru	您可以从 RDS 控制台中为 Aurora 数据库开启 DevOps Guru 覆盖范围。有关更多信息，请参阅设置 DevOps Guru for RDS。	2022 年 6 月 9 日
Amazon Aurora 支持将事件发布到加密的 Amazon SNS 主题	Amazon Aurora 现在可以将事件发布到启用了服务器端加密 (SSE) 的 Amazon Simple Notification Service (Amazon SNS) 主题，以便为携带敏感数据的事件提供额外保护。有关更多信息，请参阅订阅 Amazon RDS 事件通知。	2022 年 6 月 1 日

Amazon RDS 会将使用情况指标发布到 Amazon CloudWatch	Amazon CloudWatch 中的 `AWS/Usage` 命名空间包括 Amazon RDS 服务配额的账户级使用情况指标。有关更多信息，请参阅 Amazon Aurora 的 Amazon CloudWatch 使用情况指标。	2022 年 4 月 28 日
JSON 格式的数据 API 结果集	`ExecuteStatement` 函数的可选参数导致查询结果集以 JSON 格式的字符串形式返回。JSON 结果集可以简单方便地转换为使用应用程序语言的数据结构。有关更多信息，请参阅处理 JSON 格式的查询结果。	2022 年 4 月 27 日
Amazon Aurora Serverless v2 现已正式发布	Amazon Aurora Serverless v2 通常可供所有用户使用。有关更多信息，请参阅使用 Aurora Serverless v2。	2022 年 4 月 21 日
Aurora MySQL 支持可配置的密码套件	通过 Aurora MySQL，您现在可以使用可配置的密码套件来控制数据库服务器接受的连接加密。有关更多信息，请参阅配置密码套件以连接到 Aurora MySQL 数据库集群。	2022 年 4 月 15 日
Aurora PostgreSQL 支持带有 PostgreSQL 13 的 RDS Proxy	现在，您可以使用 Aurora PostgreSQL 13 数据库集群来创建 RDS Proxy。有关 RDS 代理的更多信息，请参阅使用 Amazon RDS 代理。	2022 年 4 月 22 日

Aurora PostgreSQL 发布说明	现在提供有 Amazon Aurora PostgreSQL 发布说明的单独指南。有关更多信息，请参阅 Aurora PostgreSQL 发布说明。	2022 年 3 月 22 日
Aurora MySQL 发布说明	现在提供有 Amazon Aurora MySQL 发布说明的单独指南。有关更多信息，请参阅 Aurora MySQL 发布说明。	2022 年 3 月 22 日
Aurora PostgreSQL 支持多个主要版本升级	现在，您可以跨多个主要版本执行 Aurora PostgreSQL 数据库集群的版本升级。有关更多信息，请参阅如何执行主要版本升级。	2022 年 3 月 4 日
Aurora PostgreSQL 支持可配置的密码套件	使用 Aurora PostgreSQL 版本 11.8 及更高版本，您现在可以使用可配置的密码套件来控制数据库服务器接受的连接加密。有关将可配置密码套件与 Aurora PostgreSQL 结合使用的信息，请参阅配置密码套件以连接到 Aurora PostgreSQL 数据库集群。	2022 年 3 月 4 日
适用于 MySQL 的 AWS JDBC 驱动程序已正式发布	适用于 MySQL 的 AWS JDBC 驱动程序是专为 Aurora MySQL 的高可用性而设计的客户端驱动程序。适用于 MySQL 的 AWS JDBC 驱动程序现已正式发布。有关更多信息，请参阅连接适用于 MySQL 的 Amazon Web Services JDBC 驱动程序。	2022 年 3 月 2 日

Aurora PostgreSQL 13.5 支持 Babelfish for Aurora PostgreSQL 1.1.0	Aurora PostgreSQL 13.5 支持 Babelfish 1.1.0。有关新功能的列表，请参阅 13.5。有关每个 Babelfish 版本支持的特征列表，请参阅各版本 Babelfish 支持的功能。有关使用情况信息，请参阅使用 Babelfish for Aurora PostgreSQL。	2022 年 2 月 28 日
Amazon Aurora 支持亚太地区（雅加达）区域的数据库活动流	有关更多信息，请参阅数据库活动流的 AWS 区域 支持。	2022 年 2 月 16 日
Performance Insights 支持新的 API 操作	Performance Insights 现在支持以下 API 操作：`GetResourceMetadata` 、`ListAvailableResourceDimensions` 和 `ListAvailableResourceMetrics` 。有关更多信息，请参阅本说明中的使用 Performance Insights API 检索指标以及《Amazon RDS Performance Insights API 参考》。	2022 年 1 月 12 日
Amazon RDS 代理支持事件	RDS 代理现在会生成可供您在 CloudWatch Events 中订阅和查看的事件，或可供您配置后发送到 Amazon EventBridge 的事件。有关更多信息，请参阅使用 RDS Proxy 事件。	2022 年 1 月 11 日

在其他 AWS 区域提供的 RDS Proxy	以下区域现在可提供 RDS 代理：非洲（开普敦）、亚太地区（香港）、亚太地区（大阪）、欧洲（米兰）、欧洲（巴黎）、欧洲（斯德哥尔摩）、中东（巴林）和南美洲（圣保罗）。有关 RDS 代理的更多信息，请参阅使用 Amazon RDS 代理。	2022 年 1 月 5 日
Amazon Aurora 在亚太地区（雅加达）区域中推出	Amazon Aurora 现已在亚太地区（雅加达）区域推出。有关更多信息，请参阅区域和可用区。	2021 年 12 月 13 日
适用于 Amazon RDS 的 DevOps Guru 为 Amazon Aurora 提供了详细的洞察和建议	适用于 RDS 的 DevOps Guru 为性能相关数据挖掘 Performance Insights。使用这些数据，该服务分析 Amazon Aurora 数据库实例的性能，并且可以帮助您解决性能问题。要了解详情，请参阅本指南中的使用适用于 RDS 的 DevOps Guru 分析性能异常和 Amazon DevOps Guru 用户指南中的适用于 RDS 的 DevOps Guru 概览。	2021 年 12 月 1 日
Aurora PostgreSQL 支持带有 PostgreSQL 12 的 RDS 代理	现在，您可以使用 Aurora PostgreSQL 12 数据库集群来创建 RDS 代理。有关 RDS 代理的更多信息，请参阅使用 Amazon RDS 代理。	2021 年 11 月 22 日

Aurora 支持数据库活动流的 AWS Graviton2 实例类	对于 Aurora MySQL 和 Aurora PostgreSQL.，您可以将数据库活动流与 db.r6g 实例类一起使用。有关更多信息，请参阅支持的数据库实例类。	2021 年 11 月 3 日
Amazon Aurora 支持跨账户 AWS KMS keys	在将数据库快照导出到 Amazon S3 时，您可以使用来自不同 AWS 账户的 KMS 密钥进行加密。有关更多信息，请参阅将数据库快照数据导出到 Amazon S3。	2021 年 11 月 3 日
Amazon Aurora 支持 Aurora PostgreSQL 的 Babelfish	Aurora PostgreSQL 的 Babelfish 扩展了您的 Amazon Aurora PostgreSQL 兼容版数据库，能够接受来自 Microsoft SQL Server 客户端的数据库连接。有关更多信息，请参阅使用 Aurora PostgreSQL 的 Babelfish。	2021 年 10 月 28 日
Aurora Serverless v1 可能需要 SSL 进行连接	Aurora Serverless v1 现已支持 PostgreSQL 的 Aurora 集群参数 `force_ssl` 和 MySQL 的 `require_secure_transport` 。有关更多信息，请参阅将 TLS/SSL 与 Aurora Serverless v1 结合使用。	2021 年 10 月 26 日
Amazon Aurora 还在其他 AWS 区域支持 Performance Insights	中东（巴林）、非洲（开普敦）、欧洲（米兰）和亚太地区（大阪）区域现已提供 Performance Insights。有关更多信息，请参阅 Performance Insights 的 AWS 区域支持。	2021 年 10 月 5 日

Aurora Serverless v1 的可配置自动扩展超时	您可以选择 Aurora Serverless v1 等待多长时间以找到弹性伸缩点。如果在此期间未找到弹性伸缩点，Aurora Serverless v1 取消扩展事件或强制更改容量，具体取决于您选择的超时操作。有关更多信息，请参阅 Aurora Serverless v1 的弹性伸缩。	2021 年 9 月 10 日
Aurora 支持 X2g 和 T4g 实例类	Aurora MySQL 和 Aurora PostgreSQL 现在都可以使用 X2g 和 T4g 实例类。您可以使用的实例类取决于 Aurora MySQL 或 Aurora PostgreSQL 的版本。有关所支持实例类型的信息，请参阅数据库实例类。	2021 年 9 月 10 日
Amazon RDS 在共享 VPC 中支持 RDS Proxy	您现在可以在共享 Virtual Private Cloud (VPC) 中创建 RDS Proxy。有关 RDS Proxy 的更多信息，请参阅《Amazon RDS 用户指南》或《Aurora 用户指南》中的“使用 Amazon RDS 代理管理连接”。	2021 年 8 月 6 日
Aurora 版本策略页面	Amazon Aurora 用户指南现在增加了一个部分，其中包含有关 Aurora 版本和相关策略的一般信息。有关详细信息，请参阅 Amazon Aurora 版本。	2021 年 7 月 14 日

从 AWS CloudTrail 跟踪记录中排除 Data API 事件	您可以从 CloudTrail 跟踪记录中排除 Data API 事件。有关更多信息，请参阅从 AWS CloudTrail 跟踪记录中排除 Data API 事件。	2021 年 7 月 2 日
Amazon Aurora PostgreSQL 兼容版支持其他扩展	新支持的扩展包括 pg_bigm、pg_cron、pg_partman 和 pg_proctab。有关更多信息，请参阅 Amazon Aurora PostgreSQL 兼容版的扩展版本。	2021 年 6 月 17 日
克隆 Aurora Serverless 集群	您现在可以创建 Aurora Serverless 的克隆集群。有关克隆的更多信息，请参阅克隆 Aurora 数据库集群卷。	2021 年 6 月 16 日
Aurora 全局数据库可在中国 (北京) 和 中国 (宁夏) 区域使用	现在，您可以在中国 (北京) 和中国 (宁夏) 区域中创建 Aurora 全局数据库。有关 Aurora 全局数据库的信息，请参阅使用 Amazon Aurora Global Database。	2021 年 5 月 19 日
Data API 支持 FIPS 140-2	Data API 支持针对 SSL/TLS 连接的美国联邦信息处理标准出版物 140-2 (FIPS 140-2)。有关更多信息，请参阅 Data API 可用性。	2021 年 5 月 14 日

适用于 PostgreSQL 的 AWS JDBC 驱动程序 (预览版)	适用于 PostgreSQL 的 AWS JDBC 驱动程序现已提供预览版，它是专为 Aurora PostgreSQL 的高可用性而设计的客户端驱动程序。有关详细信息，请参 连接 适用于 PostgreSQL 的 Amazon Web Services JDBC 驱动程序（预览版）。	2021 年 4 月 27 日
其他 AWS 区域中提供的 Data API	数据 API 现已在 亚太区域 (首尔) 和 加拿大 (中部) 区域中可用。有关更多信息，请参阅 Data API 的可用性。	2021 年 4 月 9 日
Amazon Aurora 支持 Graviton2 数据库实例类	现在，您可以使用 Graviton2 数据库实例类 db.r6g.x 创建运行 MySQL 或 PostgreSQL 的数据库集群。有关更多信息，请参阅数据库实例类类型。	2021 年 3 月 12 日
RDS Proxy 端点增强	您可以创建与每个 RDS 代理关联的其他端点。在不同 VPC 中创建端点可实现代理的跨 VPC 访问。Aurora MySQL 集群的代理还可以有只读端点。这些读取器端点连接到集群中的读取器数据库实例，可提高查询密集型应用程序的读取可扩展性和可用性。有关 RDS 代理的更多信息，请参阅 Amazon RDS 用户指南或 Aurora 用户指南中的“使用 Amazon RDS 代理管理连接”。	2021 年 3 月 8 日

Amazon Aurora 在亚太地区（大阪）区域中推出	Amazon Aurora 现已在亚太地区（大阪）区域中推出。有关更多信息，请参阅区域和可用区。	2021 年 3 月 1 日
Aurora PostgreSQL 支持在同一个数据库集群上同时启用 IAM 和 Kerberos 身份验证	Aurora PostgreSQL 现在支持在同一个数据库集群上同时启用 IAM 身份验证和 Kerberos 身份验证 有关更多信息，请参阅 Amazon Aurora 的数据库身份验证。	2021 年 2 月 24 日
Aurora 全局数据库现在支持托管式计划内故障转移	Aurora 全局数据库现在支持托管的计划内故障转移，让您能够更轻松地更改 Aurora 全局数据库的主AWS区域。您只能使用运行正常的 Aurora 全局数据库进行托管的计划内故障转移。要了解更多信息，请参阅灾难恢复和 Amazon Aurora Global Database。有关参考信息，请参阅 Amazon RDS API Reference 中的 `FailoverGlobalCluster` 。	2021 年 2 月 11 日
Aurora Serverless 的 Data API 现在支持更多数据类型	通过 Aurora Serverless 的 Data API，您现在可以将 `UUID` 和 `JSON` 数据类型用作数据库的输入。您现在也可以通过 Aurora Serverless 的 Data API，将从数据库返回的 `LONG` 类型值作为 `STRING` 值。要了解更多信息，请参阅调用 Data API。有关支持的数据类型的参考信息，请参阅 Amazon RDS 数据服务 API 参考 中的 `SqlParameter` 。	2021 年 2 月 2 日

Aurora PostgreSQL 支持将主要版本升级到 PostgreSQL 12	借助 Aurora PostgreSQL，您现在可以将数据库引擎升级到主要版本 12。有关更多信息，请参阅升级 Aurora PostgreSQL 的 PostgreSQL 数据库引擎。	2021 年 1 月 28 日
Aurora MySQL 支持就地升级	您可以将 Aurora MySQL 1.x 集群升级到 Aurora MySQL 2.x，以保留原始集群的数据库实例、端点等。这种就地升级技术通过还原快照避免了设置全新集群带来的不便。它还避免了将所有表数据复制到新集群中的开销。有关更多信息，请参阅将 Aurora MySQL 数据库集群主要版本从 1.x 升级到 2.x。	2021 年 1 月 11 日
适用于 MySQL 的 AWS JDBC 驱动程序 (预览版)	适用于 MySQL 的 AWS JDBC 驱动程序现已提供预览版，它是专为 Aurora MySQL 的高可用性而设计的客户端驱动程序。有关更多信息，请参阅适用于 MySQL 的 Amazon Web Services JDBC 驱动程序（预览版）连接。	2021 年 1 月 7 日
Aurora 在全局数据库的辅助集群上支持数据库活动流	您可以在 Aurora PostgreSQL 或 Aurora MySQL 的主集群或辅助集群上启动数据库活动流。有关支持的引擎版本，请参阅 Aurora 全局数据库的限制。	2020 年 12 月 22 日

包含 4 个数据库实例的多主集群	Aurora MySQL 多主集群中的最大数据库实例数量现在为四个。以前，最大数为两个数据库实例。有关更多信息，请参阅使用 Aurora 多主集群。	2020 年 12 月 17 日
Aurora PostgreSQL 支持 AWS Lambda 函数	您现在可以为您的 Aurora PostgreSQL 数据库集群调用 AWS Lambda 函数。有关更多信息，请参阅从 Aurora PostgreSQL 数据库集群中调用 Lambda 函数。	2020 年 12 月 11 日
Amazon Aurora 支持 Graviton2 数据库实例类预览版	现在，您可以使用 Graviton2 数据库实例类 db.r6g.x 预览版创建运行 MySQL 或 PostgreSQL 的数据库集群。有关更多信息，请参阅数据库实例类类型。	2020 年 12 月 11 日
Amazon Aurora Serverless v2 现已提供预览版。	Amazon Aurora Serverless v2 已提供预览版。要使用 Amazon Aurora Serverless v2，需要申请访问权限。有关更多信息，请参阅 Aurora Serverless v2 页面。	2020 年 12 月 1 日

Aurora PostgreSQL 现在可用于更多 AWS 区域中的 Aurora Serverless。	Aurora PostgreSQL 现在可用于更多 Aurora Serverless 中的 AWS 区域。您现在可以选择在提供 Aurora MySQL Serverless v1 的相同 AWS 区域中运行 Aurora PostgreSQL Serverless v1。具备 Aurora Serverless 支持的其他 AWS 区域包括美国西部（加利福尼亚北部）、亚太地区（新加坡）、亚太地区（悉尼）、亚太地区（首尔）、亚太地区（孟买）、加拿大（中部）、欧洲（伦敦）和欧洲（巴黎）。要查看支持 Aurora Serverless 的所有区域和 Aurora 数据库引擎的列表，请参阅支持 Aurora Serverless v1 的区域和 Aurora 数据库引擎。适用于 Aurora Serverless 的 Amazon RDS Data API 现已在相同 AWS 区域中提供。要查看支持适用于 Aurora Serverless 的数据 API 的所有区域的列表，请参阅适用于 Aurora MySQL Serverless v1 的数据 API	2020 年 11 月 24 日

Amazon RDS Performance Insights 引入了新的维度	您可以根据数据库维度组、应用程序（PostgreSQL）和会话类型（PostgreSQL）对数据库负载进行分组。Amazon RDS 还支持维度 db.name、db.application.name（PostgreSQL）和 db.session_type.name（PostgreSQL）。有关详细信息，请参阅首要负载表。	2020 年 11 月 24 日
Aurora Serverless 支持 Aurora PostgreSQL 版本 10.12	在支持 Aurora PostgreSQL for Aurora Serverless 的AWS区域中，Aurora PostgreSQL for Aurora Serverless 已经升级为 Aurora PostgreSQL 版本 10.12。有关更多信息，请参阅支持 Aurora Serverless v1 的区域和 Aurora 数据库引擎。	2020 年 11 月 4 日
Data API 现在支持基于标签的授权	Data API 支持基于标签的授权。如果您已使用标签标记了您的 RDS 集群资源，则可以在策略语句中使用这些标签来控制通过 Data API 的访问。有关更多信息，请参阅授权访问 Data API。	2020 年 10 月 27 日
Amazon Aurora 扩展了对将快照导出到 Amazon S3 的支持	现在，您可以在所有商业 AWS 区域中将数据库快照数据导出到 Amazon S3。有关更多信息，请参阅将数据库快照数据导出到 Amazon S3。	2020 年 10 月 22 日

Aurora 全局数据库支持克隆	现在，您可以创建 Aurora 全局数据库的主数据库集群和辅助数据库集群的克隆。您可以通过使用 AWS Management Console 并选择创建克隆菜单选项来执行此操作。您也可以使用 AWS CLI 并运行带 `restore-db-cluster-to-point-in-time` 选项的 `--restore-type copy-on-write` 命令。使用AWS Management Console或 AWS CLI，您还可以跨AWS账户克隆 Aurora 全局数据库中的数据库集群。有关克隆的更多信息，请参阅克隆 Aurora 数据库集群卷。	2020 年 10 月 19 日
Amazon Aurora 支持动态调整集群卷的大小	从 Aurora MySQL 1.23 和 2.09 以及 Aurora PostgreSQL 3.3.0 和 Aurora PostgreSQL 2.6.0 开始，在您通过 `DROP TABLE` 之类的操作删除数据后，Aurora 会减小集群卷的大小。要利用此增强功能，请根据集群使用的数据库引擎，升级到适当的版本之一。有关此特征以及如何检查 Aurora 集群的已用和可用存储空间的信息，请参阅管理 Aurora 数据库集群的性能和扩展。	2020 年 10 月 13 日

Amazon Aurora 支持的最大卷大小可达 128 TiB	新的和现有的 Aurora 集群卷现在可以增长到最大大小 128 tebibytes (TiB)。有关更多信息，请参阅 Aurora 存储增长方式。	2020 年 9 月 22 日
Aurora PostgreSQL 在中国 (宁夏) 区域中支持 db.r5 和 db.t3 数据库实例类	您现在可以在中国 (宁夏) 区域中创建使用 db.r5 和 db.t3 数据库实例类的 Aurora PostgreSQL 数据库集群。有关更多信息，请参阅数据库实例类。	2020 年 9 月 3 日

Aurora 并行查询增强功能	从 Aurora MySQL 2.09 和 1.23 开始，您可以利用并行查询特征的增强功能。创建并行查询集群不再需要特殊的引擎模式。现在，您可以对运行兼容 Aurora MySQL 版本的任何已预置集群使用 `aurora_parallel_query` 配置选项来打开和关闭并行查询。您可以将现有集群升级到兼容的 Aurora MySQL 版本并使用并行查询，而不是创建新集群并将数据导入到其中。您可以将 Performance Insights 用于并行查询集群。您可以停止和启动并行查询集群。您可以创建与 MySQL 5.7 兼容的 Aurora 并行查询集群。并行查询适用于使用 `DYNAMIC` 行格式的表。并行查询集群可以使用 AWS Identity and Access Management（IAM）身份验证。并行查询集群中的读取器数据库实例可以利用 `READ COMMITTED` 隔离级别。现在，还可以在其他 AWS 区域中创建并行查询集群。有关并行查询功能和这些增强功能的更多信息，请参阅 Aurora MySQL 的并行查询。	2020 年 9 月 2 日
对 Aurora MySQL 参数 binlog_rows_query_log_events 的更改	您现在可以更改 Aurora MySQL 配置参数 `binlog_rows_query_log_events` 的值。以前，此参数不可修改。	2020 年 8 月 26 日

支持 Aurora MySQL 的自动次要版本升级	如果使用 Aurora MySQL，则为 Aurora MySQL 数据库集群指定启用自动次要版本升级设置后，该设置即生效。启用自动次要版本升级后，Aurora 会在新的次要版本发布时自动升级到该版本。自动升级会在数据库的维护时段进行。对于 Aurora MySQL，此特征仅适用于与 MySQL 5.7 兼容的 Aurora MySQL 版本 2。最初，自动升级过程将 Aurora MySQL 数据库集群升级到版本 2.07.2。有关此特征如何与 Aurora MySQL 配合使用的更多信息，请参阅 Amazon Aurora MySQL 的数据库升级和补丁。	2020 年 8 月 3 日
Aurora PostgreSQL 支持将主要版本升级到 PostgreSQL 版本 11	借助 Aurora PostgreSQL，您现在可以将数据库引擎升级到主要版本 11。有关更多信息，请参阅升级 Aurora PostgreSQL 的 PostgreSQL 数据库引擎。	2020 年 7 月 28 日
Amazon Aurora 支持 AWS PrivateLink	Amazon Aurora 现在支持为 Amazon RDS API 调用创建 Amazon VPC 端点，以在 AWS 网络中保持应用程序和 Aurora 之间的流量。有关更多信息，请参阅 Amazon Aurora 和接口 VPC 端点 (AWS PrivateLink)。	2020 年 7 月 9 日

RDS Proxy 正式发布	RDS 代理现已正式发布。您可以将 RDS 代理与 RDS for MySQL、Aurora MySQL、RDS for PostgreSQL 和 Aurora PostgreSQL 一起用于生产工作负载。有关 RDS 代理的更多信息，请参阅 Amazon RDS 用户指南或 Aurora 用户指南中的“使用 Amazon RDS 代理管理连接”。	2020 年 6 月 30 日
Aurora 全局数据库写入转发	现在，您可以在全局数据库中的辅助集群上启用写入功能。使用写入转发功能，您可以在辅助集群上发出 DML 语句，Aurora 将写操作转发到主集群，并将更新的数据复制到所有辅助集群。有关更多信息，请参阅具有 Aurora 全局数据库的辅助 AWS 区域的写入转发。	2020 年 6 月 18 日
Aurora 支持与 AWS Backup 的集成	您可以使用 AWS Backup 管理 Aurora 数据库集群的备份。有关更多信息，请参阅备份和还原 Aurora 数据库集群概述。	2020 年 6 月 10 日
Aurora PostgreSQL 支持 db.t3.large 数据库实例类	您现在可以创建使用 db.t3.large 数据库实例类的 Aurora PostgreSQL 数据库集群。有关更多信息，请参阅数据库实例类。	2020 年 6 月 5 日

Aurora 全局数据库支持 PostgreSQL 版本 11.7 和托管式恢复点目标（RPO）	您现在可以为 PostgreSQL 数据库引擎版本 11.7 创建 Aurora 全局数据库。您还可以使用恢复点目标 (RPO) 管理 PostgreSQL 全局数据库如何从故障中恢复。有关更多信息，请参阅 Aurora 全局数据库的跨区域灾难恢复。	2020 年 6 月 4 日
Aurora MySQL 支持使用数据库活动流进行数据库监控	Aurora MySQL 现在包括数据库活动流，数据库活动流可提供关系数据库中数据库活动的近实时数据流。有关更多信息，请参阅使用数据库活动流。	2020 年 6 月 2 日
查询编辑器已在其他 AWS 区域中提供	适用于 Aurora Serverless 的查询编辑器现已在其他 AWS 区域中提供。有关更多信息，请参阅查询编辑器的可用性。	2020 年 5 月 28 日
Data API 已在其他 AWS 区域中提供	Data API 现已在其他 AWS 区域中提供。有关更多信息，请参阅 Data API 的可用性。	2020 年 5 月 28 日
RDS Proxy 已在加拿大（中部）区域中提供	现在，您可以在加拿大（中部）区域中使用 RDS Proxy 预览版。有关 RDS 代理的更多信息，请参阅使用 Amazon RDS 代理管理连接（预览版）。	2020 年 5 月 28 日

Aurora 全局数据库和跨区域只读副本	对于 Aurora 全局数据库，您现在可以从与辅助集群处于相同区域中的主集群创建 Aurora MySQL 跨区域只读副本。有关 Aurora 全局数据库和跨区域只读副本的更多信息，请参阅使用 Amazon Aurora Global Database 和复制 Amazon Aurora MySQL 数据库。	2020 年 5 月 18 日
RDS Proxy 已在更多 AWS 区域中提供	现在，您可在美国西部（加利福利亚北部）区域、欧洲（伦敦）区域、欧洲（法兰克福）区域、亚太区域（首尔）、亚太地区（孟买）区域、亚太区域（新加坡）和亚太区域（悉尼）中使用 RDS 代理预览。有关 RDS 代理的更多信息，请参阅使用 Amazon RDS 代理管理连接（预览版）。	2020 年 5 月 13 日
Aurora PostgreSQL 兼容版支持本地或自托管的 Microsoft Active Directory	现在，当用户连接到您的 Aurora PostgreSQL 数据库集群时，您可以使用本地或自行托管的 Active Directory for Kerberos 进行身份验证。有关更多信息，请参阅对 Aurora PostgreSQL 使用 Kerberos 身份验证。	2020 年 5 月 7 日
Aurora MySQL 多主机集群可在更多 AWS 区域区域中使用	现在，您可以在 亚太区域（首尔）、亚太区域（东京）、亚太地区（孟买）区域和欧洲（法兰克福）区域中创建 Aurora 多主集群。有关多主集群的更多信息，请参阅使用 Aurora 多主集群。	2020 年 5 月 7 日

Performance Insights 支持分析正在运行的 Aurora MySQL 查询的统计信息	现在，您可以使用 Performance Insights 分析正在对 Aurora MySQL 数据库实例运行的查询的统计信息。有关更多信息，请参阅分析正在运行的查询的统计信息。	2020 年 5 月 5 日
适用于 Data API 的 Java 客户端库正式发布	适用于 Data API 的 Java 客户端库现已正式发布。您可以下载并使用适用于 Data API 的 Java 客户端库。它可让您将客户端类映射到 Data API 的请求和响应。有关更多信息，请参阅使用适用于 Data API 的 Java 客户端库。	2020 年 4 月 30 日
Amazon Aurora 在欧洲地区（米兰）区域中推出	Amazon Aurora 现已在欧洲地区（米兰）区域中推出。有关更多信息，请参阅区域和可用区。	2020 年 4 月 28 日
Amazon Aurora 在欧洲地区（米兰）区域中推出	Amazon Aurora 现已在欧洲地区（米兰）区域中推出。有关更多信息，请参阅区域和可用区。	2020 年 4 月 27 日
Amazon Aurora 在非洲（开普敦）区域中推出	Amazon Aurora 现已在非洲（开普敦）区域中推出。有关更多信息，请参阅区域和可用区。	2020 年 4 月 22 日

Aurora PostgreSQL 现在支持 db.r5.16xlarge 和 db.r5.8xlarge 数据库实例类	您现在可以创建运行 PostgreSQL 的 Aurora PostgreSQL 数据库集群，这些数据库集群使用 db.r5.16xlarge 和 db.r5.8xlarge 数据库实例类。有关更多信息，请参阅 Aurora 数据库实例类的硬件规格。	2020 年 4 月 8 日
适用于 PostgreSQL 的 Amazon RDS Proxy	Amazon RDS 代理现在可用于 PostgreSQL。您可以使用 RDS 代理减少集群上连接管理的开销，并降低“连接过多”错误的可能性。RDS 代理目前提供针对 PostgreSQL 的公共预览版。有关更多信息，请参阅使用 Amazon RDS 代理管理连接（预览版）。	2020 年 4 月 8 日
Aurora 全局数据库现在支持 Aurora PostgreSQL	您现在可以为 PostgreSQL 数据库引擎创建 Aurora 全局数据库。Aurora 全局数据库跨多个 AWS 区域，实现低延迟的全球读取以及从区域范围的停机中进行灾难恢复。有关更多信息，请参阅使用 Amazon Aurora Global Database。	2020 年 3 月 10 日
支持 Aurora PostgreSQL 的主要版本升级	借助 Aurora PostgreSQL，您现在可以将数据库引擎升级到主要版本。这样做，您可以在升级所选 PostgreSQL 引擎版本时跳至较新的主要版本。有关更多信息，请参阅升级 Aurora PostgreSQL 的 PostgreSQL 数据库引擎。	2020 年 3 月 4 日

Aurora PostgreSQL 支持 Kerberos 身份验证	现在，当用户连接到您的 Aurora PostgreSQL 数据库集群时，您可以使用 Kerberos 身份验证来验证用户的身份。有关更多信息，请参阅对 Aurora PostgreSQL 使用 Kerberos 身份验证。	2020 年 2 月 28 日
Data API 支持 AWS PrivateLink	Data API 现在支持为 Data API 调用创建 Amazon VPC 端点，以在AWS网络中保持应用程序和 Data API 之间的流量。有关更多信息，请参阅为 Data API 创建 Amazon VPC 端点（ AWS PrivateLink ）。	2020 年 2 月 6 日
Aurora PostgreSQL 中支持 Aurora 机器学习	`aws_ml` Aurora PostgreSQL 扩展提供了的函数可供您在数据库查询中用来调用 Amazon Comprehend 以进行情绪分析，并调用 SageMaker 以运行您自己的机器学习模型。有关更多信息，请参阅在 Aurora 中使用机器学习 (ML) 功能。	2020 年 2 月 5 日
Aurora PostgreSQL 支持将数据导出到 Amazon S3	您可以从 Aurora PostgreSQL 数据库集群中查询数据，并将数据直接导出到存储在 Amazon S3 存储桶中的文件中。有关更多信息，请参阅将数据从 Aurora PostgreSQL 数据库集群导出到 Amazon S3。	2020 年 2 月 5 日

支持将数据库快照数据导出到 Amazon S3	Amazon Aurora 支持将数据库快照数据导出到 Amazon S3 for MySQL 和 PostgreSQL。有关更多信息，请参阅将数据库快照数据导出到 Amazon S3。	2020 年 1 月 9 日
在文档历史记录中包含 Aurora MySQL 发布说明	此部分现在包括 2018 年 8 月 31 日之后所发布版本的 Aurora MySQL 兼容版 发布说明的历史记录条目。有关特定版本的完整发行说明，请选择历史记录条目第一列中的链接。	2019 年 12 月 13 日
Amazon RDS Proxy	使用 Amazon RDS 代理可以减少集群上连接管理的开销，并降低“连接过多”错误的可能性。请将每个代理与一个 RDS 数据库实例或 Aurora 数据库集群进行关联。然后，请在应用程序的连接字符串中使用代理端点。Amazon RDS 代理当前是公开预览版。它支持 Aurora MySQL 数据库引擎。有关更多信息，请参阅使用 Amazon RDS 代理管理连接（预览版）。	2019 年 12 月 3 日
Aurora Serverless v1 的 Data API 支持数据类型映射提示	您现在可以使用提示来指示 Aurora Serverless v1 的数据 API 将 `String` 值作为不同的类型发送到数据库。有关更多信息，请参阅调用 Data API。	2019 年 11 月 26 日

Aurora Serverless v1 的 Data API 支持 Java 客户端库 (预览版)	您可以下载并使用适用于 Data API 的 Java 客户端库。它可让您将客户端类映射到 Data API 的请求和响应。有关更多信息，请参阅使用适用于 Data API 的 Java 客户端库。	2019 年 11 月 26 日
Aurora PostgreSQL 符合 FedRAMP HIGH 要求	Aurora PostgreSQL 符合 FedRAMP HIGH 要求。有关 AWS 和合规性工作的详细信息，请参阅 AWS 按合规性计划提供的范围内服务。	2019 年 11 月 26 日
对 Amazon Aurora MySQL 副本启用 READ COMMITTED 隔离级别	现在可以对 Aurora MySQL 副本启用 `READ COMMITTED` 隔离级别。这样做需要在会话级别启用 `aurora_read_replica_read_committed_isolation_enabled` 配置设置。将 `READ COMMITTED` 隔离级别用于 OLTP 集群上长时间运行的查询可以帮助解决历史记录列表长度的问题。启用此设置之前，请务必了解 Aurora 副本上的隔离行为与 `READ COMMITTED` 的常规 MySQL 实现有何不同。有关更多信息，请参阅 Aurora MySQL 隔离级别。	2019 年 11 月 25 日
Performance Insights 支持分析正在运行的 Aurora PostgreSQL 查询的统计信息	现在可以使用 Aurora PostgreSQL 数据库实例的 Performance Insights 分析正在运行的查询的统计信息。有关更多信息，请参阅分析正在运行的查询的统计信息。	2019 年 11 月 25 日

Aurora 全局数据库中包含更多集群	现在可以向 Aurora 全球数据库中添加多个辅助区域。您可以在更广泛的地理区域内利用低延迟全局读取和灾难恢复。有关 Aurora 全局数据库的更多信息，请参阅使用 Amazon Aurora Global Database。	2019 年 11 月 25 日
Aurora MySQL 支持 Aurora 机器学习	在 Aurora MySQL 2.07 及更高版本中，您可以调用 Amazon Comprehend 执行情绪分析，调用 SageMaker 执行各种机器学习算法。通过在查询中嵌入对存储函数的调用，可以直接在数据库应用程序中使用结果。有关更多信息，请参阅在 Aurora 中使用机器学习 (ML) 功能。	2019 年 11 月 25 日
Aurora 全局数据库不再需要引擎模式设置	创建旨在成为 Aurora 全球数据库一部分的集群时，您不再需要指定 `--engine-mode=global` 。所有满足兼容性要求的 Aurora 集群都有资格成为全球数据库的一部分。例如，集群当前必须使用与 MySQL 5.6 兼容的 Aurora MySQL 版本 1。有关 Aurora 全局数据库的信息，请参阅使用 Amazon Aurora Global Database。	2019 年 11 月 25 日

Aurora 全局数据库可用于 Aurora MySQL 版本 2	自 Aurora MySQL 2.07 版起，您可以创建与 MySQL 5.7 兼容的 Aurora 全局数据库。您无需为主集群或辅助集群指定 `global` 引擎模式。您可以将 Aurora MySQL 2.07 或更高版本的任何新预置的集群添加到 Aurora 全球数据库。有关 Aurora 全局数据库的信息，请参阅使用 Amazon Aurora Global Database。	2019 年 11 月 25 日
无需实验室模式即可进行 Aurora MySQL 热行争用优化	热行争用优化现在正式用于 Aurora MySQL，无需将 Aurora 实验室模式设置为 ON。在很多事务争用同一页面上的行时，该特征大大提高了工作负载的吞吐量。该改进涉及更改 Aurora MySQL 使用的锁定释放算法。	2019 年 11 月 19 日
无需实验室模式即可进行 Aurora MySQL 哈希联接	哈希联接特征现在正式用于 Aurora MySQL，无需将 Aurora 实验室模式设置为 ON。在需要使用 equijoin 联接大量数据时，该特征可以提高查询性能。有关使用该特征的更多信息，请参阅在 Aurora MySQL 中使用哈希联接。	2019 年 11 月 19 日
Aurora MySQL 2.* 支持更多 db.r5 实例类	Aurora MySQL 集群现在支持实例类型 db.r5.8xlarge、db.r5.16xlarge 和 db.r5.24xlarge。有关 Aurora MySQL 集群实例类型的更多信息，请参阅选择数据库实例类。	2019 年 11 月 19 日

Aurora MySQL 2.* 支持回溯	Aurora MySQL 2.* 版现在可以快速从用户错误中恢复，例如删除错误的表或错误的行。使用回溯可以将数据库回退到以前的某个时间点，无需从备份还原，即使是大型数据库也只需要几秒钟时间。有关详细信息，请参阅回溯 Aurora 数据库集群。	2019 年 11 月 19 日
Aurora 支持账单标签	您现在可以使用标签跟踪资源的成本分配，例如，Aurora 集群、Aurora 集群中的数据库实例、I/O、备份和快照等。您可以使用 AWS Cost Explorer 查看与每个标签关联的成本。有关将标签与 Aurora 一起使用的更多信息，请参阅标记 Amazon RDS 资源。有关标签以及使用标签进行成本分析的方法的基本信息，请参阅使用成本分配标签和用户定义的成本分配标签。	2019 年 10 月 23 日
适用于 Aurora PostgreSQL 的 Data API	Aurora PostgreSQL 现在支持将数据 API 与 Amazon Aurora Serverless v1 数据库集群一起使用。有关更多信息，请参阅使用适用于 Aurora Serverless v1 的数据 API。	2019 年 9 月 23 日

Aurora PostgreSQL 支持将数据库日志上传到 CloudWatch Logs	您可以配置 Aurora PostgreSQL 数据库集群以将日志数据发布到 Amazon CloudWatch Logs 中的日志组。利用 CloudWatch Logs，可以对日志数据进行实时分析并使用 CloudWatch 创建警报和查看指标。您可以使用 CloudWatch Logs 在高持久性存储中存储日志记录。有关更多信息，请参阅将 Aurora PostgreSQL 日志发布到 Amazon CloudWatch Logs。	2019 年 8 月 9 日
Aurora MySQL 的多主集群	您可以设置 Aurora MySQL 多主集群。在这些集群中，每个数据库实例都具有读/写功能。有关更多信息，请参阅使用 Aurora 多主集群。	2019 年 8 月 8 日
Aurora PostgreSQL 支持 Aurora Serverless v1	现在可以将 Amazon Aurora Serverless v1 与 Aurora PostgreSQL 一起使用。Aurora Serverless 数据库集群会根据您应用程序的需求自动启动、关闭以及扩展或缩减计算容量。有关更多信息，请参阅使用 Amazon Aurora Serverless v1。	2019 年 7 月 9 日

适用于 Aurora MySQL 的跨账户克隆	您现在可以在AWS账户之间克隆 Aurora MySQL 数据库集群的集群卷。您可以通过 AWS Resource Access Manager (AWS RAM) 授权共享。克隆的集群卷使用写入时复制机制，它只需要额外的存储空间来存储新数据或更改后的数据。有关适用于 Aurora 的克隆的更多信息，请参阅克隆 Aurora 数据库集群中的数据库。	2019 年 7 月 2 日
Aurora PostgreSQL 支持 db.t3 数据库实例类	您现在可以创建使用 db.t3 数据库实例类的 Aurora PostgreSQL 数据库集群。有关更多信息，请参阅数据库实例类。	2019 年 6 月 20 日
支持从 Amazon S3 为 Aurora PostgreSQL 导入数据	现在，您可以将数据从 Amazon S3 文件导入至 Aurora PostgreSQL 数据库集群中的表。有关更多信息，请参阅将 Amazon S3 数据导入至 Aurora PostgreSQL 数据库集群。	2019 年 6 月 19 日
Aurora PostgreSQL 现在通过集群缓存管理提供快速故障转移恢复	Aurora PostgreSQL 现在提供集群缓存管理来确保进行故障转移时快速恢复主数据库实例。有关更多信息，请参阅通过集群缓存管理提供故障转移后的快速恢复。	2019 年 6 月 11 日

适用于 Aurora Serverless v1 的 Data API 已正式发布	您可以使用 Data API，通过基于 Web 服务的应用程序访问 Aurora Serverless v1 集群。有关更多信息，请参阅使用适用于 Aurora Serverless v1 的数据 API。	2019 年 5 月 30 日
Aurora PostgreSQL 支持使用数据库活动流进行数据库监控	Aurora PostgreSQL 现在包括数据库活动流，数据库活动流可提供关系数据库中数据库活动的近实时数据流。有关更多信息，请参阅使用数据库活动流。	2019 年 5 月 30 日
Amazon Aurora 建议	Amazon Aurora 现在可为 Aurora 资源提供自动建议。有关更多信息，请参阅使用 Amazon Aurora 建议。	2019 年 5 月 22 日
Performance Insights 支持 Aurora 全局数据库	您现在可以通过 Aurora 全局数据库使用 Performance Insights。有关 Aurora 的 Performance Insights 的信息，请参阅使用 Amazon RDS Performance Insights。有关 Aurora 全局数据库的信息，请参阅使用 Aurora 全局数据库。	2019 年 5 月 13 日
Performance Insights 可用于 Aurora MySQL 5.7	Amazon RDS Performance Insights 现在可用于 Aurora MySQL 2.x 版本，其与 MySQL 5.7 兼容。有关更多信息，请参阅使用 Amazon RDS Performance Insights。	2019 年 5 月 3 日

在更多 AWS 区域中提供 Aurora 全局数据库	现在，您可以在大多数 Aurora 可用的 AWS 区域中创建 Aurora 全局数据库。有关 Aurora 全局数据库的信息，请参阅使用 Amazon Aurora Global Database。	2019 年 4 月 30 日
Aurora Serverless v1 的最小容量是 1	可以用于 Aurora Serverless v1 集群的最小容量设置是 1。以前，最小容量是 2。有关指定 Aurora Serverless 容量值的信息，请参阅设置 Aurora Serverless v1 数据库集群的容量。	2019 年 4 月 29 日
Aurora Serverless v1 超时操作	您现在可以指定 Aurora Serverless v1 容量更改超时时要采取的操作。有关更多信息，请参阅容量更改的超时操作。	2019 年 4 月 29 日
按秒计费	现在，Amazon RDS 在除 AWS GovCloud（美国）以外的所有 AWS 区域对按需实例按 1 秒增量计费。有关更多信息，请参阅 Aurora 的数据库实例计费。	2019 年 4 月 25 日
跨 AWS 区域共享 Aurora Serverless v1 快照	使用 Aurora Serverless v1 时，快照会始终加密。如果您使用自己的 AWS KMS key 为快照加密，您现在可以跨 AWS 区域复制或共享快照。有关 Aurora Serverless v1 数据库集群快照的信息，请参阅Aurora Serverless v1 和快照。	2019 年 4 月 17 日

从 Amazon S3 中还原 MySQL 5.7 备份	您现在可以创建 MySQL 5.7 数据库的备份，将其存储在 Amazon S3 上，然后将该备份文件还原到新 Aurora MySQL 数据库集群。有关更多信息，请参阅从外部 MySQL 数据库将数据迁移到 Aurora MySQL 数据库集群。	2019 年 4 月 17 日
跨区域共享 Aurora Serverless v1 快照	使用 Aurora Serverless v1 时，快照会始终加密。如果您用自己的 AWS KMS key 为快照加密，您现在可以跨各个区域复制或共享快照。有关 Aurora Serverless v1 数据库集群的快照的信息，请参阅 Aurora Serverless 和快照。	2019 年 4 月 16 日
Aurora 概念验证教程	您可以了解到如何执行概念验证来通过 Aurora 试用您的应用程序和工作负载。要查看完整教程，请参阅执行 Aurora 概念验证。	2019 年 4 月 16 日
Aurora Serverless v1 支持从 Amazon S3 备份还原	现在可以将备份从 Amazon S3 导入到 Aurora Serverless 集群。有关该程序的详细信息，请参阅使用 Amazon S3 存储桶从 MySQL 迁移数据。	2019 年 4 月 16 日

Aurora Serverless v1 的新可修改参数	您现在可以修改 Aurora Serverless v1 集群的以下数据库参数：innodb_file_format 、innodb_file_per_table 、innodb_large_prefix 、innodb_lock_wait_timeout 、innodb_monitor_disable 、innodb_monitor_enable 、innodb_monitor_reset 、innodb_monitor_reset_all 、innodb_print_all_deadlocks 、log_warnings 、net_read_timeout 、net_retry_count 、net_write_timeout 、sql_mode 和 tx_isolation 。有关 Aurora Serverless v1 集群的配置参数的更多信息，请参阅Aurora Serverless v1和参数组。	2019 年 4 月 4 日
Aurora PostgreSQL 支持 db.r5 数据库实例类	您现在可以创建使用 db.r5 数据库实例类的 Aurora PostgreSQL 数据库集群。有关更多信息，请参阅数据库实例类。	2019 年 4 月 4 日

Aurora PostgreSQL 逻辑复制	您现在可以使用 PostgreSQL 逻辑复制来复制 Aurora PostgreSQL 数据库集群的数据库的各个部分。有关更多信息，请参阅使用 PostgreSQL 逻辑复制。	2019 年 3 月 28 日
GTID 支持 Aurora MySQL 2.04	您现在可以通过 MySQL 5.7 的全局事务 ID (GTID) 特征使用复制。此特征简化了 Aurora MySQL 与外部 MySQL 5.7 兼容数据库之间的二进制日志 (binlog) 复制的执行。复制可以将 Aurora MySQL 集群用作源或目标。此特征适用于 Aurora MySQL 2.04 及更高版本。有关基于 GTID 的复制和 Aurora MySQL 的更多信息，请参阅将基于 GTID 的复制用于 Aurora MySQL。	2019 年 3 月 25 日
将 Aurora Serverless v1 日志上传到 Amazon CloudWatch	您现在可以让 Aurora 将数据库日志上传到 Aurora Serverless v1 集群的 CloudWatch。有关更多信息，请参阅查看 Aurora Serverless 数据库集群。作为此增强功能的一部分，您现在可以为数据库集群参数组中的实例级参数定义值，除非您在数据库参数组中覆盖这些值，否则它们将应用于集群中的所有数据库实例。有关更多信息，请参阅使用数据库参数组和数据库集群参数组。	2019 年 2 月 25 日

Aurora MySQL 支持 db.t3 数据库实例类	您现在可以创建使用 db.t3 数据库实例类的 Aurora MySQL 数据库集群。有关更多信息，请参阅数据库实例类。	2019 年 2 月 25 日
Aurora MySQL 支持 db.r5 数据库实例类	您现在可以创建使用 db.r5 数据库实例类的 Aurora MySQL 数据库集群。有关更多信息，请参阅数据库实例类。	2019 年 2 月 25 日
Aurora MySQL 的 Performance Insights 计数器	现在，您可以将性能计数器添加到 Aurora MySQL 数据库实例的 Performance Insights 图表。有关更多信息，请参阅性能详情控制面板组件。	2019 年 2 月 19 日
Amazon RDS Performance Insights 支持查看 Aurora MySQL 的更多 SQL 文本	Amazon RDS Performance Insights 现在支持在 Aurora MySQL 数据库实例的 Performance Insights 控制面板中查看更多 SQL 文本。有关更多信息，请参阅在性能详情控制面板中查看更多 SQL 文本。	2019 年 2 月 6 日
Amazon RDS Performance Insights 支持查看 Aurora PostgreSQL 的更多 SQL 文本	Amazon RDS Performance Insights 现在支持在 Aurora PostgreSQL 数据库实例的 Performance Insights 控制面板中查看更多 SQL 文本。有关更多信息，请参阅在性能详情控制面板中查看更多 SQL 文本。	2019 年 1 月 24 日

Aurora 备份计费	您可以使用 Amazon CloudWatch 指标 `TotalBackupStorageBilled` 、`SnapshotStorageUsed` 和 `BackupRetentionPeriodStorageUsed` 来监控您的 Aurora 备份空间使用情况。有关如何使用 CloudWatch 指标的更多信息，请参阅监控概览。有关如何管理备份数据存储的更多信息，请参阅了解 Aurora 备份存储使用量。	2019 年 1 月 3 日
Performance Insights 计数器	现在，您可以将性能计数器添加到 Performance Insights 图表。有关更多信息，请参阅性能详情控制面板组件。	2018 年 12 月 6 日
Aurora 全局数据库	您现在可以创建和下载 Aurora 全局数据库。Aurora 全局数据库跨多个 AWS 区域，实现低延迟的全球读取以及从区域范围的停机中进行灾难恢复。有关更多信息，请参阅使用 Amazon Aurora Global Database。	2018 年 11 月 28 日
Aurora PostgreSQL 中的查询计划管理	Aurora PostgreSQL 现在提供查询计划管理，您可用它来管理 PostgreSQL 查询执行计划。有关更多信息，请参阅管理 Aurora PostgreSQL 的查询执行计划。	2018 年 11 月 20 日

Aurora Serverless v1 的查询编辑器 (测试版)	您可以在 Aurora Serverless v1 集群上的 Amazon RDS 控制台中运行 SQL 语句。有关更多信息，请参阅使用 Aurora Serverless v1 的查询编辑器。	2018 年 11 月 20 日
Aurora Serverless v1 的 Data API (测试版)	您可以使用 Data API，通过基于 Web 服务的应用程序访问 Aurora Serverless v1 集群。有关更多信息，请参阅使用适用于 Aurora Serverless 的 Data API。	2018 年 11 月 20 日
Aurora Serverless v1 的 TLS 支持	Aurora Serverless v1 集群支持 TLS/SSL 加密。有关更多信息，请参阅 Aurora Serverless 的 TLS/SSL。	2018 年 11 月 19 日
自定义端点	现在，您可以创建与任意数据库实例集相关联的端点。在一些数据库实例具有与其他实例不同的容量或配置时，此特征有助于为 Aurora 集群实现负载均衡和高可用性。您可以使用自定义端点，而不是通过实例端点连接到特定数据库实例。有关更多信息，请参阅 Amazon Aurora 端点。	2018 年 11 月 12 日
Aurora PostgreSQL 中的 IAM 身份验证支持	Aurora PostgreSQL 现在支持 IAM 身份验证。有关更多信息，请参阅 IAM 数据库身份验证。	2018 年 11 月 8 日

用于还原和时间点恢复的自定义参数组	您现在可以在还原快照或执行时间点恢复操作时指定自定义参数组。有关更多信息，请参阅从数据库集群快照还原和将数据库集群还原到指定时间。	2018 年 10 月 15 日
Aurora 数据库集群的删除保护	如果为数据库集群启用删除保护，则任何用户都无法删除数据库。有关更多信息，请参阅删除数据库集群。	2018 年 9 月 26 日
停止/启动特征 Aurora	您现在可以通过单个操作停止或启动整个 Aurora 集群。有关更多信息，请参阅停止和启动 Aurora 集群。	2018 年 9 月 24 日
Aurora MySQL 的并行查询特征	Aurora MySQL 现在提供了一个选项，以并行处理 Aurora 存储基础设施中的查询的 I/O 负载。该特征加快了数据密集型分析查询，这通常是工作负载中的最耗时操作。有关更多信息，请参阅 Aurora MySQL 的并行查询。	2018 年 9 月 20 日
新指南	这是 Amazon Aurora 用户指南的首个版本。	2018 年 8 月 31 日

AWS 术语表

有关最新的 AWS 术语，请参阅 AWS 词汇表 参考中的 AWS 词汇表。